

# Предобработка данных

Все в науке о данных начинается с данных.

**Предобработка данных является важнейшим этапом**, и если она не будет выполнена, то дальнейший анализ в большинстве случаев невозможен из-за того, что аналитические алгоритмы просто не смогут работать или результаты их работы будут некорректными. Иными словами, реализуется принцип:

**GIGO — garbage in, garbage out** (мусор на входе, мусор на выходе).

Из книги Висснер-Гросс, Александр (2016). Наборы данных важнее алгоритмов.

Year	Breakthroughs in AI	Datasets (First Available)	Algorithms (First Proposed)
1994	Human-level spontaneous speech recognition	Spoken Wall Street Journal articles and other texts (1991)	Hidden Markov Model (1984)
1997	IBM Deep Blue defeated Garry Kasparov	700,000 Grandmaster chess games, aka “The Extended Book” (1991)	Negascout planning algorithm (1983)
2005	Google’s Arabic- and Chinese-to-English translation	1.8 trillion tokens from Google Web and News pages (collected in 2005)	Statistical machine translation algorithm (1988)
2011	IBM Watson became the world Jeopardy! champion	8.6 million documents from Wikipedia, Wiktionary, Wikiquote, and Project Gutenberg (updated in 2010)	Mixture-of-Experts algorithm (1991)
2014	Google’s GoogLeNet object classification at near-human performance	ImageNet corpus of 1.5 million labeled images and 1,000 object categories (2010)	Convolution neural network algorithm (1989)
2015	Google’s Deepmind achieved human parity in playing 29 Atari games by learning general control from video	Arcade Learning Environment dataset of over 50 Atari games (2013)	Q-learning algorithm (1992)
<b>Average No. of Years to Breakthrough:</b>		<b>3 years</b>	<b>18 years</b>

СРЕДНЕЕ ВРЕМЯ МЕЖДУ ПРЕДЛОЖЕНИЯМИ КЛЮЧЕВЫХ АЛГОРИТМОВ И СООТВЕТСТВУЮЩИМИ УЛУЧШЕНИЯМИ СОСТАВИЛО ОКОЛО 18 ЛЕТ, ТОГДА КАК СРЕДНЕЕ ВРЕМЯ МЕЖДУ ДОСТУПНОСТЬЮ КЛЮЧЕВЫХ НАБОРОВ ДАННЫХ И СООТВЕТСТВУЮЩИМИ УЛУЧШЕНИЯМИ БЫЛО МЕНЕЕ 3 ЛЕТ, ИЛИ ПРИМЕРНО В 6 РАЗ БЫСТРЕЕ.

Если это правда, эта гипотеза имеет фундаментальное значение для будущего прогресса в области ИИ. Например, уделение приоритетного внимания обработке высококачественных обучающих наборов данных может позволить на порядок ускорить прорывы в области ИИ по сравнению с чисто алгоритмическими достижениями. В конце концов, **сосредоточиться на наборе данных, а не на алгоритме, — потенциально более простой подход.**

# **По оценкам специалистов 80% времени уходит на подготовку данных!**

Среди проблем, вызывающих снижение качества данных, можно выделить следующие:

- пропущенные значения;
- дубликаты;
- противоречия;
- аномальные значения и выбросы;
- шум;
- некорректные форматы и представления данных;
- ошибки ввода данных.

Разберем методы поиска и исправления:

1. отсутствующих данных;
2. нетипичных данных – выбросов;
3. неинформативных данных – дубликатов;
4. несогласованных данных – одних и тех же данных, представленных в разных регистрах или форматах.

	0	1	2	3	4	5	6	7
0	6	148	72	35	0	33.6	0.627	50
1	1	85	66	29	0	26.6	0.351	31
2	8	183	64	0	0	23.3	0.672	32
3	1	89	66	23	94	28.1	0.167	21
4	0	137	40	35	168	43.1	2.288	33
5	5	116	74	0	0	25.6	0.201	30
6	3	78	50	32	88	31.0	0.248	26
7	10	115	0	0	0	35.3	0.134	29
8	2	197	70	45	543	30.5	0.158	53
9	8	125	96	0	0	0.0	0.232	54

## Пропущенные значения

Реальные наборы данных могут иметь пропуски значений параметров. Это может случиться из-за технических проблем, или если датасет собран из нескольких источников с разными наборами параметров; важно то, что в таблице присутствуют пустые ячейки

*Примеры:*

- *Отзывы на кинопоиске*

## Случайность пропусков

Предполагается, что пропуски в матрице располагаются случайно

*Примеры исключений:*

- *отказ респондентов отвечать на вопросы*

Для работы с отсутствующими значениями **нужно использовать интуицию**, чтобы выяснить, почему значение отсутствует.

**Это значение отсутствует, потому что оно не было записано или потому что оно не существует?**

- ✓ Если значение отсутствует, потому что оно не существует (например, наличие детей). Эти значения можно сохранить как NaN.
- ✓ Если значение отсутствует из-за того, что оно не было записано, можно попытаться угадать, что это могло быть, основываясь на других значениях в этом столбце и строке.

Если пропусков много — тренировка на таких данных сильно ухудшит качество модели, а то и окажется вовсе невозможной. **Многие алгоритмы машинного обучения не только требуют массив чисел (а не NaN или «missing»)**, но и ожидают, что этот массив будет состоять из валидных данных, а в случае наличия пропущенных значений в массиве будет выбрасываться исключение.

Как можно выявить  
пропущенные данные?

Функция **info()** выводит  
информацию о количестве  
ненулевых значений по столбцам

```
Ввод [56]: data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 8 columns):
#   Column  Non-Null Count  Dtype  
---  -
0    0      768 non-null    int64  
1    1      763 non-null    float64
2    2      733 non-null    float64
3    3      541 non-null    float64
4    4      394 non-null    float64
5    5      757 non-null    float64
6    6      768 non-null    float64
7    7      768 non-null    int64  
dtypes: float64(6), int64(2)
memory usage: 48.1 KB
```

Подсчет количества нулевых  
значений по столбцам

```
Ввод [48]: data[data.eq(0)].count()

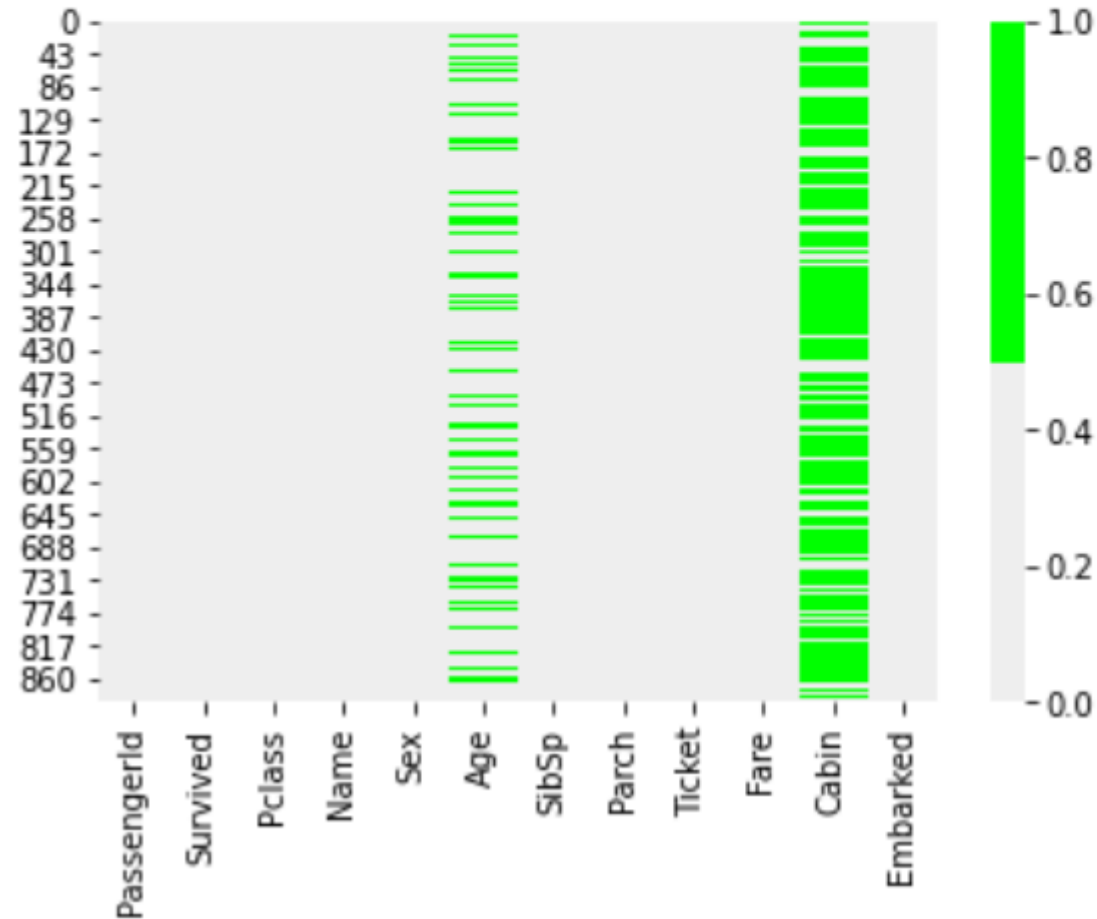
Out[48]: 0      111
         1       5
         2      35
         3     227
         4     374
         5      11
         6       0
         7       0
         dtype: int64
```

