Конспект по теме "Работа с пропусками"

Метрики эффективности источника трафика

Теория

Существует множество **источников трафика**:

- Поисковая выдача (органический трафик)
- Контекстная реклама
- E-mail рассылка
- Социальные сети
- Переходы по ссылкам с других сайтов

Цель введения метрик эффективности источника трафика - оценка и сравнение источников трафика для выявления лучших. Знание эффективности источников позволяет оперативно управлять маркетинговой стратегией.

Имеется две методики подсчёта **конверсии сайтов**. Первая - считается **доля посетителей** сайта, совершивших **целевые действия**. Вторая - считается **доля целевых действий**.

Визит - последовательность действий посетителя от перехода на сайт и до момента, когда пользователь ничего больше не делал в течение 30 минут.

Ещё одна важная метрика для определения эффективности источника трафика - **доля повторных покупателей.** Эта метрика рассчитывается как отношение числа посетителей, совершивших хотя бы две покупки, к числу посетителей, совершивших хотя бы одну покупку.

Практика

В pandas можно делать **арифметические операции** над столбцами: сложение, вычитание, умножение, деление. Например:

```
data['column1'] = data['column12'] + data['column3']
```

User ID и куки

Теория

Информацию о поведении посетителей веб-страницы собирает специальный **счётчик** — несколько строчек кода в коде сайта — и отправляет её в **системы веб-аналитики**, например, Яндекс.Метрику. Счётчик собирает: общие сведения о посетителях, с какого источника трафика они заходят, просмотр конкретных страниц пользователем и покупки. В счётчиках каждому пользователю присваивается **уникальный номер**, который нужен чтобы отличить пользователя от остальных - **User ID**.

Данные счётчиков - "сырые", они затем с помощью систем веб-аналитики превращаются в отчёты об аудитории, посещаемости и источниках. В **отчётах** можно комбинировать разные метрики и визуализировать результаты.

Для определения пользователя, который повторно зашёл на сайт, применяются **куки** - специальные текстовые файлы, которые остались в памяти устройства после первого посещения и при повторном визите отправляются на сервер.

Но если пользователь заходит с разных браузеров, из-за того что у каждого браузера свои куки, ему присваиваются разные User ID, поэтому собираются дополнительные данные, например, e-mail. Для защиты персональных данных, User ID и email зашифровываются.

Текстовые файлы с информацией о посещении сайта называются **логами**.

Когда к набору полученных определённым образом данных добавляют новую информацию, это называется **обогащение данных**. В датасете

могут встречаться пропущенные данные. Иногда, их можно проигнорировать, а иногда нужно их обработать, заполнить для анализа.

Практика

Для поиска уникальных значений в столбце, применяется метод `unique()` : `data['column'].unique()`.

Для удаления строк с пропущенными значениями нужно вызвать метод `dropna()`, а для перенумерации - `reset_index()` с аргументом `drop=True`.

Вы обнаружили *NaN и None*

Теория

NaN и *None* - эти особые значения указывают, что никакого значения нет. *NaN* отвечает за отсутствующее в ячейке число. Его тип данных *float*, поэтому с *NaN* можно проводить математические операции. *None* принадлежит к нечисловому типу *NoneType,* и математические операции с ним неосуществимы. Значения *NaN* могут привести к некорректным результатам при группировке данных. Строки с этими значениями не всегда стоит удалять: часто пропуски можно восстановить.

Практика

В pandas, метод `value_counts()` возвращает уникальные значения с их количеством.

Метод `isnull()` возвращает булевский список, в котором `True` означает, что значение в колонке пропущено.

Для замены пропусков на какое-то значение, применяется метод `fillna()` с аргументом `value`.

Категориальные и количественные переменные

Переменные бывают двух типов: категориальные и количественные. **Категориальная переменная** принимает одно значение из ограниченного

набора, а **количественная** — любое числовое значение в диапазоне. Количественные переменные, в отличие от категориальных. обладают возможностью сравнения.

Также переменные могут быть **логическими (булевыми)**. Такие переменные указывают на истинность или ложность какого-либо события. Если событие истинно, то переменная принимает значение 1, соответствующее True, а если ложно — 0, соответствующий False.

Работа с пропусками в категориальных переменных

Теория

Перед обработкой пропусков, нужно ответить на вопрос, существует ли *закономерность* в появлении пропусков. Иными словами, *не случайно ли* их возникновение в наборе данных.

Пропуски бывают трёх типов:

- **Полностью случайные**: если вероятность встретить пропуск не зависит ни от каких других значений. Ответ на этот вопрос не зависит от характера самого вопроса и от других вопросов анкеты, а сам пропуск легко восстановить по имени.
- **Случайные:** если вероятность пропуска зависит от других значений в наборе данных, но не от значений собственного столбца. Пропущенное значение связано с тем, что, например, такой категории не существует.
- **Неслучайные:** если вероятность пропуска зависит от других значений, в том числе и от значений собственного столбца. Отсутствующее значение зависит как от характера вопроса, так и от значения переменной в другом столбце.

Практика

Существует несколько вариантов замены пропусков категориальных значений. Например, замена значением по умолчанию. Такой вариант хорошо подойдёт для заполнения случайных пропусков. В pandas для этого применяется метод `fillna()`.

Не все пустые значения можно заполнить методом `fillna()`. Например, к пропущенным значениям *None* его не применить — метод распознаёт только значения *NaN* в таблице. Для замены *None* вызывают метод `loc`. Логическая индексация позволит выделить все строки в необходимом столбце, которые содержат *None*, и заменить их на новое значение.

Для применения некоторых функций к определённым столбцам, применяется метод `agg()`. Название столбца и сами функции записываются в структуру данных — **словарь**. Словарь состоит из **ключа** и **значения**. Ключ - это название столбца, к которому нужно применить функции, а значением выступает список с названиями функций.

```
{'column':['function1','function2']}
```

После применения метода `agg()` , названия столбцов стали «двойными». Чтобы обратиться к результату применения функции `['function1']` к столбцу `['column']`, просто укажите их подряд:

```
data['column']['function1']
```

Работа с пропусками в количественных переменных

Теория

Пропуски в количественных переменных заполняют *характерными значениями*. Это значения, характеризующие состояние **выборки**, - набора данных, *выбранных* для проведения исследования. Чтобы примерно оценить типичные значения выборки, годятся **среднее арифметическое** или **медиана**.

Среднее арифметическое — это сумма всех значений, поделённая на количество значений.

Медиана — это такое число в выборке, что ровно половина элементов больше него, а другая половина — меньше.

Практика

Для получения среднего арифметического применяется метод `mean()`. Его применяют ко всей таблице, к отдельному столбцу или к сгруппированным данным.

Для нахождения медианы есть специальный метод `median()`, его можно применять к таблице, столбцу или сгруппированным данным.

Конспект по теме "Изменение типов данных"

Как читать файлы из Excel

Для прочтения файлов Excel есть особый метод `read_excel()`. Он похож на `read_csv()`, но в отличие от него, `read_excel()` нужно два аргумента: строка с именем самого файла или пути к нему, и имя листа `sheet_name`. Если аргумент `sheet_name` пропущен, то по умолчанию прочитается первый по счёту лист

```
import pandas as pd
df = pd.read_excel('file.xlsx', sheet_name='List1')
```

Перевод строковых значений в числа

Для того, чтобы перевести строковые значения в числа, есть стандартный метод Pandas — **to_numeric()**. Он превращает значения столбца в числовые типы *float64* (вещественное число) или *int64* (целое число) в зависимости от исходного значения.

У метода *to_numeric()* есть параметр **errors**. От значений, принимаемых *errors*, зависят действия *to_numeric* при встрече с некорректным значением:

- `errors='raise'` — дефолтное поведение: при встрече с некорректным значением выдается ошибка, операция перевода в числа прерывается;
- `errors='coerce'` — некорректные значения *принудительно* заменяются на NaN;
- `errors='ignore'` — некорректные значения *игнорируются*, но остаются.

Для того, чтобы перевести данные в нужный тип, применяется метод `astype()`. В качестве аргумента передаётся строка с названием типа.

Методы Pandas для работы с датой и временем

Для работы с датой и временем в Python существует особый тип данных — **datetime**.

Чтобы перевести строку в дату и время, используется метод **to_datetime()**. Параметрами метода являются: столбец, содержащий строки, и формат даты в строке.

Формат даты задаётся с помощью специальной системы обозначений, где:

- %d — день месяца (от 01 до 31)
- %m — номер месяца (от 01 до 12)
- %Y — год с указанием столетия (например, 2019)
- %H — номер часа в 24-часовом формате
- %I — номер часа в 12-часовом формате
- %M - минуты (от 00 до 59)
- %S - секунды (от 00 до 59)

```
date['column']= pd.to_datetime(date['column'], format='%d.%m.%YZ%H:%M:%S')
```

Среди самых разнообразных способов представления даты и времени особое место занимает формат **unix time**. Его идея проста — это количество секунд, прошедших с 00:00:00 1 января 1970 года. *Unix*-время соответствует Всемирному координированному времени, или *UTC*.

Метод `to_datetime()` работает и с форматом *unix time*. Первый аргумент — это столбец со временем в формате unix time, второй аргумент `unit` со значением `'s'` сообщит о том, что нужно перевести время в привычный формат нужно с точностью до секунды.

Часто приходится исследовать статистику по месяцам, дням, годам. Чтобы осуществить такой расчёт, нужно поместить время в класс *DatetimeIndex* и применить к нему атрибут *month, day, year*:

```
date['column'] = pd.DatetimeIndex(date['column']).month
```

Обработка ошибок с помощью try-except

Выгружая данные из разных систем, нужно быть готовым к следующим трудностям:

- Некорректный формат приводит к невыполнению кода. Говорят, что *«код падает с ошибкой»*.
- Ошибки в данных встречаются ближе к концу файла, и код на строках с неверными значениями не выполняется. Значит, пропадают расчёты для предыдущих, правильных строк;
- Данные могут поменяться.

К сожалению, предсказать все потенциальные ошибки невозможно. Для работы с непредсказуемым поведением данных есть конструкция **try-except**. Принцип работы такой: исходный код помещают в блок *try*. Если при выполнении кода из блока *try* возникнет ошибка, воспроизведётся код из блока *except*.

```
try:
  # код, где может быть ошибка
except:
  # действия если возникла ошибка
```

Метод merge()

Данные хранятся в excel-таблице из нескольких листов. Для того, чтобы использовать данные на всех листах, нужно склеить таблицы.

Объединить несколько таблиц в одну поможет метод `merge()`.

Аргументы:

- *right* - имя DataFrame или Series, который нужно присоединить к исходной таблице
- *on* - общее поле в двух таблицам, по которым происходит соединение
- *how* - какие *id* включены в итоговую таблицу. Может принимать значения *left* - id из левой таблицы будут включены в итоговую таблицу или *right* - id из правой таблицы будут включены в итоговую таблицу.

Сводные таблицы

Сводная таблица - это ваш помощник для обобщения данных и их наглядного представления.

В Pandas для подготовки сводных таблиц есть метод `pivot_table()`

Аргументы метода:

- *index* - столбец или столбцы, по которым происходит группировка данных
- *columns* - столбец по значениям которого будет происходить группировка
- *values* - значения, по которым мы хотим увидеть сводную таблицу
- *aggfunc* - функция, которая будет применяться к значениям

Конспект по теме "Поиск дубликатов"

Ручной поиск дубликатов

Часто при анализе данных возникают дубликаты. Если не найти дубликаты, то анализ данных может привести к некорректным результатам.

