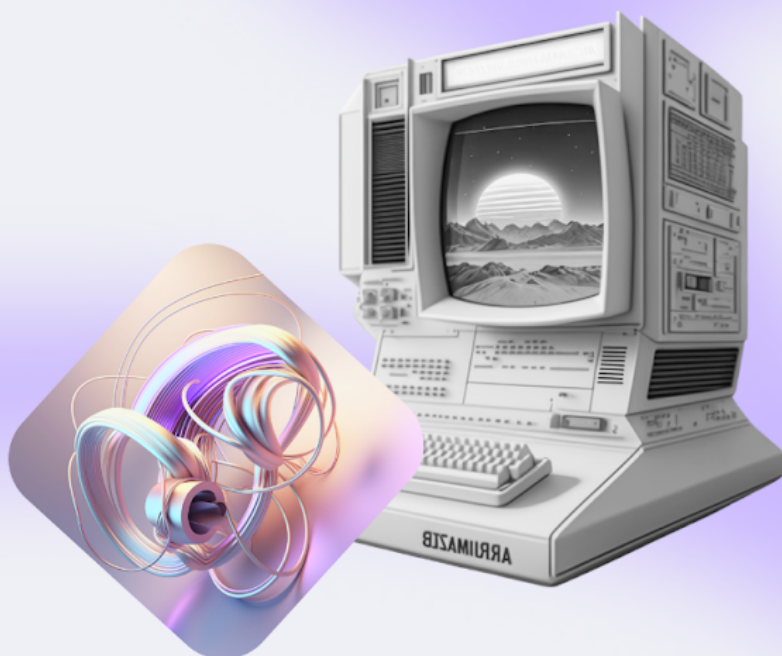


# Работа с данными

Сбор и разметка данных



# Оглавление

Словарь терминов	3
Введение	3
Качество данных. Очистка данных	4
Какие данные нужно очищать	5
Процесс очистки	5
Отсутствующие данные и заполнение недостающих значений	7
Удаление дубликатов	8
Исправление некорректных данных	8
Категориальные данные	9
Выбросы	11
Стандартизация данных	14
Очистка данных для задач компьютерного зрения	14
Качество данных. Трансформация данных	15
Методы структурной трансформации	16
Методы трансформации контента	17
Трансформация на практике	17
Процессинг данных	20
Пакетная обработка	20
Потоковая обработка	22
Обзор систем управления данными	24
Заключение	26
Что можно почитать еще?	26

# Словарь терминов

**Очистка данных** (Data cleansing) — процесс выявления и исправления ошибок, а также несоответствия данных с целью улучшить их качество. Иногда классифицируется как составная часть интеллектуального анализа данных.

## Введение

Мы рассмотрели основы сбора данных. Пришло время поговорить про обработку — этап, на котором данные, собранные из разных источников, преобразуют для использования и анализа.

Обработка данных важна даже больше, чем сбор, потому что ценность данных не в их сырой форме, а в тех выводах, которые можно из них сделать. Другими словами, успех любого проекта, основанного на данных, зависит не только от их качества, но и от возможности эффективно манипулировать и анализировать их.

Сбор и обработка данных тесно связаны. Если собранные данные неточные, неполные или неактуальные, никакие манипуляции не смогут их спасти. С другой стороны, даже самые качественные данные могут оказаться бесполезными без обработки и анализа.

Важно подходить к сбору и обработке как к единому процессу, уделять внимание обоим этапам. Собирая качественные данные и эффективно обрабатывая их, мы можем раскрыть ценные идеи, принять обоснованные решения и добиться положительных результатов в разных областях.

На этой лекции мы разберем ключевые концепции и методы работы с данными, а также попрактикуемся работать с библиотеками Python для анализа. Рассмотрим два основных этапа работы с данными — очистку и преобразование (трансформацию).

Сперва обсудим, почему важно очищать данные — выявлять и исправлять ошибки, заполнять недостающие данные, удалять дубликаты. Рассмотрим распространенные методы очистки данных, а также разберем примеры очистки с помощью библиотеки Pandas.

Затем перейдем к трансформации данных — преобразованию из одного формата в другой, фильтрации и группировке.


Приступим!

# Качество данных. Очистка данных

Очистка данных — это выявление и исправление ошибок, заполнение недостающих данных и удаление дубликатов. Процесс нужен, потому что данные, собранные из разных источников, часто неполные и непоследовательные, в них могут быть ошибки.

Первый шаг очистки — выявление ошибок или несоответствий в данных: например, неправильного форматирования, опечаток, отсутствующих значений или дублирующихся записей. Второй — исправление: либо вручную, либо с помощью алгоритмов.

Очистка — итеративный процесс. Чтобы получить чистые и согласованные данные, иногда нужно повторить эти шаги несколько раз. Процесс требует понимания доменной области, структур данных и языков программирования, а также знания статистических методов и визуализации данных.

 Очистка — критический этап работы с данными. Качество выводов и решений, которые можно сделать на основе данных, напрямую зависит от точности и полноты самих данных.

Чтобы понять, насколько важна очистка, поговорим о последствиях, к которым может привести работа с неочищенными данными.