# Как можно выявить пропущенные данные?

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib.mlab as mlab
```

```
cols = data.columns[:]
# определяем цвета
# красный - пропущенные данные
colours = ['#eeeeee', '#00ff00']
sns.heatmap(data[cols].isnull(), cmap=sns.color_palette(colours))
```

<AxesSubplot:>





## Как быть?

**1. Применение алгоритмов.** Отдельные алгоритмы умеют принимать во внимание и даже восстанавливать пропущенные значения в данных. НО в таком случае нужно понимать, как именно алгоритм обойдётся с недостающими данными, иначе есть риск получить какие-нибудь артефакты и даже не узнать, какие именно.

**2. Исключение и игнорирование** строк с пропущенными значениями стало решением по умолчанию в некоторых популярных прикладных пакетах. НО **такой метод применим только в том случае, когда малая часть объектов выборки имеет пропущенные значения.** Преимуществом данного подхода является простота и невозможность испортить данные путем замены пропусков. В случае достаточно большого размера выборки метод может показывать хорошие результаты.

**3. Замена пропусков** зачастую и вполне обоснованно кажется более предпочтительным решением. Однако это не всегда так. Неудачный выбор метода заполнения пропусков может не только не улучшить, но и сильно ухудшить результаты.

Методы замены пропусков на основе имеющейся информации часто объединяют в одну группу, называемую *Single-imputation methods*.

## Замена значений

### Замена данных средним/модой

В случае категориального признака все пропуски заменяются на наиболее часто встречающееся значение, в случае количественного признака – на среднее значение по признаку. Данный метод позволяет учитывать имеющиеся данные.

	col1	col2	col3	col4	col5		col1	col2	col3	col4	col5	
0	2	5.0	3.0	6	NaN	mean()	0	2.0	5.0	3.0	6.0	7.0
1	9	NaN	9.0	0	7.0		1	9.0	11.0	9.0	0.0	7.0
2	19	17.0	NaN	9	NaN		2	19.0	17.0	6.0	9.0	7.0

#### Плюсы:

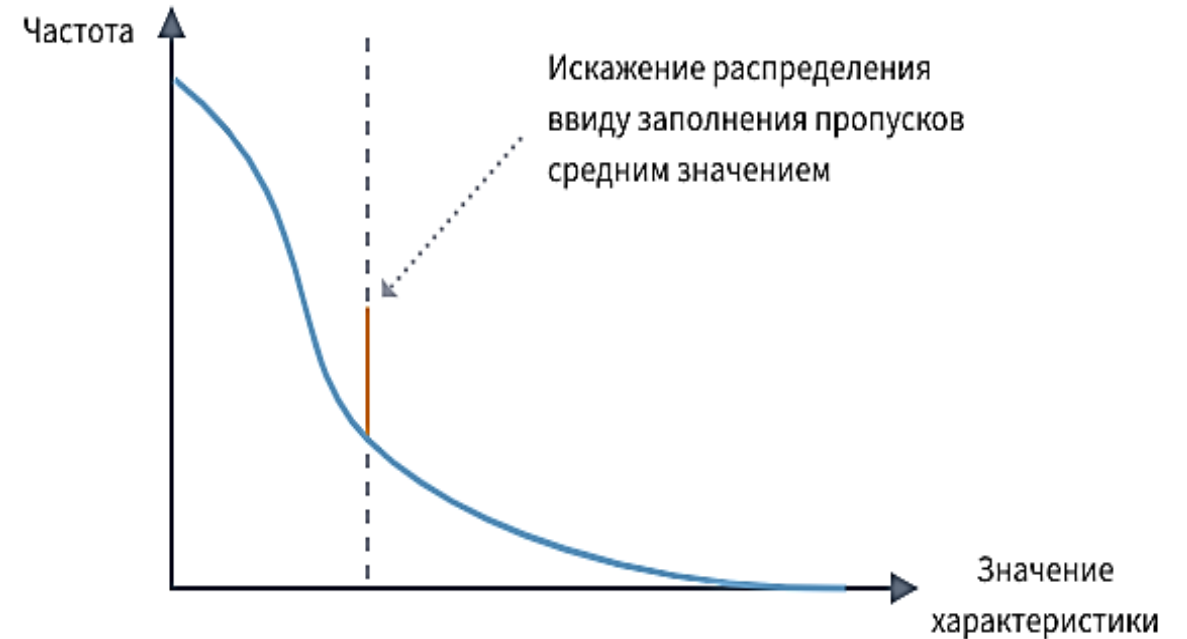
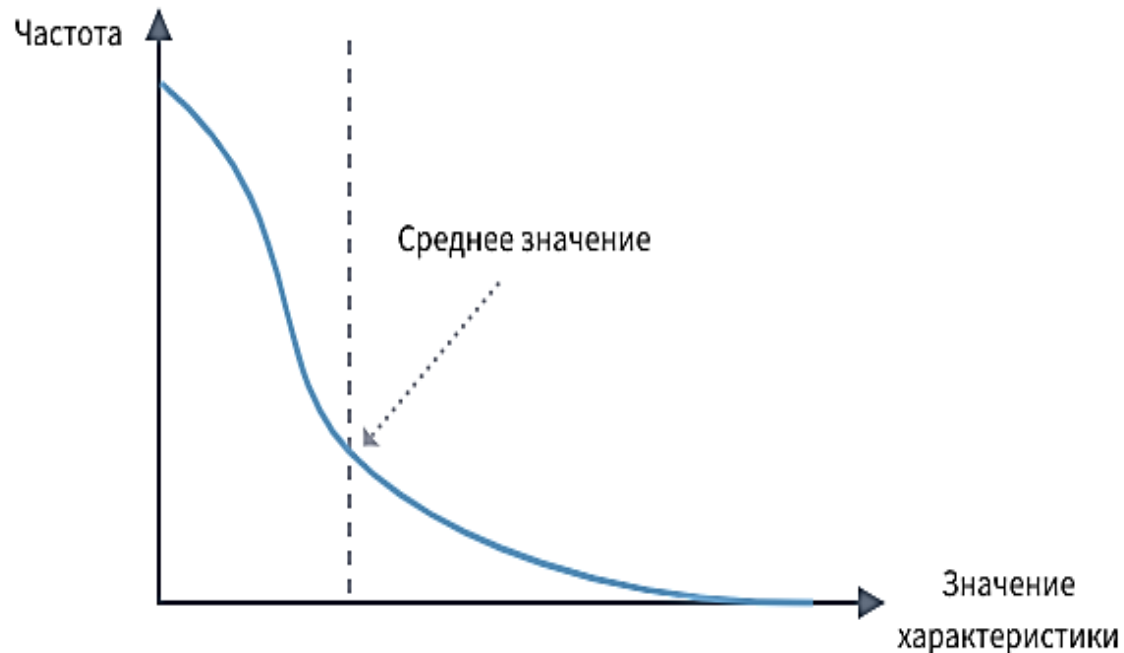
- Просто и быстро.
- Хорошо работает на небольших наборах численных данных.

#### Минусы:

- Значения вычисляются независимо для каждого столбца, так что корреляции между параметрами не учитываются.
- Не работает с качественными переменными.
- Метод не особенно точный.

При заполнении пропусков средним значением, модой, нулем или медианой свойственны одни и те же **недостатки**.

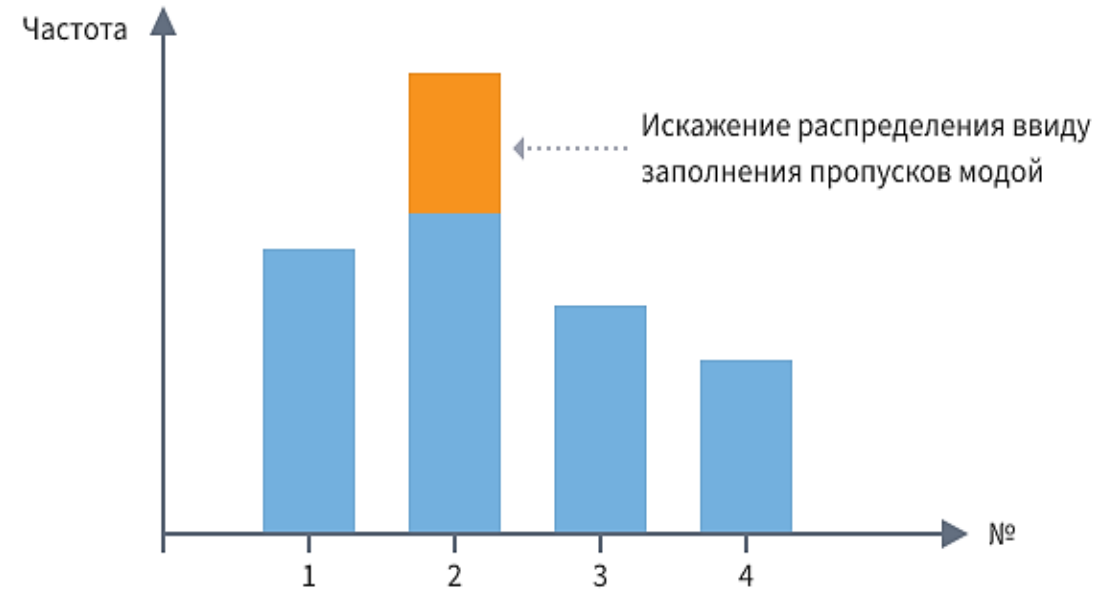
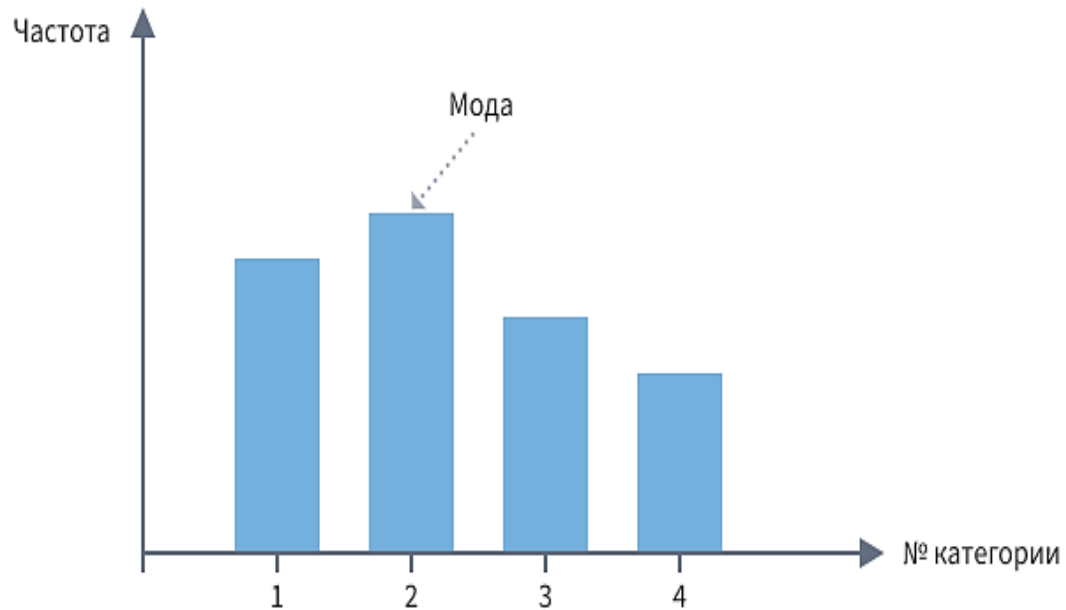
На рисунках: распределение значений непрерывной характеристики до заполнения пропусков **средним значением** и после него.



Данный метод приводит к существенному искажению распределения характеристики. Это в итоге проявляется в искажении всех показателей, характеризующих свойства распределения (кроме среднего значения), заниженной корреляции и завышенной оценке стандартных отклонений.

В случае категориальной дискретной характеристики наиболее часто используется **заполнение модой**.

На рисунке 2 показано распределение категориальной характеристики до и после заполнения пропусков.



При заполнении пропусков категориальной характеристики модой проявляются те же недостатки, что и при заполнении пропусков непрерывной характеристики средним арифметическим (нулем, медианой и тому подобным).

# Реализация замены пропущенных значений

## pandas.DataFrame.fillna

`DataFrame.fillna(value=None, *, method=None, axis=None, inplace=False, limit=None, downcast=_NoDefault.no_default)`

<https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.fillna.html>

## sklearn.impute.SimpleImputer

`sklearn.impute.SimpleImputer(*, missing_values=nan, strategy='mean', fill_value=None, copy=True, add_indicator=False, keep_empty_features=False)`

<https://scikit-learn.org/stable/modules/generated/sklearn.impute.SimpleImputer.html#sklearn.impute.SimpleImputer>