Дубликаты можно искать двумя способами.

Способ 1. Ранее вы уже знакомились с методом `duplicated()`. В сочетании с методом `sum()` он возвращает количество дубликатов. Если выполнить метод `duplicated()` без суммирования на экране будут отображены все строки. Там, где есть дубликаты, будет значение `True`, где дубликата нет - `False`.

Способ 2. Вызвать метод `value_counts()`, возвращающий уникальные значения с их частотой. Его применяют к объекту *Series*. Результат работы метода — список пар «значение-частота», отсортированный по убыванию. Значит, интересующие нас дубликаты будут в начале списка.

Ручной поиск дубликатов с учетом регистра

Дубликаты в строковых данных требуют особого внимания, поскольку регистр имеет значение: заглавная `'A'` и строчная `'a'` с точки зрения python - разные символы, но имеют одинаковое значение - буква *A*.

Чтобы учесть такие дубликаты, все символы в строке приводят к **нижнему регистру** вызовом метода `lower()`.

В pandas символы приводят к **нижнему регистру** методом с похожим синтаксисом: `str.lower()`.

Стемминг

При разделении строк по категориям, проверка вхождения определённых подстрок в строку не всегда может дать корректный результат. Один из

приемлемых вариантов для решения такой задачи - стемминг.

Стемминг - это процесс нахождения основы заданного слова, называемой **стемом**.

В Python для стемминга есть специальная библиотека NLTK. Он содержит стеммеры - специальные объекты, содержащие правила определения стемов. Метод `stem()` применяется для извлечения стема для всех слов в строке.

```
from nltk.stem import SnowballStemmer
russian_stemmer = SnowballStemmer('russian')
russian_stemmer.stem(text)
```

Лемматизация. Библиотека PyMystem

Теория

Стемминг — не единственный алгоритм для поиска слов, записанных в разных формах. Более продвинутый процесс — **лемматизация**, или приведение слова к его словарной форме **(лемме)**.

В русском языке формы записи в словаре (**леммы**) следующие:

- для существительных — именительный падеж, единственное число;
- для прилагательных — именительный падеж, единственное число, мужской род;
- для глаголов, причастий, деепричастий — глагол в инфинитиве несовершенного вида.

Практика

Одна из библиотек с функцией **лемматизации** на русском языке — *pymystem3*, разработана сотрудниками Яндекса. Для многих случаев это гораздо лучше основы слова, которую возвращает *NLTK*. *pymystem3* по умолчанию выдает **список лемматизированных слов** (слов, сведённых к лемме), а *NLTK* — строку.

```
# pymystem3 импортируется так:
from pymystem3 import Mystem
m = Mystem()
lemmas = m.lemmatize(text)
```

Для подсчёта встречаемости значений в списке используется специальный контейнер Counter из модуля collections.

```
from collections import Counter
print(Counter(lst))
```

Конспект по теме "Категоризация данных"

Знакомство с данными

Кроме внутренних преобразований важны и внешние. Необходимо привести данные в удобный и читаемый вид.

Одна из подзадач в этом процессе - именование столбцов. Для этого применяется метод `set_axis()`.

Классификация по типу

В датасетах могут встречаться категории в виде названий, их длина может быть различной.

К чему приводит такой способ хранения?

- Усложняется визуальная работа с таблицей;
- Увеличивается размер файла и время обработки данных;
- Чтобы отфильтровать данные по типу обращения, приходится набирать его полное название (а в нём можно допустить ошибки);
- Создание новых категорий и изменение старых отнимает много времени.

Для того, чтобы оптимально хранить информацию о категориях, создаётся отдельный файл-словарь, названию категории соответствует номер. Этот номер в дальнейшем используется вместо текстового наименования категории в таблице.

Классификация по возрастным группам

Часто объекты, имеющие определённое значение признака, присутствуют в наборе только единожды. Работать с такими отрывками и делать из них статистические выводы нельзя. Поэтому, с такими данными проводится **категоризация -** объединение данных в категории.

Одним из вариантов категоризации является выделение возрастных групп, например: до 18, от 18 до 65, старше 65.

Подобные правила классификации в python удобно представлять в виде функций, которые принимают параметр - значение признака, а возвращают соответствующую категорию.

Для того, чтобы получить столбец с группой на основе столбца с каким-то другим признаком, можно воспользоваться написанной нами функцией `group` и методом `apply()`.

```
data['column_group'] = data['column'].apply(group)
```

Функция для одной строки

Когда для категоризации недостаточно значения в одном каком-то столбце, то в функцию можно передать и всю строку как *Series*. В теле этой функции можно также получать значения в каком-то определённом столбце.

Применение метода apply() в случае обработки строки имеет два отличия:

1) Метод `apply()` вызывается не к столбцу data`['age']`, а к датафрейму `data`;

2) По умолчанию Pandas передаёт в функцию `group()` столбец. Чтобы на вход в функцию отправлялись строки, нужно указать параметр `axis = 1` метода `apply()`.

Конспект по теме "Первые графики и выводы"

Знакомство с задачей

Данные в формате *csv* в качестве разделителя могут иметь не только запятые, но и точки с запятой, знаки табуляции или другие символы. Десятичные дроби, записанные с запятой, тоже могут внести путаницу.

В параметрах функции `read_csv()` можно указать, *к*акими символами разделять колонки и дроби. Разделитель колонок задают параметром **sep** , а дробей — параметром **decimal**:

```
file = pd.read_csv('file.csv',  sep=';' , decimal=',')
```

Сводные таблицы для расчёта среднего

Занимаясь предобработкой данных, вы применяли `pivot_table()` — метод для построения сводных таблиц. Прежде значением *aggfunc* вы указывали *sum*, то есть складывали элементы столбца. Если параметр *aggfunc* не указывать, то по умолчанию метод `pivot_table()` рассчитает среднее арифметическое значений, указанных в параметре *values*.

Базовая проверка данных

В работе с данными почти всегда вас ждут сюрпризы:

- Из-за непонимания задачи или случайно выгрузили не те или неполные данные.
- Ошибки в алгоритмах, считающих нужное значение
- Не тот формат предоставляемых данных.
- Упущен какой-нибудь существенный факт

Словом, в данных может быть всё, что угодно. Именно вы как аналитик ручаетесь за их реалистичность. Попробуйте оценить, насколько они достоверны. Начните с базовых проверок. Например, несложно ответить на

вопросы по данным и самостоятельно либо с помощью коллег оценить, похожи ли результаты ваших расчётов на правду.

Базовая проверка может обнаружить проблему в данных. Или наоборот — свидетельствовать, что с ними всё в порядке. По крайней мере, пока.

Гистограмма

Гистограмма — это график, который показывает, как часто в наборе данных встречается то или иное значение. Гистограмма объединяет числовые значения по диапазонам, то есть считает частоту значений в пределах каждого интервала. Её построение подобно работе знакомого вам метода `value_counts()`, подсчитывающего количество уникальных значений в списке, однако `value_counts()` группирует строго одинаковые величины и хорош для подсчёта частоты в списках с категориальными переменными.

В *Pandas* гистограмму строит специальный метод `hist()`, применяемый к списку или к столбцу датафрейма. Метод `hist()` находит в наборе чисел минимальное и максимальное значения, а полученный диапазон делит на области, или корзины. Затем `hist()` считает, сколько значений попало в каждую корзину, и отображает это на графике. Параметр **bins** определяет, на сколько областей делить диапазон данных, по умолчанию таких `bins=10`. По умолчанию, гистограмма выводится для всех значений от минимального до максимального. Масштаб можно изменить вручную, указав параметр **range:** `range=(min_value, max_value)`.

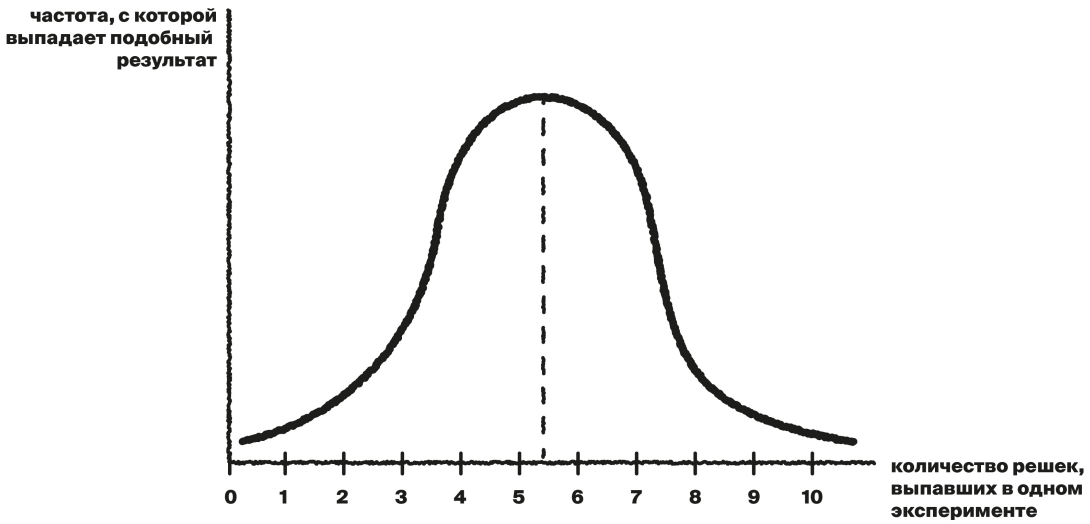
Для создания графиков (в том числе гистограмм), импортируют библиотеку **matplotlib**. Метод `show()` отображает графики.

```
import pandas as pd
import matplotlib.pyplot as plt # импортируем библиотеку, стандартно используется имя plt

pd.Series(...).hist(bins=n_bins, range=(min_value, max_value)

plt.show() # даём команду отобразить гистограмму
```

Распределения

Распределение - это все возможные значения переменной с частотой их появления. Различные распределения:

- **Нормальное**: чаще всего встречается среднее значение и близкие к нему, а крайние значения встречаются довольно редко



- **Распределение Пуассона:** число событий в единицу времени, если они в среднем происходят с измеренной частотой

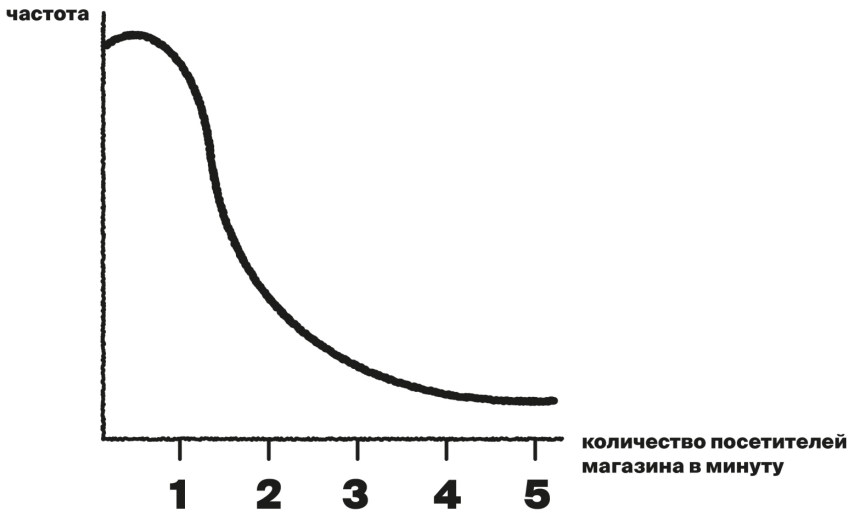


Диаграмма размаха

Характерный разброс — то, какие значения оказались вдали от среднего, и насколько их много.