- **Неточные выводы.** Например, если в наборе данных не хватает значений, любой анализ, проведенный на их основе, может быть искаженным, а выводы — неправильными или вводящими в заблуждение.
- **Нерациональное использование времени и ресурсов.** Если данные не очищены как положено, аналитики могут потратить время и ресурсы на исправление ошибок или несоответствий на более поздних этапах. Это трудоемкий и дорогостоящий процесс, которого можно было бы избежать.
- **Потеря доверия.** Если решения принимают на основе неточных или неполных данных, то доверие к организации или специалисту может быть подорвано. А потерянную репутацию не просто восстановить.
- **Правовые и этические последствия.** Неточные или неочищенные данные могут стать причиной юридических или этических проблем: от штрафов до негативной рекламы.

## Какие данные нужно очищать

Что именно очищать — зависит от источника данных. Но есть и общие моменты.

- **Неполные или отсутствующие данные.** Возможные причины появления — человеческая ошибка, сбой системы, нет ответов участников опроса.
- **Несоответствующие данные.** Это данные, которые противоречат друг другу или нарушают известные правила или взаимосвязи. Например, данные о возрасте человека записаны как более ранние, чем год его рождения.
- **Неверные данные.** Опечатки, ошибки кодирования и так далее.
- **Дублирующиеся данные.** Встречаются больше одного раза. Могут исказить результаты и привести к неточным выводам.
- **Выбросы.** К ним относятся точки данных, которые значительно отличаются от остальных. Могут оказать непропорциональное влияние на анализ.
- **Проблемы с форматированием.** Например, разные форматы дат, единиц измерения или типов данных.
- **Недостоверные данные.** Данные, которые выходят за пределы ожидаемого диапазона значений или нарушают известные правила или ограничения.

## Процесс очистки

Первое, что нужно сделать инженеру, когда он начинает работать с данными — понять их. Нужно потратить время на чтение документации к набору данных (если она есть), выяснить, что представляют собой эти данные. Нужно изучить их, чтобы получить представление об их качестве, полноте и структуре.

Сегодня мы будем работать с легендарным датасетом — [«Титаник»](#).

«Титаник» — это популярный набор данных в Data Science. Он содержит информацию о пассажирах, находившихся на борту «Титаника»: их возрасте, поле, классе, выживаемости.

Набор данных содержит столбцы:

- PassengerId — ID пассажира;
- Survived — выжил ли пассажир (0 = нет, 1 = да);
- Pclass — класс пассажира (1 = первый, 2 = второй, 3 = третий);

- Name — имя;
- Sex — пол;
- Age — возраст;
- SibSp — количество братьев / сестер и супругов пассажира на борту;
- Parch — количество родителей / детей пассажира на борту;
- Ticket — номер билета;
- Fare — цена, которую пассажир заплатил за билет;
- Cabin — номер каюты, в которой находился пассажир;
- Embarked — порт (C = Шербур, Q = Квинстаун, S = Саутгемптон).

Перейдем к работе в Google Colab. После загрузки датасета начнем очистку данных.

```
df = pd.read_csv('train.csv')
```

Сначала выведем первые строки датасета и посмотрим на данные.

```
df.head()
```

Здесь мы можем посмотреть на названия столбцов и получить первое представление о данных.

Затем мы можем посмотреть сводную информацию о наборе данных, включая количество ненулевых значений и тип данных каждого столбца. Библиотека Pandas предоставляет метод `.info()`, который отображает эту информацию. Это быстрый способ получить общее представление о данных. Его можно использовать, чтобы выявить столбцы с отсутствующими значениями или столбцы, которые нужно преобразовать в другой тип данных.

```
df.info()
```

В выводе мы видим, что набор данных Titanic содержит 891 запись и 12 столбцов. Для каждого столбца мы можем увидеть название, количество ненулевых значений и тип данных.

Например, в столбце «Возраст» 714 ненулевых значений, следовательно, 177 значений отсутствуют. В столбце «Кабина» только 204 ненулевых значений, значит большинство данных отсутствуют.

Поговорим про отсутствующие данные подробнее.

## Отсутствующие данные и заполнение недостающих значений

Другой способ проверить, в каких столбцах отсутствуют значения, — метод `isnull()`:

```
df.isnull().sum()
```

В результате получим количество отсутствующих значений в каждом столбце набора данных. Например, можем увидеть, что в столбце «Возраст» 177 пропущенных значений.

Чтобы заполнить недостающие значения, можем использовать метод `fillna()`. Один из распространенных подходов — заполнение недостающих значений средним или медианным значением столбца. Пример для столбца «Возраст»:

```
median_age = df['Age'].median()  
df['Age'] = df['Age'].fillna(median_age)
```

Код позволит заполнить недостающие значения в столбце «Возраст» медианным возрастом всех пассажиров в наборе данных.

Мы также можем использовать другие методы для заполнения недостающих значений, например среднее значение, моду или значение, основанное на значениях других переменных. Например, мы можем заполнить недостающие значения в столбце `Embarked` самым распространенным портом посадки:

```
mode_embarked = df['Embarked'].mode()[0]  
df['Embarked'] = df['Embarked'].fillna(mode_embarked)
```

Решение о том, когда использовать среднее, медиану, моду или частое значение для заполнения пропусков, а когда удалить строки с недостающими значениями, зависит от характеристик данных и исследовательского вопроса. Прежде чем принять решение, нужно учесть несколько факторов.