```
import numpy as np
from sklearn.impute import SimpleImputer
imp = SimpleImputer(missing_values=np.nan, strategy='mean')
```

```
imp = SimpleImputer(missing_values=-1, strategy='mean')
```

## Замена NaN средним значением

`data.fillna(data.mean())`

	0	1	2	3	4	5	6	7
0	6	148.0	72.0	35.0	NaN	33.6	0.627	50
1	1	85.0	66.0	29.0	NaN	26.6	0.351	31
2	8	183.0	64.0	NaN	NaN	23.3	0.672	32
3	1	89.0	66.0	23.0	94.0	28.1	0.167	21
4	0	137.0	40.0	35.0	168.0	43.1	2.288	33
5	5	116.0	74.0	NaN	NaN	25.6	0.201	30
6	3	78.0	50.0	32.0	88.0	31.0	0.248	26
7	10	115.0	NaN	NaN	NaN	35.3	0.134	29
8	2	197.0	70.0	45.0	543.0	30.5	0.158	53
9	8	125.0	96.0	NaN	NaN	NaN	0.232	54

	0	1	2	3	4	5	6	7
0	6	148.0	72.000000	35.00000	155.548223	33.600000	0.627	50
1	1	85.0	66.000000	29.00000	155.548223	26.600000	0.351	31
2	8	183.0	64.000000	29.15342	155.548223	23.300000	0.672	32
3	1	89.0	66.000000	23.00000	94.000000	28.100000	0.167	21
4	0	137.0	40.000000	35.00000	168.000000	43.100000	2.288	33
5	5	116.0	74.000000	29.15342	155.548223	25.600000	0.201	30
6	3	78.0	50.000000	32.00000	88.000000	31.000000	0.248	26
7	10	115.0	72.405184	29.15342	155.548223	35.300000	0.134	29
8	2	197.0	70.000000	45.00000	543.000000	30.500000	0.158	53
9	8	125.0	96.000000	29.15342	155.548223	32.457464	0.232	54

## Замена NaN следующим значением

`data.fillna(value=None, method="bfill")`

	0	1	2	3	4	5	6	7
0	6	148.0	72.0	35.0	NaN	33.6	0.627	50
1	1	85.0	66.0	29.0	NaN	26.6	0.351	31
2	8	183.0	64.0	NaN	NaN	23.3	0.672	32
3	1	89.0	66.0	23.0	94.0	28.1	0.167	21
4	0	137.0	40.0	35.0	168.0	43.1	2.288	33
5	5	116.0	74.0	NaN	NaN	25.6	0.201	30
6	3	78.0	50.0	32.0	88.0	31.0	0.248	26
7	10	115.0	NaN	NaN	NaN	35.3	0.134	29
8	2	197.0	70.0	45.0	543.0	30.5	0.158	53
9	8	125.0	96.0	NaN	NaN	NaN	0.232	54

	0	1	2	3	4	5	6	7
0	6	148.0	72.0	35.0	94.0	33.6	0.627	50
1	1	85.0	66.0	29.0	94.0	26.6	0.351	31
2	8	183.0	64.0	23.0	94.0	23.3	0.672	32
3	1	89.0	66.0	23.0	94.0	28.1	0.167	21
4	0	137.0	40.0	35.0	168.0	43.1	2.288	33
5	5	116.0	74.0	32.0	88.0	25.6	0.201	30
6	3	78.0	50.0	32.0	88.0	31.0	0.248	26
7	10	115.0	70.0	45.0	543.0	35.3	0.134	29
8	2	197.0	70.0	45.0	543.0	30.5	0.158	53
9	8	125.0	96.0	23.0	846.0	37.6	0.232	54

## Замена NaN следующим значением

`data.fillna(value=None, method="bfill", limit=1)`

	0	1	2	3	4	5	6	7
0	6	148.0	72.0	35.0	NaN	33.6	0.627	50
1	1	85.0	66.0	29.0	NaN	26.6	0.351	31
2	8	183.0	64.0	NaN	NaN	23.3	0.672	32
3	1	89.0	66.0	23.0	94.0	28.1	0.167	21
4	0	137.0	40.0	35.0	168.0	43.1	2.288	33
5	5	116.0	74.0	NaN	NaN	25.6	0.201	30
6	3	78.0	50.0	32.0	88.0	31.0	0.248	26
7	10	115.0	NaN	NaN	NaN	35.3	0.134	29
8	2	197.0	70.0	45.0	543.0	30.5	0.158	53
9	8	125.0	96.0	NaN	NaN	NaN	0.232	54

	0	1	2	3	4	5	6	7
0	6	148.0	72.0	35.0	NaN	33.6	0.627	50
1	1	85.0	66.0	29.0	NaN	26.6	0.351	31
2	8	183.0	64.0	23.0	94.0	23.3	0.672	32
3	1	89.0	66.0	23.0	94.0	28.1	0.167	21
4	0	137.0	40.0	35.0	168.0	43.1	2.288	33
5	5	116.0	74.0	32.0	88.0	25.6	0.201	30
6	3	78.0	50.0	32.0	88.0	31.0	0.248	26
7	10	115.0	70.0	45.0	543.0	35.3	0.134	29
8	2	197.0	70.0	45.0	543.0	30.5	0.158	53
9	8	125.0	96.0	NaN	NaN	37.6	0.232	54



# Замена пропусков новым «плейсхолдером»

- ✓ Пропуски в категориальных переменных можно закодировать новой категорией (например, MISSING)
- ✓ Пропуски в числовых признаках можно заполнить числом -999  
*(к примеру, при использовании метода решающих деревьев, такие данные попадут в отдельный лист).*

```
# категориальные признаки
df['Title1'] = df[' Title1'].fillna('MISSING')