Если считать характерным разбросом расстояние между минимальным и максимальным значением, то мы не всегда получим точное описание данных, на него могут повлиять выбросы. Поэтому, в качестве характерного разброса применяют межквартильный размах

Квартили разбивают упорядоченный набор данных на четыре части: **первый квартиль** *Q1* — число, отделяющее первую четверть выборки (25% элементов меньше, а 75% — больше него); **медиана — второй квартиль** Q2 (половина элементов больше и половина меньше неё); **третий квартиль** Q3 — это отсечка трёх четвертей (75% элементов меньше и 25% элементов больше него).

Межквартильный размах — это расстояние между первым квартилем *Q1* и третьим квартилем *Q3*.

Диаграмма размаха, или **ящик с усами**, позволяет отобразить все квартили для заданных данных.

«Ящик» ограничен первым и третьим квартилями. Внутри ящика обозначают медиану.

«Усы» простираются влево и вправо от границ ящика на расстояние, равное 1,5 **межквартильным размахам** (*IQR*). В размах «усов» попадают нормальные значения, а за пределами находятся выбросы, изображённые точками. Если правый «ус» длиннее максимума, то он заканчивается максимумом. То же — для минимума и левого уса.

В Python диаграмму размаха строят методом `boxplot()`, он позволяет визуально оценить характеристики распределения, не прибегая к гистограмме.

```
import matplotlib.pyplot as plt
data.boxplot()
plt.show()
```

Оси любого графика в pandas можно изменять, для этого нужно применить метод `xlim(x_min, x_max)` для оси X и `ylim(y_min, y_max)` для оси Y. Параметры в обоих случаях - минимальное и максимальное значение на графике

```
import matplotlib.pyplot as plt
plt.xlim(x_min, x_max)
plt.ylim(y_min, y_max)
```

Описание данных

С помощью гистограмм и диаграмм размаха можно получить графическое описание любого набора данных. Однако, не всегда по графикам можно

определить такие характеристики, как среднее, медиану, количество наблюдений в выборке и разброс их значений — **числовое описание данных**. В *Pandas* для этого применяется метод `describe()`.

Стандартное отклонение — числовая характеристика данных, входящая в числовое описание данных и характеризующая разброс величин, показывает, насколько значения в выборке отличаются от среднего арифметического. Оно позволяет понять природу распределения и определить, насколько значения однородны.

```
data['column'].describe()
```

Конспект по теме «Изучение срезов данных»

Срезы данных и поиск авиабилетов

В ходе работы над задачей, часто нужны не все данные, а лишь их часть, или **срез данных**.

Срез данных можно получить, применив фильтр. Этим фильтром может стать **булев массив**, состоящий из *True* и *False*. Так, значениями *True* отметим, что хотим включить определённую строку в срез данных. Формировать этот фильтр вручную - занятие утомительное, а на больших датасетах - практически невозможное, поэтому в pandas применяются автоматические фильтры. Ниже приведён пример автоматического фильтра.

```
data['column'] == 'value'
```

Этот фильтр в дальнейшем служит как индекс датафрейма:

```
data[data['column'] == 'value']
```

Условием создания фильтра может выступать не только равенство, но и другие операции сравнения: `!=`, `>`, `>=`, `<`, `<=`. Значения в столбцах можно сравнивать и с заданными значениями, и между собой:

```
data['column1'] > data['column2']
```

В условиях допустимы арифметические операции:

```
data['column1'] / 2 > data['column2'] + 0.5
```

Чтобы проверить наличие конкретных значений в столбце, вызовем метод `isin()`:

```
data['column'].isin(('value1', 'value2', 'value3'))
```

Иногда нужно получить срез, соответствующий сразу нескольким условиям — для этого существуют логические операции. Отдельные условия указывают в скобках:

Name	Описание	Синтаксис
ИЛИ	Результат выполнения — *True*, если **хотя бы одно из условий** — *True*	`(df['a'] > 2) | (df['c'] != 'Y')`
И	Результат выполнения логической операции *True*, только если **оба условия** — *True*	`(df['a'] > 2) & (df['c'] != 'Y')`
НЕ	Результат выполнения — *True*, если условие — *False*	`~ ((df['a'] > 2) | (df['c'] != 'Y'))`

Срезы данных методом query()

Другим, более простым методом получения срезов является метод `query()`. Необходимое условие для среза записывается в строке, которую передают как аргумент методу `query()`. А его применяют к датафрейму. В результате получаем нужный срез.

```
data.query('column == value')
```

Условия, указанные в параметре `query()`:

- поддерживают разные операции сравнения: `!=`, `>`, `>=`, `<`, `<=`, - а также математические операции
- допускают вызов методов к столбцам: `column.mean()`
- проверяют, входят ли конкретные значения в список, конструкцией: `column in ("value1", "value2", "value3")`. Если нужно узнать, нет ли в списке определённых значений, пишут так: `a not in ("value1", "value2", "value3")`.
- работают с логическими операторами в привычном виде, где «или» — *or* , «и» — *and*, «не» — *not*. Указывать условия в скобках

необязательно. Без скобок операции выполняются в следующем порядке: сначала *not*, потом *and* и, наконец, *or*.

Также, в методе `query()` можно получать значения внешних переменных:

```
variable = 2
data.query('column > @variable')
```

Работа с датой и временем

При переводе данных из строки в тип *datetime*, как вы знаете, применяется метод `pd.to_datetime()`. Зачастую, pandas пытается самостоятельно угадать формат данных. Однако в таких случаях стоит указывать параметр `yearfirst=True`, который означает, что в переданной строке сначала идёт год. Но всё же рекомендуется прописывать формат вручную, чтобы избежать ошибок.

Для доступа к отдельным компонентам даты и времени используется атрибут `.dt` столбца с датой и временем:

```
data['datetime'].dt.year
data['datetime'].dt.weekday
```

Если нужно прибавить или убавить время, то используется `pd.Timedelta()`, которому можно передать желаемый сдвиг в днях, часах, минутах и т.д., передавая соответствующие параметры. Обратите внимание, что переход через 24 часа автоматически приводит к изменении даты, нам не нужно заниматься этой арифметикой.

```
data['shifted_datetime'] = data['datetime'] + pd.Timedelta(hours=10)
```

Также нам пригодится округлённое время, для этого в pandas есть метод `.dt.round()`. В качестве параметра передаётся шаг округления строкой вида '1H', что означает 1 час (H - hour - час). Не обязательно округлять с шагом в 1 час. Можем округлять и с шагом в несколько часов. Другие популярные единицы округления:

- 'D' day день

- 'H' - hour - час
- 'T' или 'min' - minute - минута
- 'S' - second - секунда

```
data['datetime_round'] = df['datetime'].dt.round('3H')
```

Если нужно округлить в меньшую или большую сторону, вместо метода `round()` используются методы `floor()` и `ceil()`, соответственно

Графики

За построение графиков в *Pandas* отвечает метод `plot()`, который строит графики по значениям столбцов из датафрейма. На оси абсцисс *(x)* расположились индексы, а на оси ординат *(y)* — значения столбцов.

```
data.plot()
```

У метода `plot()` есть параметр `style`, который отвечает за стиль отображения точек:

- `'o'` - вместо непрерывной линии, отображается каждая точка кружком
- `'x'` - вместо непрерывной линии, отображается каждая точка символом 'x'
- `'o-'` - отображается непрерывная линия и точка

Можно изменить оси: напрямую указать, какой столбец отвечает за какую ось

```
data.plot(x='column_x', y='column_y')
```

Границы осей можно скорректировать параметрами `xlim` и `ylim`, как и в случае с ящиком с усами:

```
data.plot(xlim=(x_min, x_max), ylim=(y_min, y_max))
```

Для отображения сетки, указывается параметр `grid`, равный *True*:

```
data.plot(grid=True)
```

Размером графика управляют через параметр `figsize`. Ширину и высоту области построения в дюймах передают параметру в скобках.

```
data.plot(figsize = (x_size, y_size))
```

Группировка с pivot_table()

Когда данных много, точки на графиках сливаются и из визуального представления мало что можно понять. Для решения этой проблемы применяется группировка данных. Пример ниже применяет группировку, благодаря которой график становится гораздо информативнее.

```
(data
     .query('column_id == "value"')
     .pivot_table(index='column1', values='column2')
     .plot(grid=True, figsize=(12, 5))
)
```

С помощью графиков с группировкой можно обнаружить выбросы, которые до этого не были видны. Так, иногда из-за выбросов серьезно завышается среднее арифметическое. Как сделать так, чтобы аномально высокие значения не завышали среднее арифметическое? Решений этой задачи может быть два:

- Убрать выбросы
- Вместо среднего арифметического считать медиану. Медиана устойчива к выбросам, но всё же не безупречна: пики всё ещё могут отображаться

Сохраняем результаты

В ходе работы вы можете обнаружить, что в данных, которые вам предоставили, содержится ошибка. В таком случае, нужно сформулировать проблему, чтобы упростить поиск потенциальной ошибки в алгоритме выгрузки данных. Правильное сообщение об ошибке,

или **баг-репорт**, должно чётко объяснять, в чём именно ошибка и как её найти. Коллеги, отвечающие за выгрузку, *ничего о результатах нашего исследования не знают*. Поэтому нужно чётко формулировать, *где мы видим проблему*.

Конспект по теме «Работа с несколькими источниками данных»

Срез по данным из внешнего словаря

При работе со срезами, можно использовать внешние переменные не только числового или строкового типа, но и листы, словари, серии и даже датафреймы. Обращаться к ним можно, как к обычным внешним переменным. Если для среза используется лист, то проводится проверка, что значение в определённой колонке входит или не входит в лист:

```
our_list = [1, 2, 3]
print(data.query('column in @our_list'))
```

Аналогичная проверка осуществляется с помощью словаря, но проверяется, что значение в определённой колонке входит или не входит в ключи словаря:

```
our_dict = {0: 10, 1: 11, 2: 12}
print(data.query('column in @our_dict'))
```

Когда в переменной сохранён объект *Series,* конструкция `column in @our_series` проверит вхождение в список *значений*, а не *индексов:*

```
our_series = pd.Series([10,11,12])
print(data.query('column in @our_series'))
```

Если нужно проверить вхождение в индекс, это указывают явно, дописывая *index* через точку: `column in @our_series.index`:

```
our_series = pd.Series([10,11,12])
print(data.query('column in @our_series.index'))
```

Когда имеют дело с объектом *DataFrame*, вхождение в индекс проверяют так же, как в *Series* — приписав *index* через точку к имени датафрейма:

```
our_dataframe = pd.DataFrame ({
'column1': [2, 4, 6],
'column2': [3, 2, 2],
'column3': ['A', 'B', 'C'],
```

```
})
print(data.query('column in @our_dataframe.index'))
```

Для проверки вхождения в какой-либо столбец, передают его имя:

```
our_dataframe = pd.DataFrame ({
'column1': [2, 4, 6],
'column2': [3, 2, 2],
'column3': ['A', 'B', 'C'],
})
print(data.query('column in @our_dataframe.column2'))
```

Добавляем столбец

Несколько гистограмм можно отобразить на одном графике. Для этого можно применять следующую конструкцию:

```
ax = data1.plot(kind='hist', y='column1', histtype='step', range=(0, 500), bins=25,
                              linewidth=5, alpha=0.7, label='raw')
data2.plot(kind='hist', y='column1', histtype='step', range=(0, 500), bins=25,
                       linewidth=5, alpha=0.7, label='filtered', ax=ax, grid=True, legend=True)
```

Обратите внимание, что мы вызвали не метод `hist()`, а `plot()` с параметром **kind**, которому установили значение *kind='hist'*. Это та же гистограмма, только поддерживающая параметры, которые нельзя задействовать методом `hist()`:

- **histtype -** тип гистограммы, по умолчанию — это столбчатая (закрашенная). Значение *'step'* чертит только линию.
- **linewidth -** толщина линии графика в пикселях.
- **alpha -** густота закраски линии. 1 — это 100% закраска; 0 — прозрачная линия.
- **label -** Название линии.
- **ax**. Метод `plot()` возвращает оси, на которых был построен график. Чтобы обе гистограммы расположились на одном графике, сохраним оси первого графика в переменной `ax`, а затем передадим её значение параметру `ax` второго `plot()`. Так, сохранив оси одной гистограммы и построив вторую на осях первой, мы объединили два графика.
- **legend:** выводит легенду — список условных обозначений на графике.