- 1. Процент пропущенных значений.** Если в столбце большой процент пропущенных значений (более 50%), возможно, лучше удалить весь столбец или строку. Если процент невелик, можно использовать методы интерполяции.
- 2. Распределение данных.** Если данные распределены нормально, для восполнения недостающих значений может подойти среднее значение. Если ассиметрично — медиана.

3. **Влияние на анализ или задачу моделирования.** Если пропущенные значения возникают случайно и не влияют на распределение данных, целесообразно использовать методы интерполяции. Если связаны с целевой переменной или другими важными характеристиками, лучше удалить строки с недостающими данными, чтобы избежать смещения в анализе.
4. **Количество доступной информации.** Если в столбце много пропущенных данных, может быть трудно точно оценить среднее значение или медиану. Лучше использовать другой метод интерполяции, например, моду или наиболее частое значение.

Важно тщательно рассмотреть варианты и выбрать метод, который подходит для конкретной ситуации.

## Удаление дубликатов

Мы можем использовать метод `duplicated()` для выявления дублирующихся строк в датасете. Пример того, как это сделать:

```
duplicates = df.duplicated()
```

Далее мы используем булеву серию для индексации исходного датафрейма и выбираем только те строки, которые содержат дубликаты. Мы сохраняем результат в новом датафрейме под названием `duplicates_df`.

```
duplicates_df = df[duplicates]
```

Это позволит отобразить только дубликаты строк в наборе данных Titanic. Как видим, дубликатов в датасете нет. Если бы были, нам бы пригодился метод `drop_duplicates()` для удаления дублирующихся строк.

```
df = df.drop_duplicates()
```

## Исправление некорректных данных

Предположим, что мы хотим исправить данные в столбце `Sex` (пол). Допустим, в нем есть неверные значения, например неправильно написанные или двусмысленные.

Для исправления неверных значений мы можем использовать метод `replace()`:

```
df['Sex'] = df['Sex'].replace('male', 'M')  
df['Sex'] = df['Sex'].replace('female', 'F')
```



В примере мы используем метод `replace()` для замены значения `male` на `M` и значения `female` на `F` в столбце `Sex`.

Мы также можем использовать метод `loc`, чтобы скорректировать значения на основе определенных условий. Предположим, что в столбце `Fare` (цена билета) есть неправильное значение — отрицательное, в чем нет смысла.

Код, который мы можем использовать, чтобы исправить это значение:

```
df.loc[df['Fare'] < 0, 'Fare'] = 0
```

Далее предположим, что мы хотим устранить несоответствия в столбце `Embarked` (порт посадки пассажира). Допустим, есть несоответствия в том, как записываются эти данные: некоторые значения пишутся по-разному, где-то используются сокращения.

Мы можем снова использовать метод `replace()`:

```
df['Embarked'] = df['Embarked'].replace('S', 'Southampton')
df['Embarked'] = df['Embarked'].replace('C', 'Cherbourg')
df['Embarked'] = df['Embarked'].replace('Q', 'Queenstown')
```

В примере мы используем метод `replace()` для замены значения `S` на `Southampton`, значения `C` на `Cherbourg` и значения `Q` на `Queenstown` в столбце `Embarked`.

Мы также можем использовать метод `map()`, чтобы сопоставить значения с шаблоном:

```
port_map = {'Southampton' : 'S', 'Cherbourg' : 'C', 'Queenstown' : 'Q'}
df['Embarked'] = df['Embarked'].map(port_map)
```

В примере мы создаем словарь под названием `port_map`, который сопоставляет неправильные значения в столбце `Embarked` с правильными. Затем мы используем метод `map()`, чтобы применить это отображение к столбцу `Embarked`.

## Категориальные данные

Категориальные данные — это переменные, которые можно разделить на несколько категорий, но у которых нет естественного порядка или ранжирования. Эти категории можно описать с помощью названий или меток.

Категориальные данные отличаются от числовых (или количественных), которые можно упорядочить и измерить.

Два основных типа категориальных данных:

1. **Номинальные данные** — категории, у которых нет естественного порядка или ранжирования. Например:
  - цвета — красный, синий, зеленый;
  - фрукты — яблоко, банан, вишня.
2. **Ординарные данные** — категории, у которых есть порядок, но расстояния между категориями не определены и не согласованы. Например:
  - Уровни образования — средняя школа, бакалавриат, магистратура, аспирантура. Уровни упорядочены, но есть разница в сроке и интенсивности обучения.
  - Шкалы оценок — плохо, удовлетворительно, хорошо, отлично. Порядок есть, но разница в качестве или удовлетворенности между каждым рейтингом не всегда одинакова.

Вернемся к данным из «Титаника». Предположим, мы хотим обработать столбец Sex, который содержит категориальные данные. Чтобы использовать их в модели машинного обучения, нам нужно преобразовать их в числовой формат.

Один из распространенных методов работы с категориальными данными — one-hot encoding. При этом создается новый двоичный столбец для каждого уникального значения в исходном столбце.

```
df = pd.get_dummies(df, columns=['Sex'])
```

Мы используем функцию `get_dummies()` для one-hot encoding столбца Sex. Это создает два новых столбца, один для Sex\_female и один для Sex\_male, и устанавливает значение 1 и 0 для соответствующего пола.