# численные признаки
df['l Title2'] = df[' Title2'].fillna(-999)
```

**Таким образом, можно сохранить данные о пропущенных значениях, что тоже может быть ценной информацией.**

## Повторение результата последнего наблюдения

Замена  
значений

LOCF (Last observation carried forward) — повторение результата последнего наблюдения. Данный метод применяется, как правило, при заполнении пропусков **во временных рядах**, когда последующие значения априори сильно взаимосвязаны с предыдущими.

Когда применение LOCF обосновано.

Пример: если мы измеряем температуру воздуха в некоторой географической точке на открытом пространстве, причем измерения проводятся каждую минуту, то при нормальных условиях — если исключить природные катаклизмы — измеряемая величина априори не может резко (на 10–20 °C) измениться за столь короткий интервал времени между последующими измерениями. Следовательно, заполнение пропусков предшествующим известным значением в такой ситуации обоснованно.

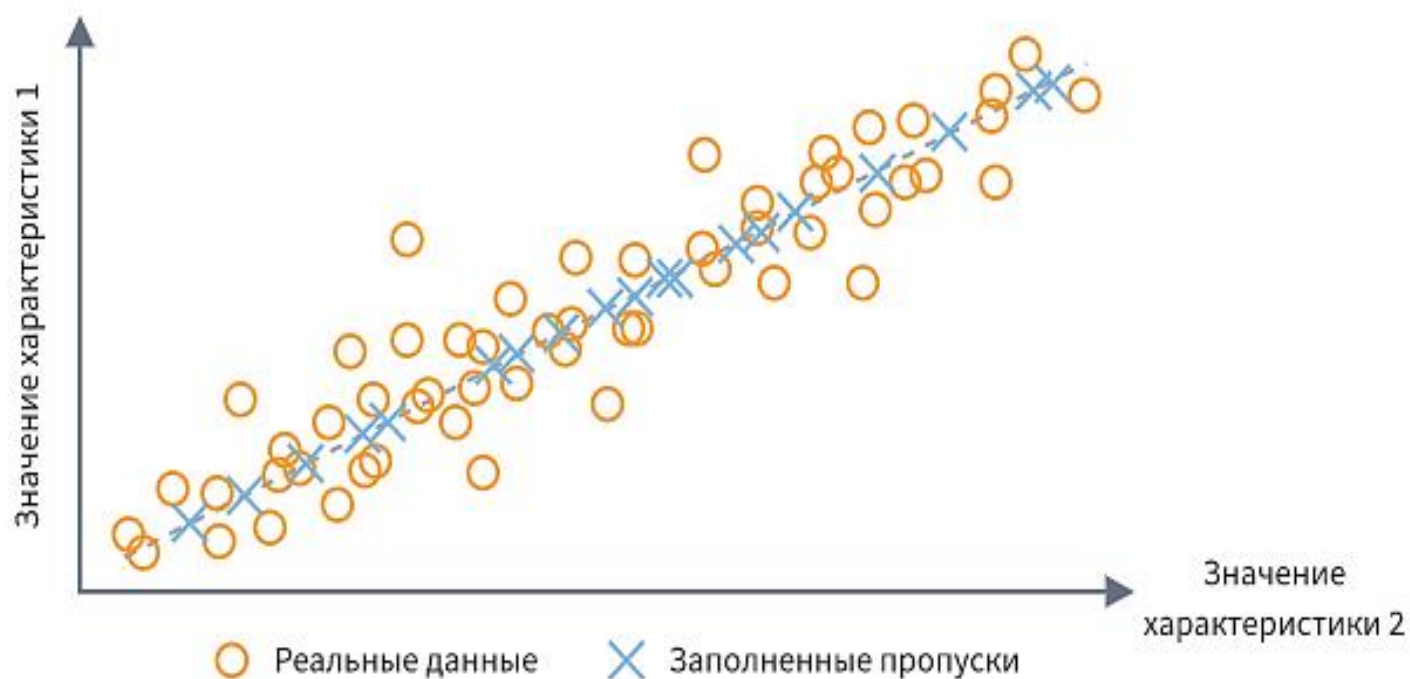
Хотя в описанной выше ситуации метод логичен и обоснован, он тоже может привести к существенным искажениям статистических свойств. Так, возможна ситуация, когда применение LOCF приведет к дублированию выброса (заполнению пропусков аномальным значением). Кроме того, если в данных много последовательно пропущенных значений, то гипотеза о небольших изменениях уже не выполняется и, как следствие, использование LOCF приводит к неправильным результатам.

## Восстановление пропусков на основе регрессионных моделей

Данный метод заключается в том, что пропущенные значения заполняются с помощью модели **линейной регрессии**, построенной на известных значениях набора данных.

На рисунке показан пример результатов заполнения пропущенных значений характеристики 1 на основе известных значений характеристики 2.

Метод линейной регрессии позволяет получить правдоподобно заполненные данные.



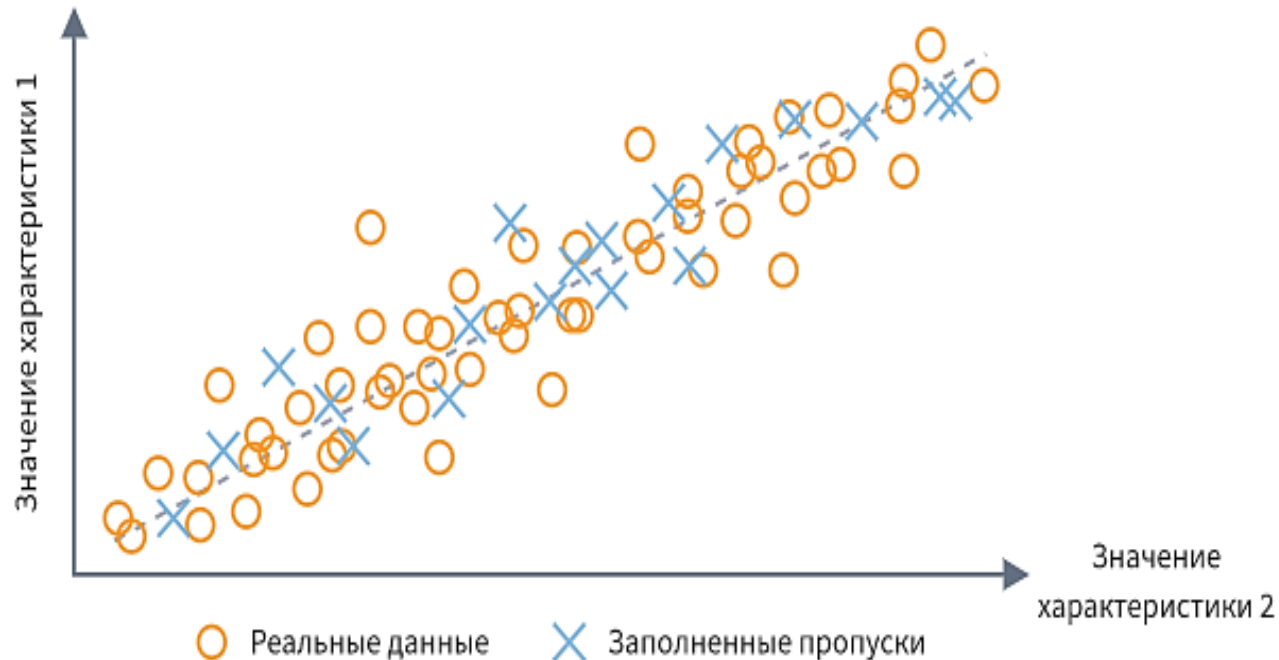
Однако реальным данным свойственен некоторый разброс значений, который при заполнении пропусков на основе линейной регрессии отсутствует. Как следствие, вариация значений характеристики становится меньше, а корреляция между характеристикой 2 и характеристикой 1 искусственно усиливается.

В результате данный метод заполнения пропусков становится тем хуже, чем выше вариация значений характеристики, пропуски в которой мы заполняем, и чем выше процент пропущенных строк.

Есть метод, решающий эту проблему: **метод стохастической линейной регрессии.**

На рисунке — **заполнение пропусков на основе стохастической линейной регрессии**

Модель стохастической линейной регрессии отражает не только линейную зависимость между характеристиками, но и отклонения от этой линейной зависимости. Этот метод обладает положительными свойствами заполнения пропусков на основе линейной регрессии и, кроме того, не так сильно искажает значения коэффициентов корреляции.



Заполнение пропусков с помощью стохастической линейной регрессии в общем случае приводит к наименьшим искажениям статистических свойств выборки.

А в случае, когда между характеристиками прослеживаются явно выраженные линейные зависимости, метод стохастической линейной регрессии нередко превосходит даже более сложные методы.

# Замена данных с помощью метода k-NN

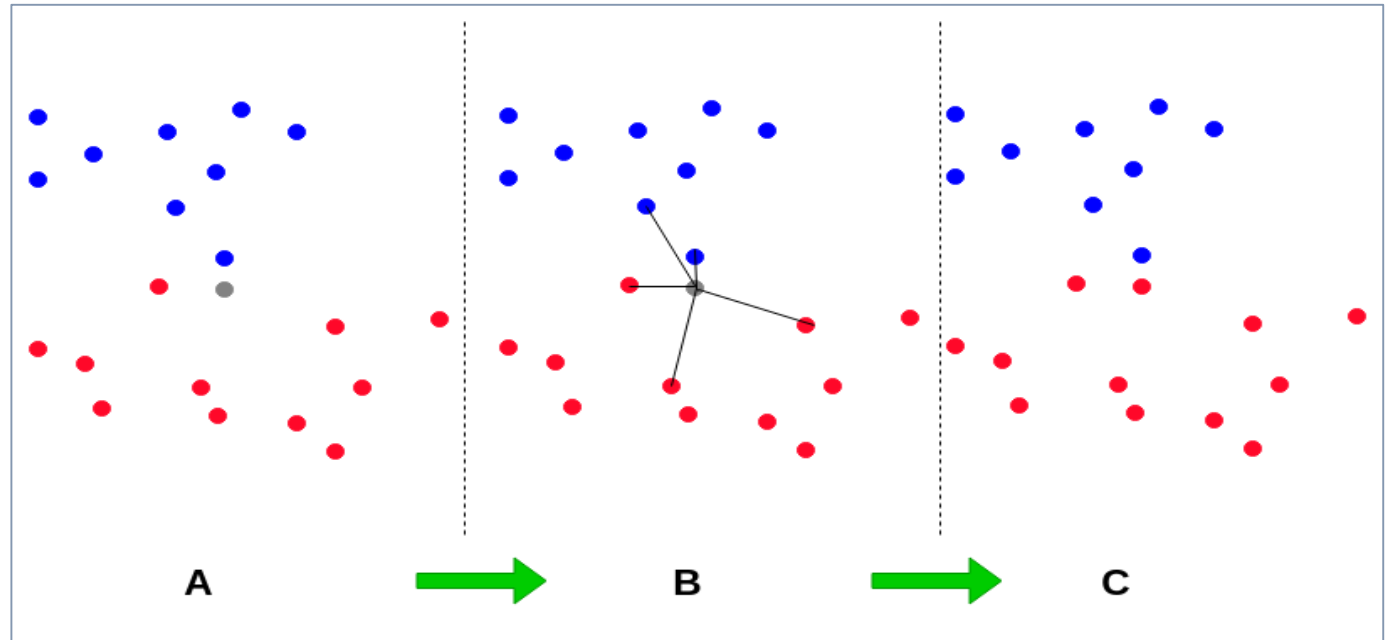
k-Nearest Neighbour (k ближайших соседей) — простой алгоритм классификации, он использует сходство точек, чтобы предсказать недостающие значения на основании  $k$  ближайших точек, у которых это значение есть. Иными словами, выбирается  $k$  точек, которые больше всего похожи на рассматриваемую, и уже на их основании выбирается значение для пустой ячейки.

## Плюсы:

- На некоторых датасетах может быть точнее среднего/медианы или константы.
- Учитывает корреляцию между параметрами.

## Минусы:

- Вычислительно дороже, так как требует держать весь набор данных в памяти.
- Важно понимать, какая метрика дистанции используется для поиска соседей.
- Может потребоваться предварительная нормализация данных.
- Чувствителен к выбросам в данных.



## Замена данных с помощью глубокого обучения

Глубокое обучение (нейросети) хорошо работает с дискретными и другими нечисленными значениями.

### **Плюсы:**

- Точнее других методов.
- Может работать с качественными параметрами.

### **Минусы**

- Восстанавливает только один столбец.
- На больших наборах данных может быть вычислительно дорого.
- Нужно заранее решить, какие столбцы будут использоваться для предсказания недостающего значения.

## Пример

Определим функцию для измерения качества каждого подхода.  
Рассчитаем среднюю абсолютную ошибку (mean absolute error, MAE)

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error

# Function for comparing different approaches
def score_dataset(X_train, X_valid, y_train, y_valid):
    model = RandomForestRegressor(n_estimators=10, random_state=0)
    model.fit(X_train, y_train)
    preds = model.predict(X_valid)
    return mean_absolute_error(y_valid, preds)
```