При добавлении столбцов в датафрейм, нужно учесть несколько нюансов. Пусть у нас есть два датафрейма: `data1` и `data2`. Добавим в `data1` новую колонку, значения которой будут совпадать со значениями столбца `data2['column']`:

```
data1['new_column'] = df2['column']
```

Если бы столбец `new` уже был в `data1`, то все его элементы были бы удалены, а вместо них записаны новые.

Кажется просто: *Pandas* копирует столбец из `data2` и вставляет его в `data1`, однако всё сложнее. Для каждой строки первого датафрейма *Pandas* ищет «пару» — строку с таким же индексом во втором датафрейме. Находит и берёт значение из этой строки. В нашем случае индексы в `data1` и `data2` совпадали, и всё казалось простым копированием строк по порядку. Если же индексы не будут совпадать, например, в `data2` не будет некоторых значений индекса `data1` , то при копировании столбца на их месте будет значение *NaN*.

Число строк в `data2` не обязательно должно совпадать с числом строк `data1`. Если в `data2` не хватит значений, то будет *None*. А будут лишние — просто не попадут в обновлённый датафрейм. А вот повторяющиеся значения в индексе `data2` приведут к ошибке. *Pandas* не поймёт, какое из значений следует подставить в `data1`.

Отдельный столбец можно создать и без датафрейма, в *Series* — это будет набор значений с индексами. При попытке присвоить объект с индексами, *Pandas* подберёт соответствующие индексам строки. Если передавать столбцу список значений без индекса, такой как *list*, присвоение будет идти по порядку строк.

Объединяем данные из двух таблиц

При работе над задачей, нужно грамотно выбирать метод усреднения, так как он может повлиять на выводы. Где-то среднее арифметическое более точно опишет данные, а где-то может дать некорректный результат - тогда нужно вычислять медиану.

Метод `pivot_table` группирует данные, а что с ними делать, указывает значение параметра *aggfunc*. Среди таких значений есть *'first'* — первое значение из группы и *'last'* — последнее значение из группы.

В одном вызове `pivot_table` можно передать параметру *aggfunc* списком несколько функций — в результирующей таблице они будут в соседних столбцах.

Переименование столбцов

Когда в индексе не одно значение, а целый список, то получаются двухэтажные названия столбцов - **мультииндекс**. Вообще индексы можно представить как ещё один столбец в датафрейме. Выходит, мультииндекс — несколько столбцов. Это справедливо и для названий столбцов: они как дополнительные строки в датафрейме. В таком случае мультииндекс — это две и более строк с названиями.

Объединение столбцов методами merge() и join()

Когда нужно к существующему датафрейму добавить несколько новых столбцов, существует более лаконичный способ сделать это, по сравнению добавлением каждого столбца по очереди. Такая процедура называется объединение (join) или слияние (merge).

Для слияния используем метод `merge()`, параметр `how` - метод объединения:

```
data1.merge(data2, on='column', how='inner')
```

Если некоторые значения в колонке слияния совпадают, то остаются только одни, такой тип объединения называется `inner`, т.е. внутренняя общая область где есть и данные `data1` и `data2`. Существует тип слияния `outer`, т.е. внешняя общая область — область, где есть данные хотя бы в одном из df1 или df2; режим объединения `left`,когда в результате слияния обязательно будут все строки из левого DataFrame (в нашем случае `data1`); и есть полностью аналогичный режим `right`, когда сохраняются все строки из `data2`.

Если совпадут имена столбцов в `data1` и `data2`, то pandas переименует их, чтобы мы могли разобраться, где какое значение. К названию столбца из `data1` в таком случае припишется `_x`, а к столбцу из `data1` - `_y`. Вы можете самостоятельно задать, что приписать к названию столбцов, используя параметр `suffixes=('_x', '_y')` (заменив `'_x', '_y'` на нужные вам окончания имён столбцов).

Отметим так же, что если столбец индекса именованный, то можно передать его имя в параметр `on`. Объединять можно не только по одному столбцу, но и по совпадению значений в нескольких столбцах — достаточно только передать список в `on`.

Так же в pandas существует аналогичный метод `join()`, который ищет совпадения по индексам в `data1` и `data2`, если не указать параметр `on`. А в случае указания параметра `on` — он будет искать соответствующий столбец в `data1`, и сравнивать его с индексом `data2`. Кроме того, если в `merge()` по умолчанию `how='inner'`, то `join()` использует по умолчанию `how='left'`. Отличается и название параметра `suffixes` — они разделены на 2 независимых `lsuffix` и `rsuffix`. Ещё `join` позволяет объединить сразу более двух DataFrame — их набор можно передать списком вместо одного `data2`.

Конспект по теме «Взаимосвязь данных»

Диаграмма рассеяния

График, где значения соединяются линиями, хорош, если иллюстрирует непрерывную связь. Если же в нашем распоряжении данные, которые никак не связаны друг с другом, гораздо лучше обозначить их точками. Это возможно на особом типе графиков — **диаграмме рассеяния (scatter plot):**

```
data.plot(x='column_x', y='column_y', kind='scatter')
```

Корреляция

Очевидный недостаток диаграммы рассеяния в том, что местами может оказаться огромное количество точек, слившихся в единую массу. В облаке значений не разглядеть области более высокой плотности. Есть два способа сделать график нагляднее:

- Сделать точки полупрозрачными, задав параметр *alpha* и подобрать его оптимальное значение
- Построить график, поделённый на шестиугольные области

График делят на ячейки; пересчитывают точки в каждой ячейке. Затем ячейки заливают цветом: чем больше точек — тем цвет гуще. Такой график называется **hexbin - графиком, разделённым на шестиугольные области**. Число ячеек по горизонтальной оси задают параметром *gridsize*, аналогом *bins* для `hist()`.

```
data.plot(x='column_x', y='column_y', kind='hexbin', gridsize=our_gridsize, sharex=False, grid=True)
```

Смысл этого графика, как и у гистограммы — отображение частотности. Но на гистограмме показана только одна величина, а здесь две. Повышенная частота определённых сочетаний указывает на закономерность. Часто цель анализа данных в том и состоит, чтобы показать связь двух величин.

Взаимозависимость двух величин называется **корреляция**. График позволяет утверждать, что две величины явно взаимосвязаны, или **коррелируют.** В том случае, если существует прямая зависимость величин (чем больше одна, тем больше другая), то говорят, что корреляция **положительная.** В том случае, если существует обратная зависимость величин (чем больше одна, тем меньше другая), то говорят, что корреляция **отрицательная**.

Численно взаимосвязь оценивается с помощью **коэффициента корреляции Пирсона**. Он помогает определить, как сильно меняется одна величина при изменении другой; и

принимает значения от - 1 до 1. Если с ростом первой величины, растёт вторая, то коэффициент корреляции Пирсона — положительный. Если при изменении одной величины другая остаётся прежней, то коэффициент равен 0. Если рост одной величины связан с уменьшением другой, коэффициент отрицательный. Чем ближе коэффициент корреляции Пирсона к крайним значениям: 1 или -1, тем сильнее взаимозависимость. Если значение близко к нулю, значит связь слабая, либо отсутствует вовсе. Бывает, что коэффициент нулевой не оттого, что связи между значениями нет, а потому что у неё более сложный, не линейный характер. Потому-то коэффициент корреляции такую связь не берёт.

Коэффициент Пирсона находят методом `corr()`. Метод применяют к столбцу с первой величиной, а столбец со второй передают в параметре. Какая первая, а какая — неважно:

```
print(data['column_1'].corr(data['column_2']))
print(data['column_2'].corr(data['column_1']))
```

Совместное распределение для множества величин

Когда в задаче нужно найти попарные взаимосвязи величин, это можно сделать с помощью попарных диаграмм рассеяния. В *Pandas* такую задачу решают не `data.plot()`, а специальным методом: `pd.plotting.scatter_matrix(data)`.

```
pd.plotting.scatter_matrix(data)
```

Помимо попарных диаграмм рассеяния, можно получить попарный коэффициент корреляции для всех величин. Это можно сделать с помощью **матрицы корреляции**:

```
data.corr()
```

Конспект по теме «Валидация результатов»

Укрупняем группы

Если объект исследования является обособленным, никак не связанным с ближайшими соседями, то для анализа лучше подходит **столбчатый график**, так как он подчёркивает изолированность данных. Столбчатый график строят методом `plot()`, параметром передают тип графика: `kind='bar'`.

```
data.plot(y='column', kind='bar');
```

Однако, если объектов - много и не все они обладают достаточно высокими целевыми показателями, можно произвести их группировку для дальнейшей обработки. Группировку в таком случае можно произвести как выборочное переименование.

Выборочно изменяют значения методом **where()**. Ему передают 2 параметра: булев массив и новые значения. Если в булевом массиве True, соответствующее ему значение не изменится; а если False — значение поменяется на второй параметр метода.

```
data['column'].where(s > control_value, default_value)
```

В некоторых случаях, для визуализации данных уместно использовать круговую диаграмму, т.к. она хороша для показа доли каждого значения от 100% всех. Чтобы получить круговую диаграмму достаточно в `.plot` задать `kind='pie'`. При этом нужно задать столбец с данными параметром `y`:

```
data.plot(y='column', kind='pie');
```

Разбитые по группам данные

Работать с данными в зависимости от определённый значений в столбце по отдельности стандартными методами не всегда бывает удобно. Для того, чтобы работать с данными по группам, существует метод `groupby()`, который автоматически перебирает уникальные значения. Мы передадим ему столбец, а он вернёт последовательность пар: значение столбца — срез данных с этим значением. И в дальнейшем мы сможем обрабатывать эти срезы в соответствии с нашими нуждами.

```
for column_value, column_slice in data.groupby('column'):
  # do something
```

Конспект по теме «Описательная статистика»

Непрерывные и дискретные переменные

Категориальная (качественная) переменная принимает значение из ограниченного набора.

Количественная (численная) переменная принимает числовое значение в диапазоне. Количественные переменные подразделяются на:

- **Непрерывные**, которые могут принимать любое численное значение.
- **Дискретные**, которые могут принимать строго определённые значения.

Гистограмма частот для непрерывной переменной

Известная нам гистограмма хорошо подходит для работы с **дискретными** переменными. Для отображения частот непрерывных переменных нужно придумать что-то другое.