Мы также можем использовать label encoding для преобразования категориальных данных в числовой формат.

```
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
df['Embarked'] = le.fit_transform(df['Embarked'].astype(str))
```

В этом примере мы используем функцию `LabelEncoder()` из библиотеки `scikit-learn` для кодирования в столбце Embarked. Функция преобразует категориальные данные в числовые значения, причем каждому уникальному значению присваивается уникальное целое число.

## Выбросы

Выбросы (outlier) — это точки данных, которые значительно отличаются от остальных данных и могут оказывать существенное влияние на статистический анализ и модели машинного обучения.

Методы работы с выбросами:

- **Удаление.** Удалить выбросы можно с помощью статистических методов, таких как Z-score, IQR или стандартное отклонение. Но удаление может привести к потере данных и потенциальной погрешности анализа.
- **Винсоризация** (Winsorizing). Предполагает установку экстремальных значений на определенный процент. Например, для верхнего 1% точек данных можно установить значение 99-го процентиля.
- **Преобразование данных** с помощью математической функции, которая уменьшает влияние выбросов. Например, для сжатия диапазона значений можно использовать функцию логарифма или квадратного корня.
- **Обработка выбросов как отсутствующих значений.** Выбросы рассматривают как отсутствующие значения и заменяют их средним значением, медианой или модой.

Подход к обработке выбросов зависит от контекста данных и целей анализа или моделирования.

Чтобы выявить отклонения в данных, можно использовать статистические методы, такие как Z-скор или стандартное отклонение. Они предполагают вычисление порогового значения, при превышении которого точки данных считаются выбросами. Например, точка данных, которая отклоняется от среднего значения более чем на 3 стандартных отклонения, может считаться выбросом.

Перейдем к коду. Допустим, мы хотим обработать выбросы в столбце Fare (цена билета). Предположим, что в этом столбце есть выбросы и их нужно обработать.

Для выявления и обработки выбросов мы можем использовать метод Z-score. Нужно рассчитать Z-score для каждой точки данных, который показывает, на сколько стандартных отклонений от среднего значения находится точка данных. Точки данных с Z-score, превышающим определенный порог, считаются выбросами.

Z-score — это статистическая мера, показывающая, на сколько стандартных отклонений точка данных отличается от среднего значения:

- Z-score = 0 — точка данных совпадает со средним значением,
- Z-score = 1 — точка данных на 1 стандартное отклонение выше среднего.
- Z-score = -1 — точка данных на 1 стандартное отклонение ниже среднего и так далее.

Установив пороговое значение для Z-score, мы можем определить точки данных, которые значительно отличаются от остальных и нуждаются в обработке, например в удалении или замене на другое значение.

Пример того, как определить и обработать выбросы в столбце Fare с помощью метода Z-score:

```
# Вычисление Z-score для столбца "Fare"
from scipy import stats
z = np.abs(stats.zscore(df['Fare']))
# Установка порогового значения Z-score
threshold = 3
# Выявление выбросов на основе Z-score
outliers = df['Fare'][z > threshold]
# Замена выбросов медианным значением
df.loc[z > threshold, 'Fare'] = df['Fare'].median()
```

В примере мы сначала рассчитаем Z-score для столбца Fare с помощью функции `zscore()` из библиотеки `scipy.stats`. Затем устанавливаем пороговое значение для Z-score и определяем выбросы на основе порогового значения. Наконец, заменяем выбросы медианным значением для столбца Fare с помощью функции `median()`.

Другой подход к работе с выбросами — это винсоризация, то есть установка экстремальных значений на определенный процентиль.

Винсоризация — это статистический метод, который предполагает замену экстремальных значений в наборе данных менее экстремальными значениями, чтобы уменьшить влияние выбросов.

Представим, что у нас есть набор данных из 10 чисел:

2, 4, 6, 8, 10, 12, 14, 16, 18, 100

В наборе данных число 100 — экстремальное значение, оно может значительно повлиять на любые статистические расчеты с набором данных. Чтобы уменьшить влияние этого выброса, мы можем использовать винсоризацию для замены экстремального значения на менее экстремальное.

Распространенный подход к винсоризации заключается в замене верхнего  $x\%$  и нижнего  $x\%$  значений на значения  $x$ -го и  $(100-x)$ -го перцентилей соответственно. Например, если мы зададим  $x$ , равным 10%, мы заменим верхние и нижние 10% значений на значения 10-го и 90-го перцентилей соответственно.

Используя этот подход, мы можем провести винсоризацию набора данных следующим образом:

2, 4, 6, 8, 10, 12, 14, 16, 18, 18

В коде это можно сделать с помощью функции `winsorize()` из библиотеки `scipy.stats`.

```
from scipy.stats.mstats import winsorize

df['Fare'] = winsorize(df['Fare'], limits=[0.05, 0.05])
```

Другой подход к работе с выбросами заключается в преобразовании данных с помощью математической функции, которая уменьшает их влияние. Одно из распространенных преобразований — функция логарифма, которая сжимает диапазон значений и уменьшает влияние экстремальных значений.

Преобразуем `Fare` с помощью функции логарифма:

```
df['Fare'] = np.log1p(df['Fare'])
```

В примере мы используем функцию `log1p()` из библиотеки `numpy`, чтобы применить преобразование логарифма к столбцу `Fare`.