Удалим столбцы с пропущенными значениями

```
# Get names of columns with missing values
cols_with_missing = [col for col in X_train.columns
                      if X_train[col].isnull().any()]

# Drop columns in training and validation data
reduced_X_train = X_train.drop(cols_with_missing, axis=1)
reduced_X_valid = X_valid.drop(cols_with_missing, axis=1)

print("MAE from Approach 1 (Drop columns with missing values):")
print(score_dataset(reduced_X_train, reduced_X_valid, y_train, y_valid))
```

MAE from Approach 1 (Drop columns with missing values):

183550.22137772635



Заменим пропущенные значения средним значением по каждому столбцу

```
from sklearn.impute import SimpleImputer

# Imputation
my_imputer = SimpleImputer()
imputed_X_train = pd.DataFrame(my_imputer.fit_transform(X_train))
imputed_X_valid = pd.DataFrame(my_imputer.transform(X_valid))

# Imputation removed column names; put them back
imputed_X_train.columns = X_train.columns
imputed_X_valid.columns = X_valid.columns

print("MAE from Approach 2 (Imputation):")
print(score_dataset(imputed_X_train, imputed_X_valid, y_train, y_valid))
```

MAE from Approach 2 (Imputation):

178166.46269899711

Разберем методы поиска и исправления:

1. отсутствующих данных;

2. нетипичных данных – выбросов;

3. неинформативных данных – дубликатов;

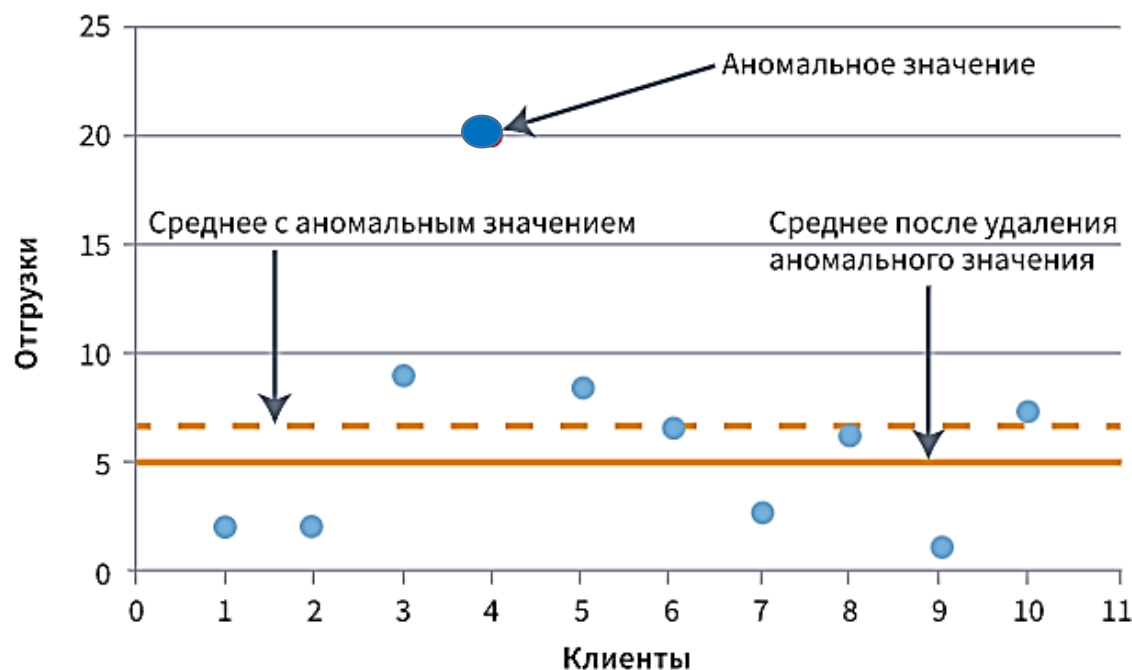
4. несогласованных данных – одних и тех же данных, представленных в разных регистрах или форматах.

## 2. Нетипичные данные (выбросы)(Outlier value)

Выбросы – это данные, которые существенно отличаются от других наблюдений. Они могут соответствовать реальным отклонениям, но могут быть и просто ошибками.

Выбросы необходимо подавить или удалить, поскольку они могут вызвать некорректную работу алгоритмов и привести к искажению результатов анализа данных.

Степень устойчивости алгоритма к наличию в данных выбросов называется **робастностью**.



Что делать?

- Оценить наличие
- Заменить/Исключить

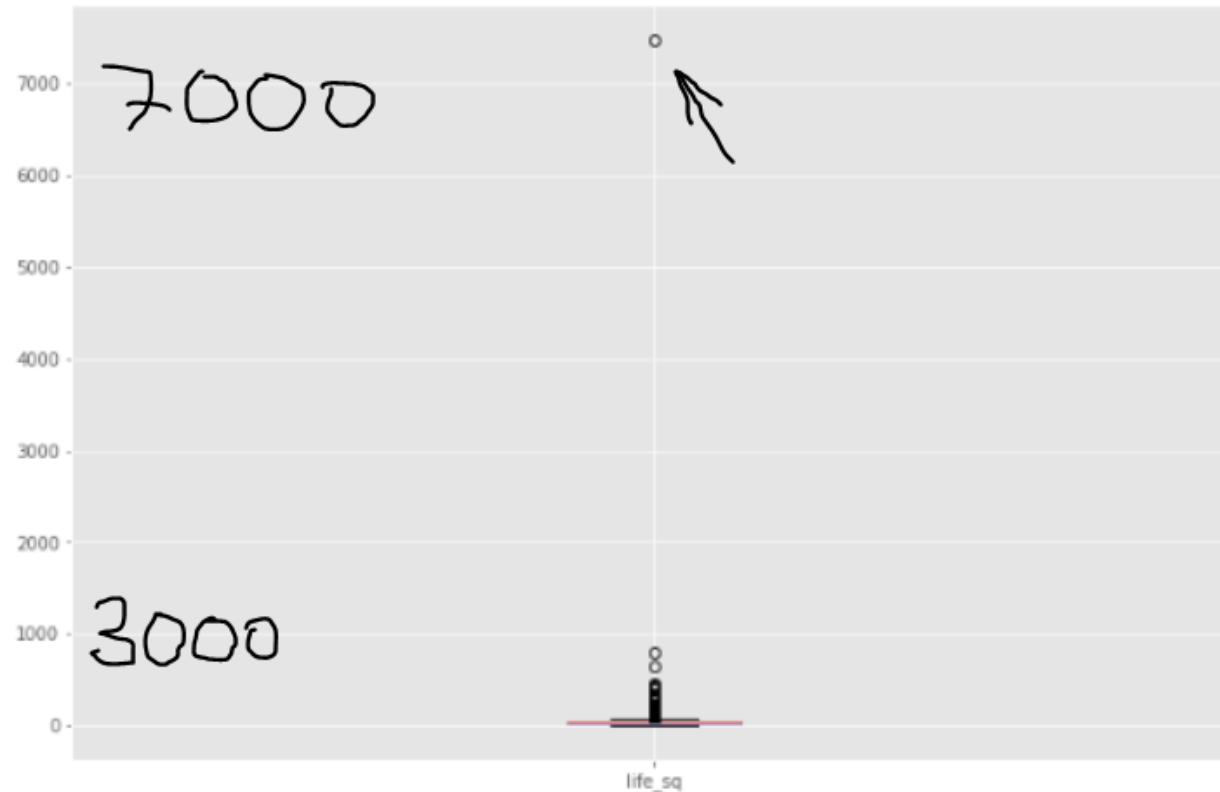
Вопрос: как оценить данные на наличие выбросов

## 2.1. Как обнаружить выбросы?

Для численных и категориальных признаков используются разные методы изучения распределения, позволяющие обнаружить выбросы.