Одним из подходов к визуализации значений непрерывных переменных является разделение множества значений на интервалы и подсчёт количества значений, попадающих в каждый интервал.

В Pandas при построении гистограммы можно задавать не только количество интервалов - корзин, но и явно указывать их границы:

```
data.hist(bins=[value1, value2, value3, value4, ..., valueN])
```

Однако, этот подход не может дать полное представление о значениях переменной, так как полученная гистограмма сильно зависит от того, как мы разбили множество значений на интервалы

Гистограммы плотностей

Для того, чтобы решить недостаток разбиения на интервалы, применяется метод, отображающий частоту не высотой столбца в

гистограмме, а его площадью. Площадь столбца находят, как площадь прямоугольника: длину интервала умножают на высоту столбца. Найденная площадь — частота непрерывной переменной, а высота столбца — **плотность частоты**. Гистограмма, использующая в качества переменной - столбца плотность частоты, называется **плотностная гистограмма.**

Для того, чтобы оценить, сколько значений попало в любой интервал, не обязательно выбранный для построения, берут два значения и ищут площадь плотностной гистограммы между ними. Полученное число и будет оценкой количества значений, попавших в интервал.

Плотность частоты для непрерывных переменных можно задавать не только прямоугольниками, как на гистограммах, но и кривыми функциями. Работает тот же принцип: площадь между двумя значениями пропорциональна частоте попадания значений в интервал между ними.

Метрики локации данных

Такие характерные значения выборки, как медиана и среднее значение, также называют **метрики локации данных:** по медиане и среднему можно судить, *где* примерно расположен набор данных на числовой оси.

Для расчёта среднего значения берут *все значения датасета* — это наиболее полное использование информации при поиске метрики локации. Среднее значение называют **алгебраическая метрика локации.**

Медиана и квартили просто делят набор данных на части. Медиана — **структурная метрика локации.**

Кто разбросал данные?

Для представления о данных, недостаточно знать метрики локации, нужно ещё понимать, как данные разбросаны вокруг них. У структурной метрики локации есть структурные метрики разброса - квартили.

Для подсчёта разброса значений вокруг алгебраической метрики может применяться такой метод: вычисление среднего расстояние между средним значением и всеми остальными значениями переменной.

Дисперсия

Ранее предложенный метод подсчёта разброса значений вокруг алгебраической метрики, имеет право на существование, но он не всегда даёт полное представление о разбросе.

Улучшенная метрика разброса — не просто среднее расстояние между значениями датасета и средним, а средний квадрат этого расстояния. Эта величина называется **дисперсия** , её находят по формуле:

$$\sigma^2 = \frac{\sum (\mu - x_i)^2}{n}$$

где греческая буква мю μ обозначает среднее арифметическое значение совокупности данных:

$$\mu = \frac{\sum (x_i)}{n}$$

Библиотека **Numpy** в Python содержит большую библиотеку высокоуровневых математических функций. Импортируют её так:

```
import numpy as np
```

Дисперсию рассчитывают методом **var()**

```
import numpy as np
variance = np.var(x)
```

Стандартное отклонение

У дисперсии есть один небольшой недостаток: единица её измерения — это квадрат исходной величины. Чтобы вернуться к исходной единице измерения, из дисперсии извлекают квадратный корень. Получившаяся величина называется **стандартным отклонением:**

$$\sigma = \sqrt{\frac{\sum (\mu - x_i)^2}{n}}$$

Стандартное отклонение находят методом **std()** из библиотеки *Numpy*:

```
import numpy as np
standard_deviation = np.std(x)
```

Если дисперсия известна заранее, можно применить метод **sqrt()** из библиотеки *Numpy*. Корень из дисперсии будет равен стандартному отклонению:

```
import numpy as np
variance = 2.9166666666666665
standard_deviation = np.sqrt(variance)
```

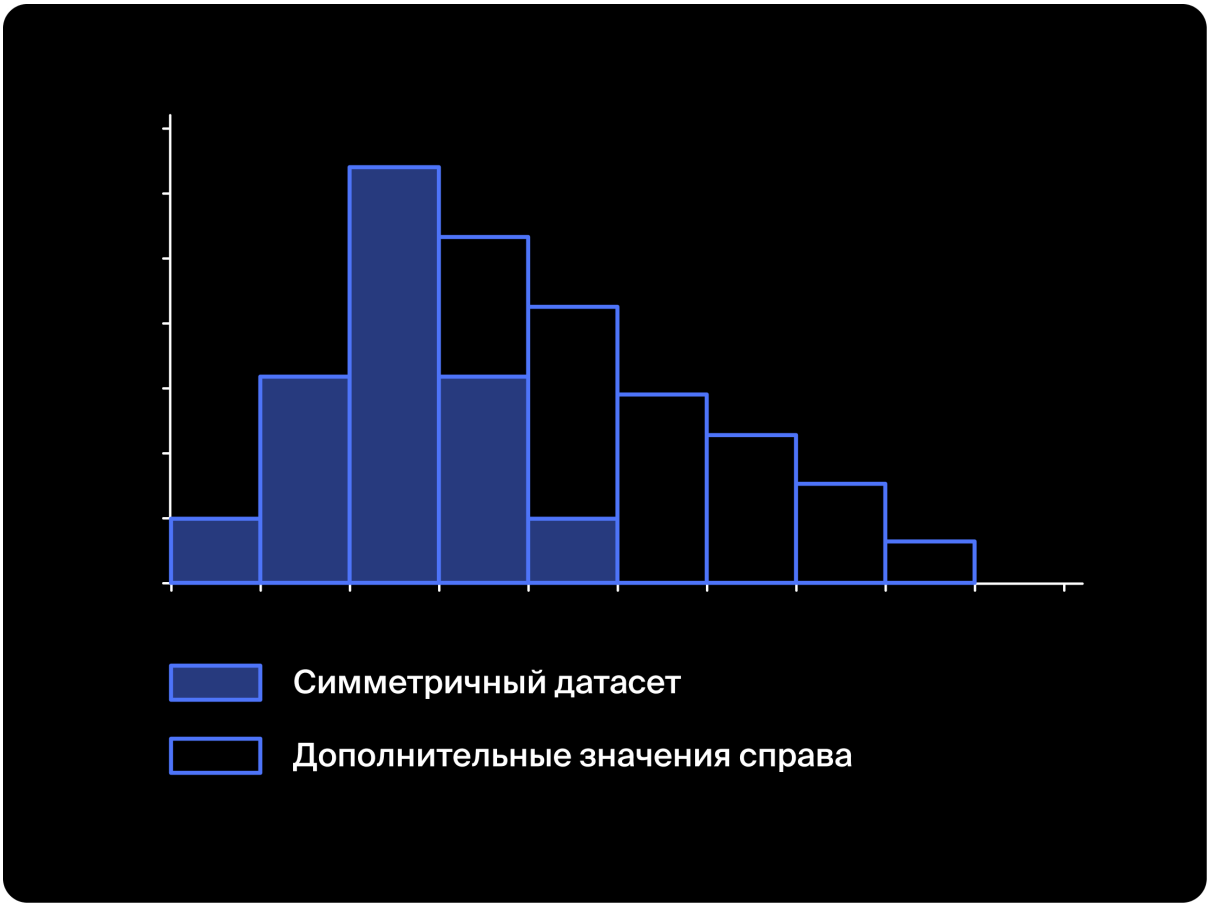
Для большинства распределений верно правило трёх стандартных отклонений, или **правило трёх сигм.** Оно гласит — практически все значения (около 99%) находятся в промежутке:

$$(\mu - 3\sigma, \mu + 3\sigma)$$

Это правило позволяет не только находить интервал, в который наверняка попадут практически все значения интересующей нас переменной, но и искать значения вне этого интервала — часто их называют **выбросами**.

Скошенность наборов данных

Многие данные «из жизни», распределены нормально, или симметрично. Однако датасеты могут быть ассиметричными, то есть, иметь **скошенность** в положительную или отрицательную сторону. Определить скошенность легко по гистограмме. Для этого нужно представить асимметричную гистограмму как симметричную с «дополнительными» значениями.



Такая гистограмма с *дополнительными значениями справа* отображает частоту значений в **скошенном вправо наборе данных.** Его также называют датасетом **с положительной скошенностью**, ведь дополнительные значения находятся со стороны положительного направления оси.

Скошенный влево датасет получится, если добавить к симметричному набору данных значений слева. Если влево идёт отрицательное направление оси, такой набор данных назовут датасетом **с отрицательной скошенностью**.

Скошенность данных также хорошо иллюстрирует диаграмма размаха.

Чтобы понять, в какую сторону скошен датасет, необязательно строить графики. Достаточно взглянуть на метрики локации: медиану и среднее. Помня о том, что медиана в отличие от среднего устойчива к выбросам, легко сделать вывод, что для скошенных вправо данных медиана будет меньше среднего, а для скошенных влево — больше.

Конспект по теме «Теория вероятностей»

Что такое вероятность: эксперименты, элементарные исходы и события

Эксперимент — это повторяемый опыт, который может окончиться разными исходами, или, как принято говорить, **элементарными исходами**: исход либо случился, либо не случился.

В простейшем случае эти исходы не отличаются: мы не можем предпочесть один из них другому, — и значит вероятность каждого из них одинакова. Такие исходы называются **равновероятными**. В честном эксперименте (эксперимент с равновероятными исходами) с **n** элементарными исходами вероятность каждого исхода одинакова и равна 1/**n**.

Множество всех элементарных исходов эксперимента принято называть **вероятностным пространством**. Из него можно выделить подмножества, содержащие в себе некоторое количество элементарных исходов - **события**.

Невозможное событие - событие, которое не произойдёт никогда, вероятность его появления равна 0. **Достоверное событие** - событие, которое точно произойдёт, вероятность его которого равна 1. Вероятность появления других событий находится в промежутке от 0 до 1.

При сохранении условия равновероятности всех элементарных исходов **вероятность события** — количество исходов, входящих в это событие, делённое на общее количество исходов, то есть на размер вероятностного пространства.