`log1p()` — это математическая функция, которая вычисляет натуральный логарифм числа плюс 1. Это обычное преобразование для уменьшения влияния экстремальных значений, таких как выбросы. Используется в статистическом анализе и машинном обучении

Представим, что у нас есть набор данных о ценах:

10, 20, 30, 40, 50, 1000

В этом наборе данных значение 1000 — выброс. После использования функции `log1p()` преобразованные цены выглядят следующим образом:

2.39789527, 2.99573227, 3.40119738, 3.71357207, 3.93182563, 6.90875478

Выбор подхода для обработки выбросов будет зависеть от контекста данных и целей анализа или моделирования.

## Стандартизация данных

Стандартизация данных — это распространенная техника для предварительной обработки и анализа данных. Данные преобразуют так, чтобы их среднее значение было равно нулю, а стандартное отклонение — единице.

Преобразование может быть полезно по разным причинам: для повышения эффективности моделей машинного обучения, сравнения данных в разных масштабах или обеспечения нормального распределения данных.

Процесс стандартизации включает вычитание среднего значения и деление на стандартное отклонение. В Python это можно сделать с помощью библиотеки `sklearn`:

```
from sklearn.preprocessing import StandardScaler
data = np.array([[1, 2], [3, 4], [5, 6], [7, 8]])
scaler = StandardScaler()
scaled_data = scaler.fit_transform(data)
print(scaled_data)
```

В примере мы используем набор данных, содержащий 4 строки и 2 столбца. Затем мы создаем объект `StandardScaler`. Наконец, мы преобразуем данные с помощью метода `fit_transform()` и выводим результат. Среднее значение теперь равно нулю, а стандартное отклонение единице, как и ожидалось.

Стандартизация данных не всегда нужна или уместна. Решение о стандартизации должно зависеть от контекста данных и целей анализа или моделирования.

## Очистка данных для задач компьютерного зрения

Итак, мы разобрались с очисткой данных для приложений машинного обучения. В то же время очистка данных — важный этап в задачах компьютерного зрения, где изображения и видео — основные источники данных.

В компьютерном зрении качество и точность данных могут оказать значительное влияние на производительность модели и точность ее прогнозов. Поэтому очистка данных предполагает использование разных методов и подходов, учитывающих уникальные характеристики изображений и видеоданных.

Давайте кратко разберем основные методы, применяемые в этой области.

- **Изменение размера изображений до единого размера** — важный этап предварительной обработки. Одинаковый размер и соотношение сторон изображений облегчает обучение модели и повышает ее точность.
- **Нормализация данных изображения** помогает уменьшить влияние освещения и различий в контрасте. Методы для работы — регулировка яркости и контрастности изображений, преобразование их в градации серого.
- **Увеличение изображения** — создание дополнительных обучающих данных. Методы для работы — переворачивание (зеркальное отображение), поворот или обрезка. Может улучшить производительность модели и снизить риск переоценки.
- **Обесцвечивание изображений** — уменьшение влияния шума и других артефактов на изображениях. Методы для работы — размытие, сглаживание, применение фильтров для удаления шума.
- **Обнаружение объектов** — идентификация и локализация объектов на изображениях. Методы для работы — сегментация (разделение изображения на основе цвета или текстуры), извлечение признаков (определение ключевых особенностей изображений, таких как края или углы).

Используя эти и другие методы очистки данных, мы можем повысить качество и точность изображений и видео для задач компьютерного зрения, обеспечить надежность и достоверность моделей.

Выбор методов очистки данных должен зависеть от контекста и целей моделирования. Иногда нужны эксперименты и итерации, чтобы найти эффективные методы.

## Качество данных. Трансформация данных

Трансформация данных — это процесс их преобразования из одного формата или структуры в другой, чтобы повысить качество или полезность данных для конкретной цели.

Трансформация включает широкий спектр методов и подходов, таких как нормализация, агрегация, фильтрация и разработка признаков (feature engineering).

Процесс преобразования данных можно разделить на две категории:

- **Структурная трансформация** — изменение формата или структуры данных. Например:
  - преобразование данных из одного формата в другой,
  - объединение нескольких наборов данных в один,
  - реструктуризация данных для соответствия рамкам моделирования или анализа.

Структурные преобразования могут повысить эффективность данных, сделать работу с ними более удобной, облегчить анализ и моделирование.

- **Трансформация контента** — изменение фактического содержания или значений данных. Например:
  - масштабирование данных до определенного диапазона,
  - применение математических функций к данным,
  - применение статистических методов для выявления и удаления выбросов.

Трансформация контента может помочь повысить точность и полезность данных, сделать их пригодными для использования.

## Методы структурной трансформации

**Интеграция** — объединение нескольких наборов данных в один. Полезна, когда несколько источников данных нужно объединить в один набор для анализа. Сложность в том, что в структуре и формате данных могут быть различия — может потребоваться очистка и нормализация.

**Реструктуризация данных** — изменение структуры данных для соответствия схеме моделирования или для анализа. Может включать преобразование данных, когда мы меняем местами строки и столбцы, или преобразование данных для создания новых переменных или характеристик.

**Конвертация данных** — изменение формата файла или структуры данных. Например, преобразование данных из CSV в файл Excel или преобразование изображения в другой формат.