### 2.1.1. Гистограмма/коробчатая диаграмма

Если признак численный, можно построить box-plot (ящик с усами).



## 2.1.2. Описательная статистика

Можно проанализировать описательную статистику.

Например, для признака видно, что максимальное значение равно 7478, в то время как 75% квартиль равен только 43. Значение 7478 – выброс.

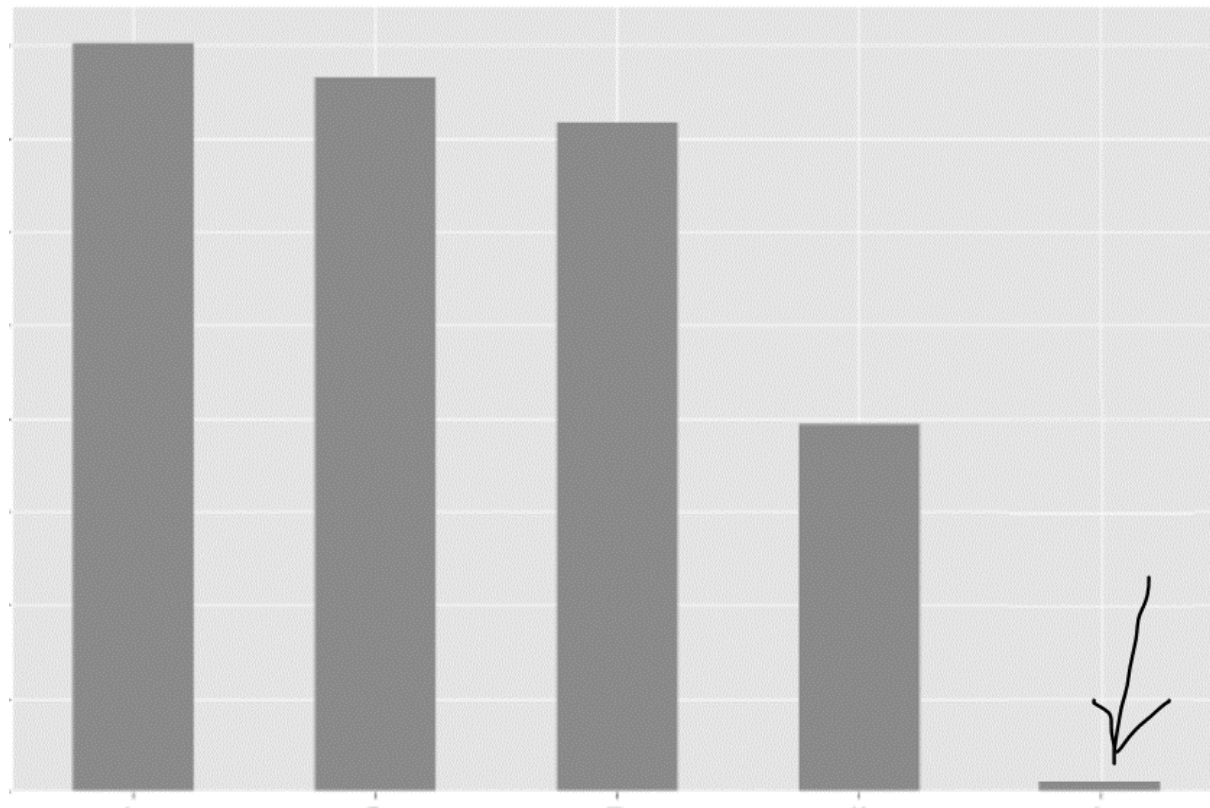
```
df['life_sq'].describe()
```

```
count    24088.000000
mean      34.403271
std       52.285733
min        0.000000
25%       20.000000
50%       30.000000
75%       43.000000
max       7478.000000
Name: life_sq, dtype: float64
```

### 2.1.3. Столбчатая диаграмма

Для категориальных признаков можно построить столбчатую диаграмму – для визуализации данных о категориях и их распределении.

Например, на рисунке распределение признака вполне равномерно и допустимо. Но если существует категория только с одним значением, то это будет выброс.



Разберем методы поиска и исправления:

1. отсутствующих данных;
2. нетипичных данных – выбросов;
3. неинформативных данных – дубликатов;
4. несогласованных данных – одних и тех же данных, представленных в разных регистрах или форматах.

### 3. Неинформативные данные

Вся информация, поступающая в модель, должна служить целям проекта. Если она не добавляет никакой ценности, от нее следует избавиться.

Три основных типа «ненужных» данных:

1. неинформативные признаки с большим количеством одинаковых значений,
2. нерелевантные признаки,
3. дубликаты записей.



## 3.1 Неинформативные признаки

Если признак имеет слишком много строк с одинаковыми значениями, он не несет полезной информации для проекта.

### Как обнаружить?

Можно составить список признаков, у которых более 95% строк содержат одно и то же значение.

### Что делать?

Если после анализа причин получения повторяющихся значений пришли к выводу, что признак не несет полезной информации, используйте `drop()`.

```
num_rows = len(df.index)
low_information_cols = []

for col in df.columns:
    cnts = df[col].value_counts(dropna=False)
    top_pct = (cnts/num_rows).iloc[0]

    if top_pct > 0.95:
        low_information_cols.append(col)
        print('{0}: {1:.5f}%'.format(col, top_pct*100))
        print(cnts)
        print()
```

## 3.2 Нерелевантные признаки (син. *неважный, незначимый*)

Нерелевантные признаки обнаруживаются ручным отбором и оценкой значимости.

Например в датасете по ценам на жилье в России, признак, регистрирующий температуру воздуха в Торонто точно не имеет никакого отношения к прогнозированию цен на российское жилье. Если признак не имеет значения для проекта, его нужно исключить.

*Часто бывает так, что признаки довольно сильно зависят друг от друга и их одновременное наличие избыточно. Например, у вас есть две переменные - «время, проведенное на беговой дорожке в минутах» и «сожженные калории». Эти переменные сильно взаимосвязаны: чем больше времени вы проводите на беговой дорожке, тем больше калорий вы сжигаете. Следовательно, нет смысла хранить оба, поскольку только один из них уже содержит необходимую информацию.*

### 3.3 Дубликаты записей

Если значения признаков (всех или большинства) в двух разных записях совпадают, эти записи называются дубликатами.

#### Как обнаружить повторяющиеся записи?

Способ обнаружения дубликатов зависит от того, что именно мы считаем дубликатами.

- ✓ Например, в наборе данных есть уникальный идентификатор *id*. Если две записи имеют одинаковый *id*, то считаем, что это одна и та же запись. Удаляем все неуникальные записи.
- ✓ Другой распространенный способ вычисления дубликатов: по набору ключевых признаков. *Например, неуникальными можно считать записи с одной и той же площадью жилья, ценой и годом постройки.*

#### Что делать с дубликатами?

Очевидно, что повторяющиеся записи нам не нужны, значит, их нужно исключить из набора.

1. отсутствующих данных;
2. нетипичных данных – выбросов;
3. неинформативных данных – дубликатов;
4. несогласованных данных – одних и тех же данных, представленных в разных регистрах или форматах.

Большая проблема очистки данных – разные форматы записей.

Рассмотрим четыре самых распространенных несогласованности:

1. Разные регистры символов.
2. Разные форматы данных (например, даты, адреса).
3. Опечатки в значениях категориальных признаков.

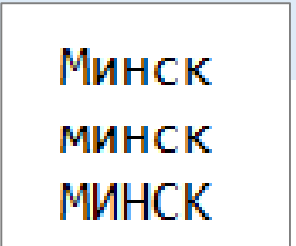
## 4.1 Разные регистры символов

Непоследовательное использование разных регистров в категориальных значениях является очень распространенной ошибкой, которая может существенно повлиять на анализ данных.

### Как обнаружить?

Следующий код выводит все уникальные значения признака:

```
df['title'].value_counts(dropna=False)
```



Минск  
минск  
МИНСК

### Что делать?

Эта проблема легко решается принудительным изменением регистра:

```
# пусть все будет в нижнем регистре  
df['title'] = df['title'].str.lower()  
df['title'].value_counts(dropna=False)
```

## 4.2 Разные форматы данных

Ряд данных в наборе находится не в том формате, с которым нам было бы удобно работать. Например, даты, записанные в виде строки, следует преобразовать в формат `DateTime`.

### Как обнаружить?

В примере признак *timestamp* представляет собой строку, хотя является датой:

### Что же делать?

Значения признака *timestamp* следует преобразовать в удобный формат:

По ссылке можно найти код для данного примера

<https://techrocks.ru/2020/04/02/data-cleaning-with-python/>

	id	timestamp	full_sq	life_sq	floor	n
0	1	2011-08-20	43	27.0	4.0	
1	2	2011-08-23	34	19.0	3.0	
2	3	2011-08-27	43	29.0	2.0	
3	4	2011-09-01	89	50.0	9.0	
4	5	2011-09-05	77	77.0	4.0	

## 4.2 Разные форматы данных

### Адреса

Мало кто следует стандартному формату, вводя свой адрес в базу данных.

### Что делать?

Минимальное форматирование включает следующие операции:

- приведение всех символов к нижнему регистру;
- удаление пробелов в начале и конце строки;
- удаление точек;
- стандартизация формулировок: замена street на st, apartment на apt и т. д.

## 4.3 Опечатки

Опечатки в значениях категориальных признаков приводят к таким же проблемам, как и разные регистры символов. Например, признак `city`, а его значениями будут *torontoo* и *tronto*. В обоих случаях это опечатки, а правильное значение – *toronto*.

### Как обнаружить?

Для обнаружения опечаток требуется особый подход.