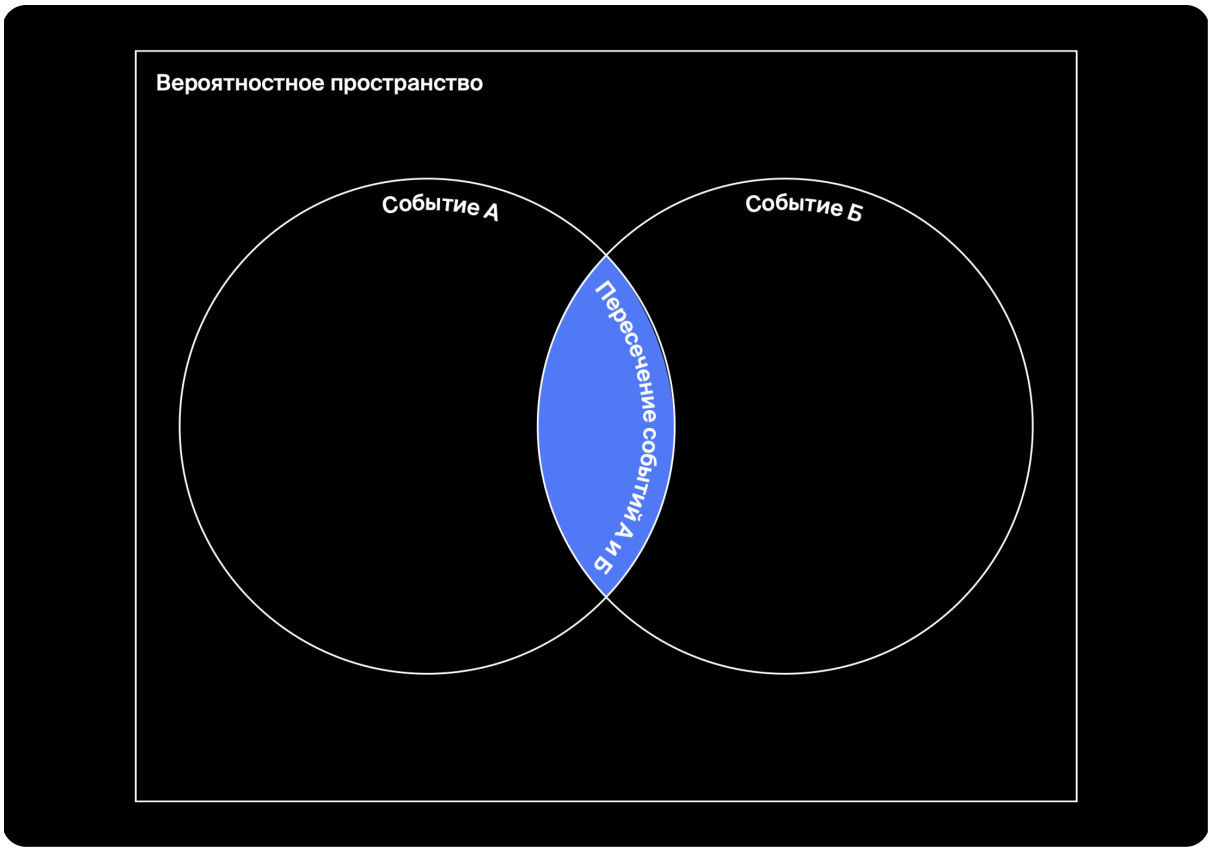
Закон больших чисел

Закон больших чисел: чем больше раз повторяется эксперимент, тем ближе частота заданного на этом эксперименте события будет к его вероятности.

Это правило можно использовать и в обратную сторону: если мы не знаем вероятность какого-то события, но можем много раз повторить эксперимент, по частоте выпадания исходов, входящих в это событие, можно судить о его вероятности.

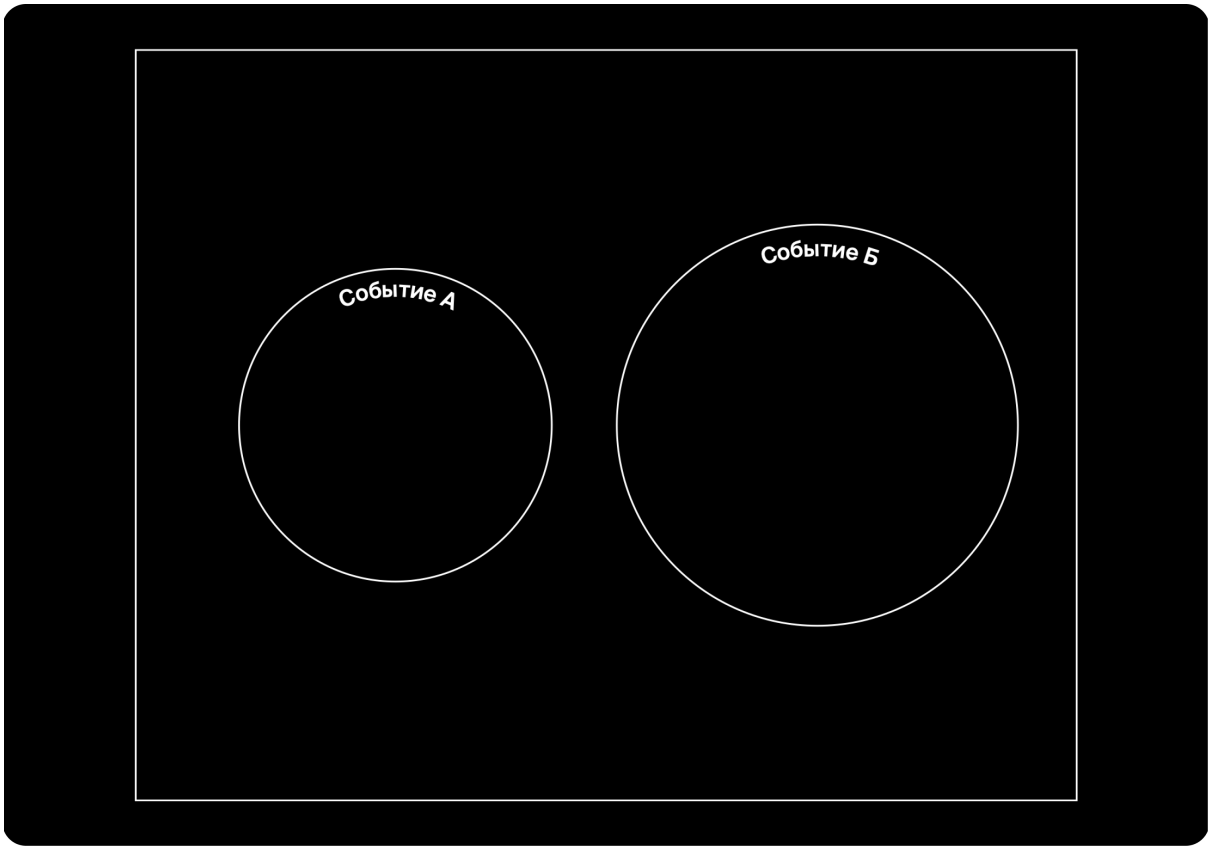
Взаимоисключающие и независимые события, умножение вероятностей

Для отображения пересечения между событиями, используется **диаграмма Эйлера-Венна**:



События А и Б пересекаются, значит существуют элементарные исходы, входящие и в А, и в Б.

Взаимоисключающими называют события, которые не могут произойти одновременно при проведении эксперимента — на диаграмме Эйлера-Венна они не пересекаются:



Вероятность взаимоисключающих событий равна нулю.

События называются **независимыми**, если наступление одного из них не влияет на вероятность другого. Если события независимы, то вероятность их пересечения равна произведению их вероятностей. Это правило работает и в обратную сторону.

Если взаимоисключающие события охватывают всё вероятностное пространство, сумма их вероятностей равна единице.

Взаимоисключаемость событий видна на диаграмме Эйлера-Венна. А вот независимость так просто не обнаружишь, нужно проверять условие равенства произведения вероятностей событий вероятности их пересечения.

Случайные величины, распределение вероятностей и интервалы значений

Случайная величина — это переменная, которая принимает **случайные значения** - те значения, которые нельзя предсказать до проведения эксперимента. У эксперимента есть исходы, которые могут описываться как количественно, так и качественно. Случайная же величина определяется на этих исходах *численно*. Это способ спроецировать исходы эксперимента, как бы они ни определялись, на числовую ось.

Как и все количественные переменные, случайная величина может быть **дискретной** или **непрерывной**.

Распределением вероятности случайной величины называется таблица, содержащая всевозможные значения случайной величины и вероятности их появления.

Для хранения числовых таблиц, используется тип данных **numpy array** из библиотеки Numpy:

```
table = np.array([[2,3,4,5,6,7],
[3,4,5,6,7,8],
...
[7,8,9,10,11,12]])
```

Если при работе со словарём вам необходимо получить список всех ключей словаря, то это можно сделать с помощью метода `keys()`. А список всех значений словаря — с помощью метода `values()`:

```
dictionary = {...}
print(dictionary.keys())
print(dictionary.values())
```

Математическое ожидание и дисперсия

Для эксперимента можно задать случайную величину и найти численное значение, к которому она будет в среднем стремиться при многократном повторе эксперимента. Это значение называется **математическим ожиданием** случайной величины.

Если эксперимент состоит из *равновероятных элементарных исходов*, заданных численно, математическое ожидание будет равно *среднему* возможных значений.

Математическое ожидание случайной величины — сумма всех значений случайной величины, помноженных на их вероятности:

$$E(X) = \sum p(x_i)x_i$$

Математическое ожидание — аналог метрики локации, только не для датасета, а для случайной величины. Оно показывает, вокруг какого значения распределена случайная величина, и — по закону больших чисел — к какому значению она будет в среднем стремиться при повторе эксперимента.

Поскольку случайная величина распределена вокруг этой «метрики локации», можно найти и меру её разброса. Для этого нужно найти математическое

ожидание квадрата случайной величины — это несложно если учесть, что значения меняются, а их вероятности — нет.

Если мы знаем математическое ожидание самой случайной величины и её квадрата, **дисперсию** находят по формуле:

$$Var(X) = E(X^2) - (E(X))^2$$

Вероятность успеха в биномиальном эксперименте

Эксперименты с двумя возможными исходами называются **биномиальными экспериментами**. Обычно один из результатов называют успехом, а второй, соответственно, неудачей. Если вероятность успеха равна `p`, то вероятность неудачи `(1 - p)`.

Биномиальное распределение

Количество способов выбрать **k** успехов из **n** повторений эксперимента находят по формуле:

$$C_n^k = \frac{n!}{k!(n-k)!}$$

где <число>! (читается как <число> факториал) равно произведению всех натуральных чисел от 1 до этого числа: n! = 1 · 2 · 3 · 4 · ... · (n-1) · n.

Для вычисления факториала используется библиотека *math* и её метод *factorial*:

```
from math import factorial
x = factorial(5)
```

Рассмотрим задачу о биномиальном эксперименте (повторении эксперимента с двумя исходами *n* раз) в общем виде. Если вероятность успеха `p` и неуспеха `1 - p`, а эксперимент был повторен *n* раз, то вероятность любого количества успехов *k* из этих *n* экспериментов:

$$P(k \text{ успехов из } n \text{ попыток}) = C_n^k p^k (1-p)^{n-k}$$

Условия, при которых можно утверждать что случайная величина распределена биномиально:

- проводится конечное фиксированное число попыток *n*;
- каждая попытка — простой биномиальный эксперимент ровно с двумя исходами;
- попытки независимы между собой;

- вероятность успеха *p* одинаковая для всех *n* попыток.

Нормальное распределение

Ключевая теорема в статистике — **центральная предельная теорема**. Если немного упростить, она гласит: «Много независимых случайных величин, сложенных вместе, дают нормальное распределение».

Нормальное распределение описывает множество реальных непрерывных величин. Нормальное распределение определяют два параметра — среднее и дисперсия:

$$X \sim \mathbb{N}(\mu, \sigma^2)$$

Эта запись читается так: переменная *X* распределена нормально со средним мю (μ) и дисперсией сигма в квадрате (σ^2), то есть стандартным отклонением сигма.

Для того, чтобы по известным параметрам распределения найти вероятность попадания в те или иные интервалы, вызовем два метода из пакета *scipy.stats:* **norm.ppf** и **norm.cdf**:

- ppf — функция процентных значений;
- cdf — кумулятивная функция распределения.

Обе работают с нормальным распределением, заданным своими средним и стандартным отклонением.

- Функция *norm.ppf* выдаёт *значение переменной* для известной *вероятности* интервала слева от этого значения.
- Функция *norm.cdf,* наоборот, выдаёт для известного значения *вероятность* интервала слева от этого значения.