**Обобщение данных** — агрегирование данных для создания новых переменных или характеристик. Может включать вычисление суммарной статистики (среднее значение или медиана) или создание новых переменных на основе агрегирования существующих. Помогает упростить данные и уменьшить размер набора, что облегчает анализ и моделирование.



## Методы трансформации контента

**Масштабирование** — нормализация значений в датасете таким образом, чтобы они попадали в определенный диапазон. Полезно для сравнения переменных с разными единицами измерения или для подготовки данных для алгоритмов машинного обучения, которые требуют, чтобы входные данные находились в определенном диапазоне.

**Нормализация** — преобразование значений таким образом, чтобы они соответствовали определенному распределению, например, нормальному.

**Применение математических или статистических функций** может помочь преобразовать данные так, чтобы они были полезны для анализа или машинного обучения. Например, мы можем применить логарифмическую функцию к переменной, чтобы преобразовать ее в более нормально распределенную.

**Преобразование типов данных** — например, преобразование строк в числовые значения или категориальных переменных в фиктивные.

**Квантирование** (биннинг) — это процесс обработки данных, который преобразует непрерывные данные в дискретные путем замены значений диапазонами. Предполагает разделение значений в наборе данных на набор бинов или категорий.

**Извлечение признаков** — преобразование данных для извлечения соответствующих признаков или переменных, которые полезны для анализа или машинного обучения. Например, мы можем извлечь характеристики из текстовых данных, такие как частота определенных слов или фраз.

## Трансформация на практике

А теперь давайте на примерах разберем некоторые методы трансформации данных.

Начнем с интеграции данных с помощью библиотеки Python pandas.

Предположим, у нас есть два набора данных, которые содержат информацию о продажах и уровнях запасов различных продуктов:

	<b>product_id</b>	<b>sales_date</b>	<b>sales_quantity</b>
<b>0</b>	1	2022-01-01	10
<b>1</b>	2	2022-01-01	20
<b>2</b>	1	2022-01-02	15
<b>3</b>	2	2022-01-02	25

	<b>product_id</b>	<b>sales_date</b>	<b>inventory_level</b>
<b>0</b>	1	2022-01-01	100
<b>1</b>	2	2022-01-01	200
<b>2</b>	1	2022-01-02	90
<b>3</b>	2	2022-01-02	180

Чтобы объединить эти два набора данных, мы можем использовать функцию `merge()` в `pandas` — она объединяет два набора данных на основе общего столбца или набора столбцов. В этом случае общими будут столбцы `product_id` и `sales_date`.

```
merged_data = pd.merge(sales, inventory, on=['product_id',
'sales_date'], how='left')
```

Этот код объединяет два набора данных на основе столбцов `product_id` и `sales_date` с помощью функции `merge()` с типом объединения `left join`. В результате создается новый датафрейм под названием `merged_data`, который содержит все столбцы из обоих наборов данных.

	<b>product_id</b>	<b>sales_date</b>	<b>sales_quantity</b>	<b>inventory_level</b>
<b>0</b>	1	2022-01-01	10	100
<b>1</b>	2	2022-01-01	20	200
<b>2</b>	1	2022-01-02	15	90
<b>3</b>	2	2022-01-02	25	180

Интеграцию данных можно выполнять не только в Jupyter-ноутбуке с помощью `Pandas`, но и в базах данных — `MongoDB` и `ClickHouse`. Обе базы данных поддерживают различные методы интеграции, такие как объединение таблиц, агрегирование данных и слияние датасетов.

Следующий пример — реструктуризация данных с помощью библиотеки pandas.

Предположим, у нас есть набор данных, который содержит информацию о продажах разных товаров в разных регионах:

	product_id	region	month	sales_quantity
0	1	North	Jan	10
1	2	North	Jan	20
2	1	South	Jan	15
3	2	South	Jan	25
4	1	North	Feb	12
5	2	North	Feb	22
6	1	South	Feb	18
7	2	South	Feb	28

Для реструктуризации этих данных мы можем использовать функцию `pivot_table()` в pandas, которая позволяет агрегировать данные на основе одного или нескольких столбцов. В этом случае мы можем агрегировать данные на основе столбцов `region` и `month`, чтобы создать новый набор данных, который показывает общее количество продаж для каждого продукта в каждом регионе за каждый месяц.

```
pivot_data = sales2.pivot_table(index='product_id',  
columns=['region', 'month'], values='sales_quantity', aggfunc='sum')
```

Используем функцию `pivot_table()` для агрегации данных на основе столбцов `region` и `month`, столбец `sales_quantity` является значением для агрегирования.

region	North		South	
month	Feb	Jan	Feb	Jan
product_id				
1	12	10	18	15
2	22	20	28	25

В этой таблице каждая строка представляет отдельный продукт, а каждый столбец — различную комбинацию региона и месяца. Значения в таблице показывают общее количество продаж каждого продукта в каждом регионе за каждый месяц.

# Процессинг данных

Процессинг данных — это важнейший компонент инженерии данных, который включает в себя их преобразование и анализ.

Агрегирование, фильтрация, объединение данных из нескольких источников и другие методы процессинга помогут преобразовать данные в более удобный формат и пригодятся для анализа: например, чтобы выявить закономерности, тенденции и аномалии.

Обработку данных можно выполнять разными способами, включая пакетный и потоковый.