Простой способ идентификации подобных элементов – *нечеткая логика* или ***редактирование расстояния***. Суть этого метода заключается в измерении количества букв (расстояния), которые нам нужно изменить, чтобы из одного слова получить другое.

### Что делать?

Можно установить критерии для преобразования этих опечаток в правильные значения.

Например, если расстояние некоторого значения от слова *toronto* не превышает 2 буквы, мы преобразуем это значение в правильное – *toronto*.



# Несбалансированность классов

В машинном обучении нередко возникают ситуации, когда в обучающем наборе данных **доля примеров некоторого класса оказывается слишком низкой, а другого — слишком большой**. Эта ситуация в теории машинного обучения известна как несбалансированность классов (class imbalance), а классификация в условиях несбалансированности классов называется несбалансированной классификацией (unbalanced classification).

Несбалансированность классов как правило создаёт проблемы при решении задач классификации, поскольку построенные на таких данных модели имеют «перекос» в сторону преобладающего класса, т.е. с большей вероятностью присваивают его метку класса новым наблюдениям при практическом использовании модели.

- ✓ Сокращение числа примеров избыточного класса
- ✓ Увеличение числа примеров недостаточного класса (генерация значений).

Подробнее <https://loginom.ru/blog/imbalance-class>

## Прочие этапы предобработки данных

**Трансформация данных** заключается в оптимизации их представлений и форматов с точки зрения решаемых задач и целей анализа. Трансформация не ставит целью изменить информационное содержание данных. Ее задача — представить эту информацию в таком виде, чтобы она могла быть использована наиболее эффективно.

Вообще, трансформация данных — очень широкое понятие, не имеющее четко очерченных границ. Иногда этот термин иногда распространяют на любые манипуляции с данными, независимо от их целей и методов. Однако в контексте анализа данных трансформация данных имеет вполне конкретные цели и задачи, а также использует достаточно стабильный набор методов. К основным из них относятся **нормализация, преобразование типов и форматов, сортировка, группировка, слияние и др.**

<https://wiki.loginom.ru/articles/data-transformation.html>

**Квантование данных (Биннинг)** — это преобразование непрерывной переменной в категориальную переменную. Например, если мы хотим применить условия к непрерывным столбцам, скажем, к столбцу «вес машины», мы можем создать новый категориальный столбец с помощью:

вес > 1500 и вес < 2500 как «Легкий»

вес > 2500 и вес < 3500 как «средний»

вес > 3500 и вес < 4500 как "Heavy"

вес > 4500 как «очень тяжелый»

<https://www.analyticsvidhya.com/blog/2020/09/pandas-speed-up-preprocessing/>

Фильтрация данных (спектральный анализ)

<https://basegroup.ru/community/articles/data-filtration>

# Преппроцессинг

Преппроцессинг описан в разделе [Preprocessing data библиотеки scikit-learn](#). В scikit-learn это трансформация и нормализация данных. Делать это необходимо, так как многие алгоритмы чувствительны к выбросам, а так же распределению данных в выборке.

**StandardScaler** центрирует данные, удаляет среднее значение для каждого объекта, а затем масштабирует, деля на среднее отклонение.  $x_{scaled} = \frac{x-u}{s}$ , где  $u$  среднее, а  $s$  отклонение.

**MinMaxScaler** трансформирует признаки в выбранном диапазоне.  $x_{scaled} = \frac{x - \min(x)}{\max(x) - \min(x)}$

**MaxAbsScaler** трансформирует в диапазон  $[-1,1]$ . Используется для центрированных вокруг нуля или разреженных данных.  $x_{scaled} = \frac{x}{\max(abs(x))}$

**RobustScaler**. Для данных, в которых много выбросов.

Для нормализации данных можно использовать **Normalizer**. Часто это необходимо, когда алгоритм предсказывает, базирясь на взвешенных значениях, основанных на расстояниях между точками данных. Особенно актуально для классификации текста и кластеризации.



# Умная нормализация данных

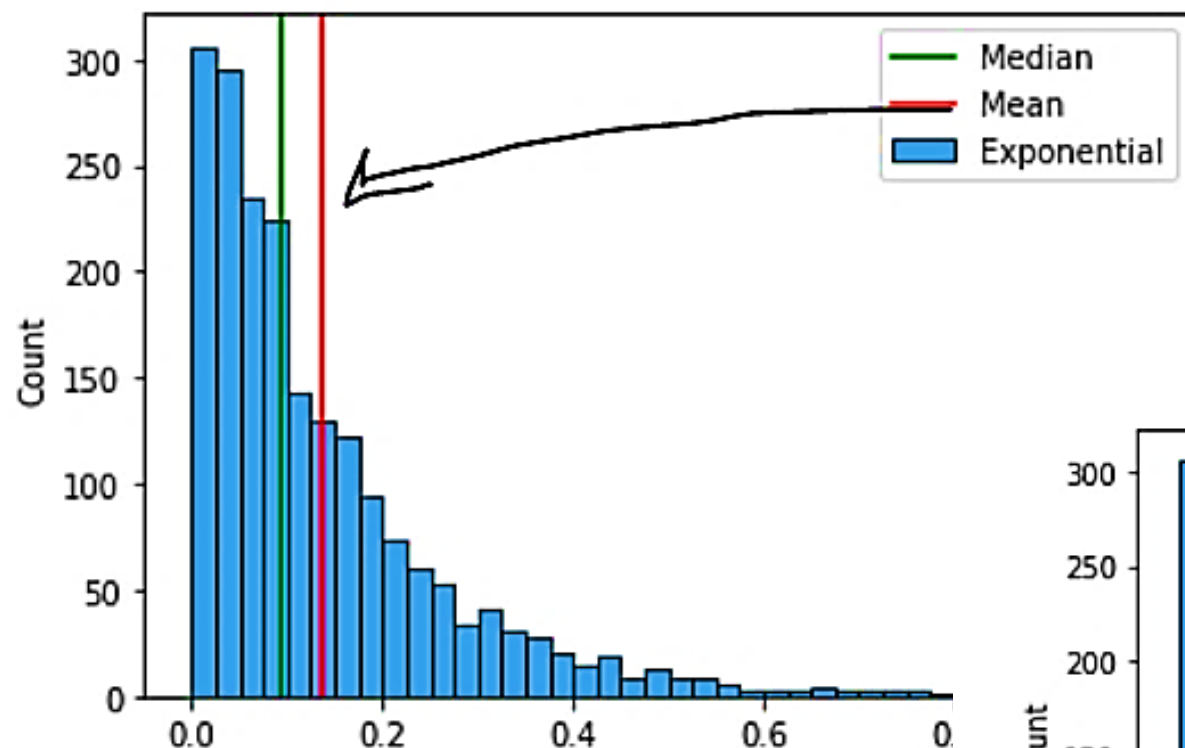
Python\*, Алгоритмы\*, Машинное обучение\*, Искусственный интеллект, Data Engineering\*

Главное условие правильной нормализации — все признаки должны быть равны в возможностях своего влияния.

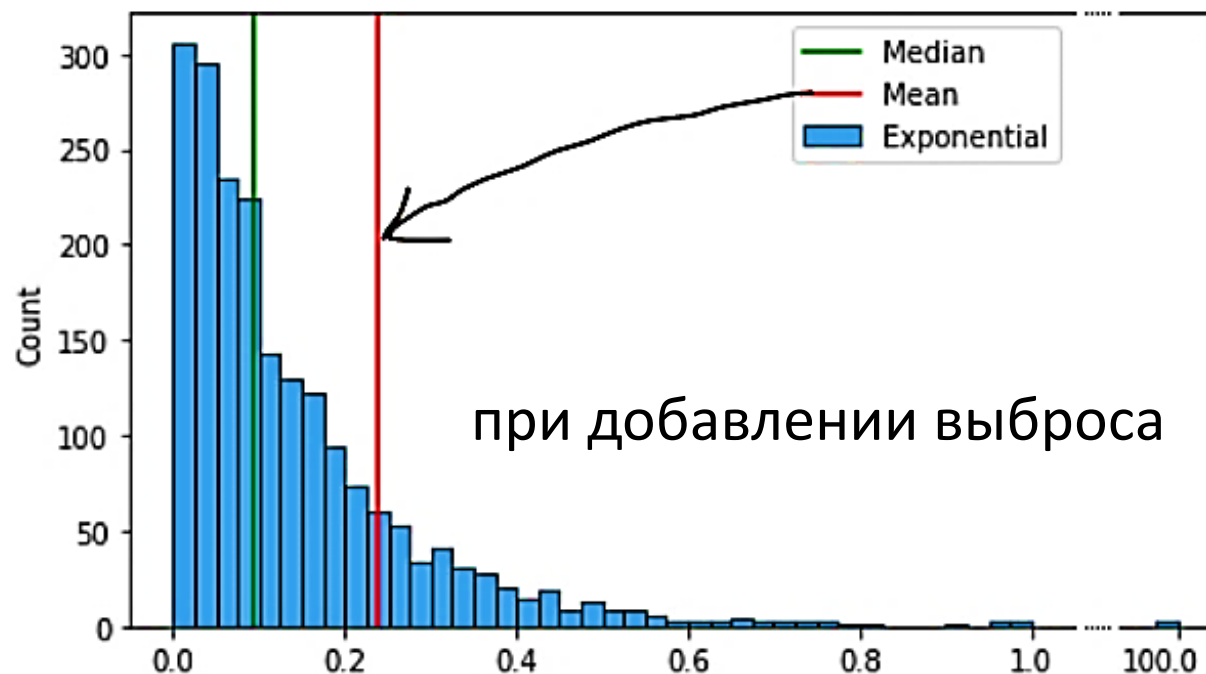
Что лучше взять за центр? Часто используется среднее арифметическое значение. Здесь проявляется проблема № 1 — различные типы распределений не позволяют применять к ним методы, созданные для нормального распределения.