Чтобы задать нормальное распределение, используется метод *norm()* из пакета *scipy.stats* с двумя аргументами: математическим ожиданием и стандартным отклонением. Найдём вероятность получить некоторое значение **x**:

```
from scipy import stats as st

# задаем нормальное распределение
distr = st.norm(1000, 100)

x = 1000

result = distr.cdf(x) # считаем вероятность получить значение x
```

С помощью функции **norm.cdf** можно посчитать вероятность получить значение в промежутке от **x1** до **x2:**

```
from scipy import stats as st

# задаем нормальное распределение
distr = st.norm(1000, 100)

x1 = 900
x2 = 1100

result = distr.cdf(x2) - distr.cdf(x1) # считаем вероятность получить значение между x1 и x2
```

Для того, чтобы по вероятность получить значение, воспользуемся методом **norm.ppf**:

```
from scipy import stats as st

# задаем нормальное распределение
distr = st.norm(1000, 100)

p1 = 0.841344746

result = distr.ppf(p1)
```

Нормальная аппроксимация биномиального распределения

При большом количестве повторений биномиального эксперимента биномиальное распределение приближается к нормальному.

Для дискретного биномиального распределения, заданного числом повторов эксперимента `n` и вероятностью успеха `p`, математическое ожидание равно `n·p`, а дисперсия: `n·p·(1-p)`.

Если *n* больше 50, эти параметры биномиального распределения можно взять как среднее и дисперсию для нормального распределения, которое будет достаточно близко описывать биномиальное. Максимально близкое к биномиальному нормальное распределение задаётся его математическим ожиданием `n·p` в качестве среднего значения и дисперсией `n·p·(1-p)`.

Конспект по теме "Проверка гипотез"

Случайная выборка и выборочное среднее

Логика проведения статистической проверки гипотез немного другая, по сравнению с механизмами в теории вероятностей. Прежде всего, мы будем судить о большом объёме данных, **генеральной совокупности**, по части — **выборке**.

Для анализа необязательно загружать все данные, достаточно взять небольшую, но **репрезентативную**, представляющую всю генеральную совокупность, часть данных. Самый простой способ добиться репрезентативности — взять **случайную выборку**. Из всего датасета генератором случайных чисел отбирают случайные элементы. По ним будут судить обо всей генеральной совокупности.

Она может состоять из нескольких неравных по размеру частей, сильно отличающихся по исследуемому параметру. Тогда есть смысл взять пропорциональные случайные выборки из этих частей, и потом соединить между собой. Получается **стратифицированная выборка**, более репрезентативная, чем просто случайная. Она так называется, потому что мы разбили генеральную совокупность на **страты** — группы, объединённые общим признаком. Случайные выборки получают уже из них.

По выборке судят о генеральной совокупности — точнее об её статистических параметрах. Обычно достаточно оценить среднее и дисперсию, чтобы сделать выводы о **равенстве или неравенстве средних значений** исследуемых совокупностей. Нас будет интересовать именно такая постановка задачи.

Что можно сказать о среднем и дисперсии генеральной совокупности по среднему и дисперсии, посчитанным на выборке, или **выборочному среднему** и **выборочной дисперсии**? Почти всё, при условии, что выборка достаточно велика.

Одна из формулировок центральной предельной теоремы звучит так: если в выборке достаточно наблюдений, **выборочное распределение** выборочного среднего из любой генеральной совокупности распределено

нормально вокруг среднего этой генеральной совокупности. *«Любая генеральная совокупность»* означает, что сама генеральная совокупность может быть распределена как угодно. Датасет из средних значений выборок всё равно будет нормально распределён вокруг среднего всей генеральной совокупности.

Стандартное отклонение выборочного среднего от настоящего среднего генеральной совокупности называется **стандартной ошибкой** и находится по формуле:

$$E.S.E. = \frac{S}{\sqrt{n}}$$

E.S.E. — оценённая стандартная ошибка. «Оценённая» — имея только выборку, мы не знаем точную ошибку и *оцениваем* её исходя из имеющихся данных.

S — оценка стандартного отклонения генеральной совокупности.

n — размер выборки. Раз корень из *n* стоит в знаменателе, стандартная ошибка уменьшается с увеличением размера выборки.

Формулирование гипотез

Никакие экспериментально полученные данные никогда не *подтвердят* какую-либо гипотезу. Это наше фундаментальное ограничение. Данные могут лишь не противоречить ей или, наоборот, показывать крайне маловероятные результаты (при условии, что гипотеза верна). Но и в том, и в другом случае нет оснований утверждать, что выдвинутая гипотеза *доказана*.

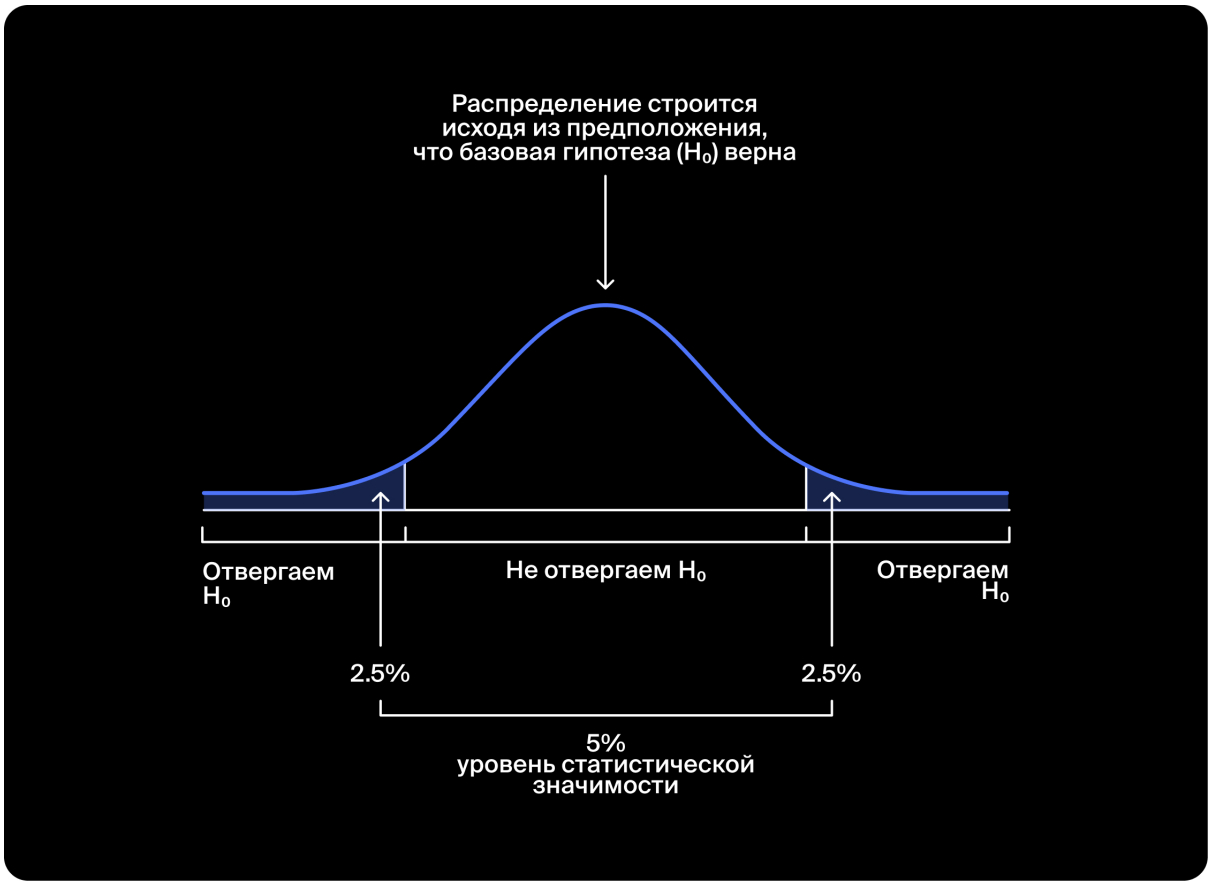
Допустим, данные гипотезе не противоречат, тогда мы её *не отвергаем*. Если же мы приходим к выводу, что получить такие данные в рамках этой гипотезы вряд ли возможно, у нас появляется основание отбросить эту гипотезу.

Типичные статистические гипотезы касаются средних значений генеральных совокупностей и звучат так:

- среднее генеральной совокупности равно конкретному значению;
- средние двух генеральных совокупностей равны между собой.

Алгоритм проверки статистических гипотез всегда начинается с формулирования гипотез. Сначала формулируется **нулевая гипотеза H_0**. Например, «среднее рассматриваемой генеральной совокупности равно A», где *A* — некоторое число. Исходя из H_0 формулируется **альтернативная гипотеза H_1**. Для этой H_0 она звучит как «среднее генеральной совокупности не равно A». H_0 всегда формулируется так, чтобы использовать знак равенства.

Построим распределение на предположении, что гипотеза H_0 верна. В нашем случае это будет нормальное распределение вокруг интересующего нас параметра — среднего. Дисперсия, или стандартное отклонение, оценивается по данным выборки.



Для нормального распределения вероятность попасть в тот или иной интервал равна площади графика над этим интервалом. В районе среднего значения и в некотором диапазоне вокруг него будут значения, которые весьма вероятно получить случайно.

Как определить, где мы ещё не отвергаем нулевую гипотезу, а где пора? Критические значения задаются выбранным уровнем значимости

проверки гипотезы. **Уровень значимости** — это суммарная вероятность того, что измеренное эмпирически значение окажется далеко от среднего. Если наблюдаемое на выборке значение попадает в эту зону, вероятность такого события при верной нулевой гипотезе признаётся слишком малым, значит, у нас есть основание отвергнуть нулевую гипотезу. Когда значение попадает в зону «Не отвергаем H_0», то оснований отвергать нулевую гипотезу нет. Считаем, что эмпирически полученные данные не противоречат нулевой гипотезе.

В Python существует метод, который просто возвращает **статистику разности** между средним и тем значением, с которым вы его сравниваете. Главное — уровень значимости, на котором они находятся друг от друга — **p-value**.

Статистика разности — это количество стандартных отклонений между сравниваемыми значениями, если оба распределения привести к стандартному нормальному распределению со средним 0 и стандартным отклонением 1. По этой цифре сложно сориентироваться.

Есть смысл принимать решение о принятии или отвержении нулевой гипотезы по ***p-value***. Это вероятность получить наблюдаемый результат при условии, что нулевая гипотеза верна. Общепринятые пороговые значения — 5% и 1%. Окончательное решение, какой порог считать достаточным, всегда остаётся за аналитиком.