- **Пакетный** — обработка большого объема данных одновременно.
- **Потоковый** — обработка данных в режиме реального времени по мере их поступления.

И у пакетной, и у потоковой обработки есть свои преимущества и недостатки.

## Пакетная обработка

Пакетная обработка, как правило, выполняется по расписанию, например ночью или раз в неделю. Такой подход помогает уменьшить влияние на производительность системы в часы пик.

Этапы процесса: сбор данных, подготовка, пакетная обработка.

Ключевые характеристики:

- **Большой объем.** Пакетная обработка предназначена для обработки больших объемов данных. Система может эффективно обрабатывать их, не влияя на производительность в часы пик.
- **Обработка по расписанию.** Пакетная обработка обычно запланирована, например на ночь или неделю. Система может обрабатывать данные в период низкой загрузки, что снижает влияние на ее производительность.
- **Постановка данных.** Данные перед обработкой, как правило, подвергаются стадированию. Постановка включает в себя сбор данных и их подготовку к обработке, например, преобразование форматов или очистку. Постановка позволяет системе убедиться, что данные чистые и правильно отформатированы.

- **Параллельная обработка.** Разбиение обработки на мелкие задачи и их одновременное выполнение. Система сможет повысить производительность и сократить время для обработки больших объемов данных.

Пакетная обработка обычно используется для извлечения, преобразования и загрузки данных. Например, в процессе ETL (Extract, Transform, Load) пакетная обработка подходит для извлечения данных из нескольких источников, преобразования их в стандартный формат, а затем загрузки в хранилище данных для анализа.

Технологии, которые используют в пакетной обработке:

- **Apache Hadoop** — платформа с открытым исходным кодом для хранения и обработки больших объемов данных в распределенной среде. Включает несколько компонентов, в том числе:
  - Hadoop Distributed File System (HDFS) для хранения данных,
  - MapReduce для параллельной обработки данных на кластере узлов.
- **Apache Spark** — распределенная вычислительная система с открытым исходным кодом для обработки больших объемов данных. Включает несколько компонентов, в том числе:
  - Spark Core для обработки данных,
  - Spark SQL для запросов к структурированным данным,
  - Spark Streaming для обработки данных в реальном времени.
- **Apache Flink** — это распределенная вычислительная система с открытым исходным кодом для обработки больших объемов данных. Включает несколько компонентов, в том числе:
  - Flink Core для обработки данных,
  - Flink SQL для запроса структурированных данных,
  - Flink Streaming для обработки данных в реальном времени.
- **Apache Beam** — унифицированная модель программирования с открытым исходным кодом для пакетной и потоковой обработки данных. Предоставляет набор SDK для построения конвейеров данных, которые могут выполняться на различных механизмах исполнения, таких как Apache Flink, Apache Spark и Google Cloud Dataflow.

- **Talend** — это платформа интеграции данных, которая предоставляет набор инструментов для проектирования и развертывания конвейеров данных. Talend включает несколько компонентов, в том числе Talend Studio для проектирования конвейеров данных и Talend Job Server для выполнения и планирования конвейеров данных.

У технологий пакетной обработки данных (Apache Hadoop, Apache Spark и Apache Flink) есть несколько сходств и различий.

Все три технологии предназначены для обработки больших объемов данных в распределенной среде. Все предоставляют набор инструментов и компонентов для проектирования, развертывания и выполнения конвейеров пакетной обработки.

Однако Apache Hadoop в основном предназначен для хранения и обработки данных в пакетном режиме, а Apache Spark и Apache Flink — как для пакетной, так и для потоковой обработки.

Apache Spark и Apache Flink также обеспечивают более унифицированную модель программирования для пакетной и потоковой обработки, в то время как Apache Hadoop требует отдельных инструментов и фреймворков для пакетной и потоковой обработки.

Кроме того, Apache Flink обеспечивает лучшую поддержку сложной обработки событий и обработки данных с низкой задержкой, а Apache Spark — лучшую поддержку машинного обучения и обработки графов.

Выбор технологии пакетной обработки зависит от требований задачи: скорости и стоимости обработки, объема и сложности данных.

## Потоковая обработка

Потоковая обработка — обработка данных в режиме реального времени по мере их получения. Обычно используется в инженерии данных для анализа и преобразования данных, которые постоянно генерируются устройствами интернета вещей (IoT), соцсетями и транзакционными системами.

Этапы процесса: сбор данных, обработка в режиме реального времени, хранение результатов.

Обработка потоков обычно выполняется с использованием распределенных вычислительных сред, таких как Apache Kafka, Apache Flink или Apache Storm.

Ключевые характеристики:

- **Обработка данных в реальном времени**, по мере их получения. Позволяет быстро принимать меры, например, запускать оповещения или автоматизировать процессы.
- **Большие объемы**. Благодаря обработке в реальном времени, система может эффективно обрабатывать большие объемы данных, не требуя больших объемов хранения.
- **Низкая задержка**. Поточковая обработка рассчитана на низкую задержку. Система может быстро обрабатывать данные и реагировать на события в реальном времени.
- **Распределенная обработка**. Поточковая обработка обычно выполняется с использованием распределенных вычислений, что позволяет системе обрабатывать данные параллельно на нескольких узлах. Система может повысить производительность и сократить время для обработки больших объемов данных.