Если Вы спросите любого специалиста по статистике, какое значение лучше всего показывает “типичного представителя” совокупности, то он скажет, что это — **медиана**, а не среднее арифметическое. Последнее хорошо работает только в случае нормального распределения и совпадает с медианой (алгоритм стандартизации вообще оптимален именно для нормального распределения). А у Вас распределения разных признаков могут (и скорее всего будут) кардинально разные.

Вот, например, различия между медианой и средним арифметическим значением для экспоненциального распределения.



В отличие от среднего значения медиана практически не чувствительна к выбросам и асимметрии распределения. Поэтому её оптимально использовать как “нулевое” значение при центрировании



# Выбор значимых признаков

Отбор признаков (**feature selection**) – это оценка важности того или иного признака отсекающие ненужных.

- ❖ Выбор через model-free методы: [scikit-feature](#)

- ❖ Статистики (`sklearn.feature_selection.SelectKBest`)
- ❖ Корреляции Пирсона, Спирмена

- ❖ Выбор через model-based методы:

- ❖ RandomForest (`rf.feature_importances_`, PFI)
- ❖ Lasso, ElasticNet (`lr.coef_`)
- ❖ NN (DFS, HVS, PFI)
- ❖ RFE (SVM, kNN, т.д.)

В 90% случаев основные результаты модели достигаются именно на этом этапе

Алгоритмы выбора признаков реализованы в модуле `sklearn.feature_selection`

## model-free методы

Вот лишь некоторые примеры данных методов:

### 1. Фильтр с низкой дисперсией (VarianceThreshold)

Например, если переменные какого-либо параметра не изменяются (т.е. **параметр имеет нулевую дисперсию**), то **этот параметр никак не влияет на модель**. Поэтому, можно рассчитать дисперсию каждого параметра в выборке, а затем отбросить параметры, имеющие низкую дисперсию по сравнению с другими параметрами в наборе данных. Например, по данным титаника видно, что наименьшую дисперсию дает параметр Parch:

Pclass	0.699015
Age	211.019125
SibSp	1.216043
<u>Parch</u>	<u>0.649728</u>
Fare	2469.436846

Age	Age in years
sibsp	# of siblings / spouses aboard the Titanic
parch	# of parents / children aboard the Titanic
ticket	Ticket number
fare	Passenger fare



## 2. Фильтр высокой корреляции

Высокая корреляция между двумя переменными означает, что они имеют схожие тенденции и, вероятно, несут схожую информацию. Это может резко снизить производительность некоторых моделей (например, моделей линейной и логистической регрессии). Можно вычислить корреляцию между независимыми числовыми переменными. *Если коэффициент корреляции больше определенного порогового значения (как правило 0,5–0,6), мы можем отбросить одну из переменных (удаление переменной очень субъективно и всегда должно производиться с учетом предметной области).*

В качестве общего правила мы должны оставить те переменные, которые показывают приличную или высокую корреляцию с целевой переменной.

Если обнаружены сильные корреляции, то коррелирующие признаки можно записать через функции и сократить их количество



## Выбор через model-based методы

### RandomForest for feature selection

```
num_of_most_important_features = 20
forest = RandomForestClassifier(n_estimators=100)
forest.fit(X_train, y_train)

importances = forest.feature_importances_
indices = np.argsort(importances)[::-1]

five_most_important_features = [X_train.columns[indices[f]] for f in range(num_of_most_important_features)]
five_most_important_features.append('target')

selected_train = train.ix[:,five_most_important_features]
print(selected_train.shape) # after selection we can train new model on the selected features
```

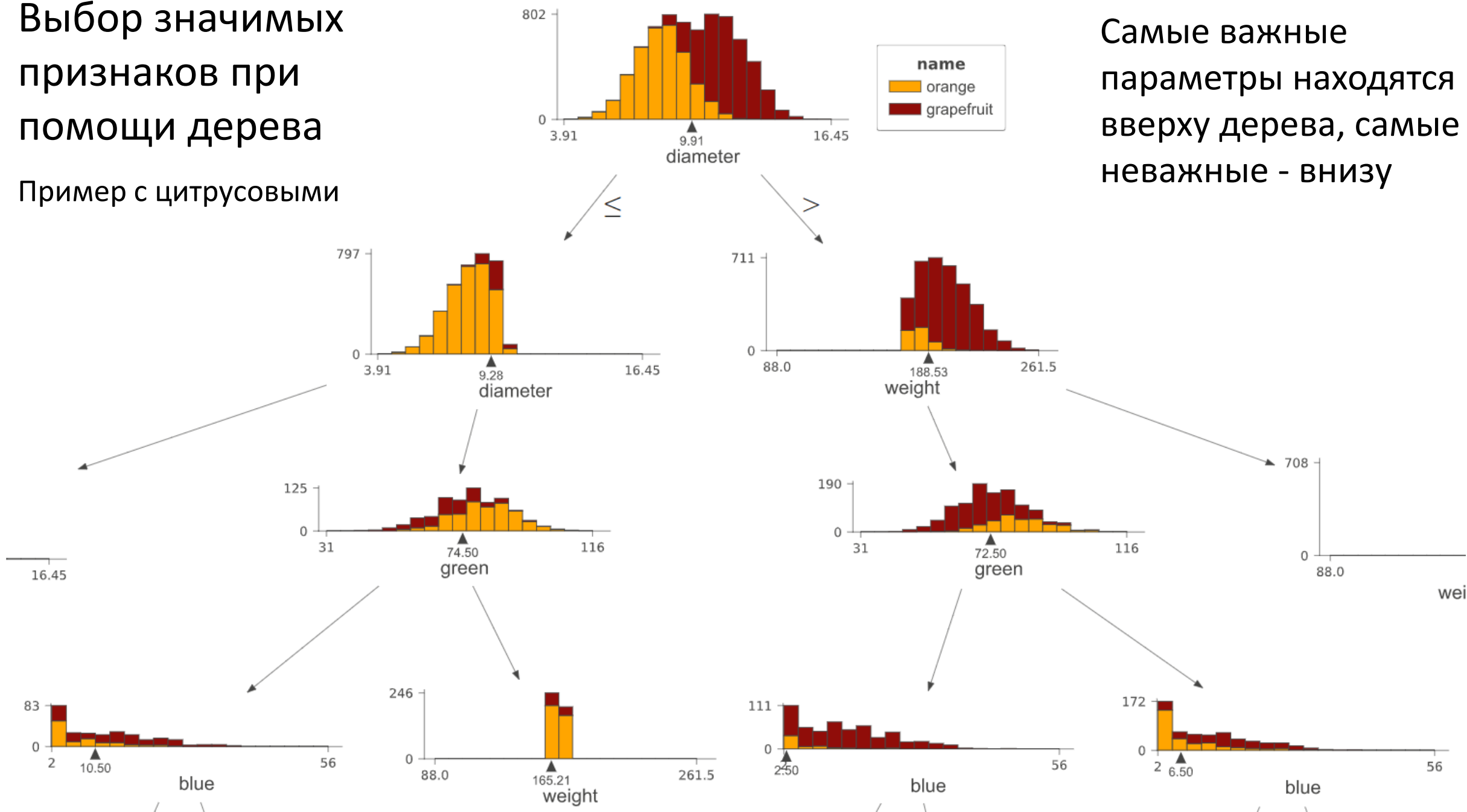
Выделяем важные признаки при помощи случайного леса

Тренируем модель случайного леса, затем сортируем массив признаков и выбираем первые  $f$  признаков

# Выбор значимых признаков при помощи дерева

Пример с цитрусовыми

Самые важные параметры находятся вверху дерева, самые неважные - внизу



### Рекурсивное устранение признаков

При наличии внешнего оценщика, который присваивает веса характеристикам (например, коэффициентам линейной модели), цель исключения рекурсивных признаков (*RFE*) состоит в том, чтобы выбрать признаки путем рекурсивного рассмотрения все меньших и меньших наборов признаков. Оценщик обучается на начальном наборе признаков, и **определяется важность каждого признака**. Затем наименее важные признаки удаляются из текущего набора признаков. Эта процедура рекурсивно повторяется для сокращенного набора, пока в конечном итоге не будет достигнуто желаемое количество признаков.

## PCA for feature extraction

```
pca = PCA(n_components=0.95)
pca_train = pca.fit_transform(train[train.columns[:-1]])
print('{0} components preserve 95% of total variance'.format(pca.n_components_))
```

```
57 components preserve 95% of total variance
```

В примере желаемые 95 % точности при использовании метода PCA дали 57 компонент из 93-х исходного датасета

# Несбалансированные данные

- ❖ Использовать методы балансировки данных: [imbalanced-learn](#)
  - ❖ Undersampling (SVM, kNN, NN)
  - ❖ Oversampling (SVM, kNN, NN)
- ❖ Использовать методы генерирующие данные в процессе своего обучения (NN)
- ❖ Использовать метрики для несбалансированных данных (F1-score, [Matthews correlation coefficient](#))

## Balancing data

```
sm = SMOTE()  
X_train_augmented, y_train_augmented = sm.fit_sample(X_train, y_train)  
print('Original data: {0}'.format(X_train.shape))  
print('Augmented data: {0}'.format(X_train_augmented.shape))
```

```
Original data: (49502, 93)  
Augmented data: (60875, 93)
```

Алгоритм SMOTE() из библиотек **imblearn** делает оверсэмплинг, генерирует объекты меньшего класса

# Вопрос

Что реализует данный фрагмент кода?

```
import pandas as pd
df = pd.DataFrame([[ "a", "x"],
                    [np.nan, "y"],
                    [ "a", np.nan],
                    [ "b", "y"]], dtype="category")

imp = SimpleImputer(strategy="most_frequent")
```