Потоковую обработку обычно используют для фильтрации и агрегации данных, а также для аналитики в реальном времени.

Например, в системе электронной коммерции потоковую обработку можно использовать для анализа поведения клиентов: какие продукты они просматривают и покупают. А также для запуска предложений или скидок в режиме реального времени.

Технологии, которые используют в потоковой обработке:

- **Apache Kafka** — распределенная потоковая платформа с открытым исходным кодом для обработки потоков данных в реальном времени. Включает несколько компонентов, в том числе:
  - Kafka Connect для интеграции с другими источниками данных,
  - Kafka Streams для обработки данных в режиме реального времени.
- **Apache Flink** — распределенная вычислительная система с открытым исходным кодом для обработки как пакетных, так и потоковых данных.
- **Apache Storm** — распределенная система обработки потоков данных в реальном времени с открытым исходным кодом. Включает несколько компонентов, в том числе:

- Spouts для получения данных,
- Bolts для обработки и анализа данных в режиме реального времени.
- **AWS Kinesis** — облачная управляемая потоковая платформа для обработки потоков данных в реальном времени. Включает несколько компонентов, в том числе:
  - Kinesis Data Streams для обработки данных в реальном времени,
  - Kinesis Data Firehose для загрузки данных в другие хранилища данных.
- **Google Cloud Pub/Sub** — облачная управляемая служба обмена сообщениями для обработки потоков данных в реальном времени. Включает несколько компонентов, в том числе:
  - Topics для публикации и подписки на потоки данных,
  - Subscriptions для обработки и анализа данных в режиме реального времени.

## Обзор систем управления данными

Управление данными — это процесс управления доступностью, удобством использования, целостностью и безопасностью данных в организации. А системы управления данными — это набор принципов, политик и процедур для этого процесса.

Разберем несколько примеров систем управления данными.

**DAMA-DMBOK** (Data Management Body of Knowledge) — общепризнанная система управления данными. Предоставляет полный набор рекомендаций по дата-менеджменту. Охватывает 11 областей управления данными, включая их архитектуру, моделирование, качество и безопасность.

Концепция DAMA-DMBOK призвана помочь организациям установить общее понимание практики и терминологии управления данными.

Пример из DAMA-DMBOK, который иллюстрирует важность обработки данных, — концепция интеграции. Интеграция данных — это процесс объединения данных из разных источников в единое представление для поддержки бизнес-процессов и принятия решений.



Предположим, что розничная компания хочет проанализировать данные о продажах из своих онлайн- и офлайн-магазинов, чтобы получить представление о поведении и предпочтениях покупателей. Компания может использовать:

- пакетную обработку, чтобы извлечь данные из разных источников, преобразовать их в стандартный формат и загрузить в хранилище для анализа;
- потоковую обработку, чтобы проанализировать поведение покупателей в режиме реального времени и предоставить им персонализированные рекомендации на основе истории просмотров и покупок.

То есть выбор метода интеграции данных будет зависеть от бизнес-требований и характеристик интегрируемых данных.

**COBIT** (Control Objectives for Information and Related Technology) — комплексная система управления ИТ, которая включает управление данными в качестве одного из своих компонентов.

Концепция предоставляет набор лучших практик для управления данными, включая управление их качеством, конфиденциальность и безопасность.

COBIT призвана помочь организациям согласовать свои ИТ- и бизнес-цели и улучшить управление данными.

**Система ISO 8000** — это серия международных стандартов по управлению качеством данных. Предоставляет набор рекомендаций по управлению качеством данных, включая их точность, полноту и согласованность. Призвана помочь организациям создать систему управления качеством данных, которая соответствует их бизнес-целям и удовлетворяет потребности заинтересованных сторон.

**Система FAIR** (Findable, Accessible, Interoperable, and Reusable) — набор руководящих принципов, который позволяет сделать данные находимыми, доступными, взаимозаменяемыми и пригодными для повторного использования. Призвана помочь организациям повысить ценность и удобство использования своих данных.

В целом, системы управления данными обеспечивают структурированный подход к эффективному управлению данными. Приняв систему управления данными, компании могут установить набор руководящих принципов, политик и процедур для управления данными, обеспечения их доступности, пригодности для использования, безопасности и высокого качества.

# Заключение

В этой лекции мы рассмотрели темы, связанные с очисткой и преобразованием данных. Начали с очистки: поговорили про ее важность и методы — заполнение пропущенных значений, удаление дубликатов и обработку выбросов. Затем рассмотрели трансформацию контента и структурные трансформации, такие как интеграция и реструктуризация данных.

Цель очистки и преобразования — подготовка данных к анализу или машинному обучению, а также обеспечение того, чтобы у них был один формат, подходящий для решения конкретной задачи. Тщательно очищая и преобразуя данные, мы можем гарантировать, что наши аналитические данные и модели машинного обучения будут точными, надежными и полезными для принятия обоснованных решений.

## Что можно почитать еще?

1. [Очистка данных: проблемы и современные подходы](#)
2. [EDA with Pandas](#)
3. [Трансформация данных в pandas, ч.1](#)
4. [Библиотека pandas — объединение таблиц, группировки, таблицы сопряженности и сводные таблицы](#)
5. [Управление данными. DAMA-DMBOK/реальность](#)