Для проверки гипотезы о равенстве среднего генеральной совокупности некоторому значению, можно использовать метод `scipy.stats.ttest_1samp()`. Параметры метода: `array` — массив, содержащий выборку; `popmean` — предполагаемое среднее, на равенство которому мы делаем тест. После выполнения метод вернёт статистику разности между popmean и выборочным средним из array, а также уровень значимости:

```
from scipy import stats as st

interested_value = 120

results = st.ttest_1samp(
    array,
    interested_value)

print('p-значение: ', results.pvalue)
```

Гипотеза о равенстве средних двух генеральных совокупностей

Когда генеральных совокупностей две, бывает нужно сопоставить их средние. Чтобы проверить гипотезу о равенстве среднего двух генеральных совокупностей по взятым из них выборкам, примените метод `scipy.stats.ttest_ind()`. Методу передают параметры: `array1`, `array2` — массивы, содержащие выборки; `equal_var` — **необязательный** параметр, задающий считать ли равными дисперсии выборок. Если есть основание полагать, что выборки взяты из схожих по параметрам совокупностей, тогда укажите `equal_var = True`, и дисперсия каждой выборки будет оценена по объединенному датасету из двух выборок, а не для каждой по отдельности по значениям в ней самой. Это позволяет получить более точные результаты, но только в том случае, если считать примерно равными дисперсии генеральных совокупностей, из которых взяты выборки. Иначе нужно указать `equal_var = False`; по умолчанию он задан как `equal_var = True` (если вообще его не указывать).

```
from scipy import stats as st

sample_1 = [...]
sample_2 = [...]

results = st.ttest_ind(
    sample_1,
    sample_2)

print('p-значение: ', results.pvalue)
```

Гипотеза о равенстве средних для зависимых (парных) выборок

Когда генеральная совокупность одна, полезно понять, равно ли себе среднее этой совокупности до и после изменения. **Парная выборка** означает, что мы измеряем некоторую переменную для одних и тех же единиц. Чтобы проверить гипотезу о равенстве средних двух

генеральных совокупностей для зависимых (парных) выборок в Python, применим функцию `scipy.stats.ttest_rel()`:

```
from scipy import stats as st

before = [...]
after = [...]

results = st.ttest_rel(
    before,
    after)

print('p-значение: ', results.pvalue)
```

Конспект по теме "Извлечение данных из веб-ресурсов"

Что такое Web Mining

Когда предоставленных компанией данных мало для решения задачи, аналитик добывает информацию самостоятельно. Такое обогащение данных позволяет не только учесть больше факторов, но и выявить новые закономерности, прийти к неожиданным выводам. Аналитики обогащают имеющуюся информацию в интернете. Сперва находят ценные для исследования веб-ресурсы, а затем извлекают из них нужные данные. Этот процесс называют **Web Mining**, или **парсинг**.

Что аналитик должен знать об интернете? Браузер. HTML. HTTP.

Интернет — сеть компьютеров, которые обмениваются данными. В интернете действуют концепции, которые утверждают правила *представления информации в интернете (учитывая как компьютер их отображает) и обмена информации в интернете*. Для этого был придуман язык, на котором можно создавать документы в интернете — **HTML**, написана программа для просмотра этих документов — браузер и сформулированы единые правила, по которым документы передаются — транспортный протокол **HTTP**.

Что такое транспортный протокол

Интернет — сеть из компьютеров, которые обмениваются информацией. Чтобы это было возможным, нужны правила: в каком виде один компьютер высылает данные другому. Обмен данными в интернете построен на принципе «запрос — ответ»: браузер формирует запрос, сервер его анализирует и отправляет ответ. Правила, по которым нужно

формулировать запросы и ответы, определяет транспортный протокол — HTTP.

Сегодня большинство сайтов применяют более совершенный протокол передачи информации — **HTTPS**. Это защищённая версия HTTP-протокола. Он гарантирует, что все коммуникации между вашим браузером и веб-сайтом зашифрованы.

Когда вы заходите на сайт, браузер отправляет HTTP-запрос на сервер, а тот в свою очередь формирует ответ: HTML-код нужной страницы. Запрос, который формирует браузер, может включать в себя:

- *HTTP*-метод: он определяет операцию, которую нужно совершить. Есть несколько методов, самые популярные из них: GET — запрос данных с сервера — и POST — отправление данных на сервер.
- Путь до ресурса: это часть адреса без имени сайта.
- Версию HTTP-протокола, который используется для отправки запроса
- Заголовки запроса, в них можно передать серверу дополнительную информацию.
- Тело запроса есть не у всех запросов.

Ответ может включать:

- Версию HTTP-протокола.
- Код и сообщение ответа.
- Заголовки, содержащие дополнительную информацию для браузера.
- Тело ответа.

Введение в HTML

Чтобы получить необходимую информацию из веб-страниц, нужно заполучить код страницы и контент внутри него. Для этого нужно проанализировать HTML-код.

В HTML каждый объект страницы размечается для корректного представления на сайте. Разметка состоит в том, что блоки информации заключают в управляющие конструкции — *теги*. Такие «бирки» указывают браузеру, как отобразить то, что в них «обёрнуто».

HTML-элемент состоит из *тегов* и размещённого между ними содержания — *контента*. У любого тега **HTML** есть имя и угловые скобки. В начале HTML-элемента ставят *открывающий тег* с именем тега, а в конце — *закрывающий тег*, где имя будто перечеркнуто косой чертой. Созданный элемент называют по имени тега.

Типовая структура страницы HTML выглядит так:

1. <html> ... </html>

Тег **<html>** открывает каждый HTML-документ и определяет его начало, а **</html>** означает его конец. Внутри этого тега хранится заголовок **<head>** и тело **<body>** HTML-документа.

2. <head> ... </head>

Эта пара тегов обозначает заголовок документа. Внутри заголовка помещаются теги для названия документа **<title>** и дополнительной (мета) информации **<meta>**.

3. <body> ... </body>

Тег **<body>** показывает, где начинается тело HTML-страницы. Внутри тела помещают всё наполнение HTML-страницы: заголовки, абзацы текста, таблицы, изображения.

Чтобы в разметке легче было разобраться, в коде веб-страницы разработчики оставляют комментарии внутри специальных тегов `<!-- -->`.

Таблицы в HTML обычно помещаются в элемент-контейнер *table* между тегами `<table>` и `</table>`. Внутри контейнера содержимое таблицы делится на строки тегами `<tr>`, а строки — на ячейки тегами `<td>`. Верхняя, первая строка вместо ячеек обычно содержит заголовки столбцов в тегах `<th>`.

Текст часто помещают в элемент *p*. Абзац текста располагается между открывающим `<p>` и закрывающим `</p>`.

Распространён тег блоков `<div>`. Это обёртка для других элементов. Контейнер *div* удобен, что может включить в себя любое число разнородных элементов и определить им общие свойства или поведение.

Внутри тегов можно указывать **атрибуты**. Они служат для передачи дополнительных сведений в элемент. Для разных сведений — разные

атрибуты. Имя атрибута говорит браузеру, какой признак он определяет, а значение — каким этот признак должен стать.

Чаще всего вам будут нужны атрибуты `id` и `class`. Атрибут *id* — идентификатор с уникальным именем. Значение атрибута *class* — имя, которое могут носить несколько элементов, как разные члены семьи носят общую фамилию.

Инструменты разработчика

В каждом современном браузере есть «швейцарский нож» программиста — **панель инструментов разработчика**. Здесь можно посмотреть код всей страницы или конкретного элемента, изучить стили каждого элемента страницы и даже изменить их отображение на своём компьютере. В браузерах панель инструментов разработчика вызывают комбинацией `Ctrl + Shift + I`.

Ваш первый get-запрос

Чтобы получить данные с сервера, вам понадобится метод **get()**. Для отправления HTTP-запросов подключают библиотеку **Requests**:

```
import requests
```

Метод *get()* библиотеки *Requests* выступает в роли браузера. Он принимает ссылку на сайт в качестве аргумента. Метод отправит get-запрос на сервер, обработает полученный оттуда результат и вернёт объект **response**. *Response* — специальный объект, содержащий ответ сервера на HTTP-запрос:

```
req = requests.get(URL) # сохраняем объект Response в переменную req
```

Объект *Response* содержит ответ сервера: код состояния, содержание запроса и код самой HTML-страницы. Атрибуты объекта *Response* позволяют возвращать не все данные с сервера, а только нужные для

анализа. Например, объект *Response* с атрибутом *text* «отдаст» лишь текстовое содержание запроса:

```
print(req.text) # название атрибута пишут после объекта Response, разделяя точкой
```

Атрибут *status_code* отвечает на вопрос: отправил сервер ответ или возникла какая-то ошибка:

```
print(req.status_code)
```

К сожалению, не все запросы возвращаются с данными. Иногда результатом запроса бывает ошибка: в зависимости от типа её обозначают специальным кодом. Вот коды ошибок, которые чаще всего возникают:

Коды ошибок

Код ошибки	Название	Что означает?
200	OK	Всё отлично
302	Found	Расположение ресурса изменилось
400	Bad Request	Синтаксическая ошибка в запросе
404	Not Found	Ресурс не найден
500	Internal Server Error	Внутренняя ошибка сервера
502	Bad Gateway	Ошибка при обмене данных между серверами
503	Server Unvailable	Сервер временно не может обрабатывать запросы

Регулярные выражения

Для поиска строк с больших текстах используется мощный инструмент — регулярные выражения. **Регулярное выражение** — правило для поиска подстрок (фрагментов текста внутри строк). Регулярные выражения позволяют создавать сложные правила, так что одно выражение вернёт несколько подстрок.

Для работы с регулярными выражениями в Python импортируют библиотеку **re**. Дальше поиск ведётся в два этапа. Сначала создают шаблон регулярного выражения. Это алгоритм, по которому нужно искать строку в тексте. Затем готовый шаблон передают специальным методам библиотеки *re*, которые ищут, заменяют и удаляют нужные символы. Таким образом, шаблон определяет, что и как искать, а метод — что с этим потом делать.

В таблице приведены простейшие шаблоны регулярных выражений. Сложные регулярные выражения состоят из их комбинаций.

Синтаксис регулярных выражений

Регулярное выражение	Описание	Пример	Пояснение
[]	Один из символов в скобках	[a-]	a или -
[^...]	Отрицание	[^a]	любой символ кроме «a»
-	Интервал	[0-9]	интервал: любая цифра от 0 до 9
.	Один любой символ, кроме перевода строки	a.	as, a1, a_
\d (аналог [0-9])	Любая цифра	a\d | a[0-9]	a1, a2, a3
\w	Любая буква, цифра или _	a\w	a_, a1, ab
[A-z]	Любая латинская буква	a[A-z]	ab
[А-я]	Любая буква кириллицы	a[А-я]	ая
?	Ноль или одно вхождение	a?	a или ничего
+	Одно и более вхождений	a+	a или aa, или aaa
*	Ноль и более вхождений	a*	ничего или a, или aa
^	Начало строки	^a	a1234, abcd
$	Конец строки	a$	1a, ba

Самые распространённые задачи аналитика:

- найти подстроку в строке
- разбить строки на подстроки на основании шаблона

- заменить части строки на другую строку

Вот какие методы библиотеки *re* для этого понадобятся:

1. **search(pattern, string)** ищет шаблон *pattern* в строке *string*. Хотя *search()* ищет шаблон во всей строке, возвращает он только первую найденную подстроку:

```
import re
print(re.search(pattern, string))
```

Метод *search()* возвращает объект типа **match**. Параметр *span* указывает диапазон индексов, подходящих под шаблон. В параметре *match* указано само значение подстроки.

Если нам не нужны дополнительные сведения о диапазоне, выведем только найденную подстроку методом *group()*:

```
import re
print(re.search(pattern, string).group())
```

2. **split(pattern, string)** разделяет строку *string* по границе шаблона *pattern*.

```
import re
print(re.split(pattern, string))
```

Строка разделена на несколько частей. Границы деления строки проходят там, где метод встретил указанный в аргументе шаблон. Количеством делений строки можно управлять. За это отвечает параметр **maxsplit** метода *split()* (по умолчанию равен 0).

```
import re
print(re.split(pattern, string, maxsplit = num_split))
```

3. **sub(pattern, repl, string)** ищет подстроку по шаблону *pattern* в строке *string* и заменяет её на подстроку **repl**.

```
import re
print(re.sub(pattern, repl, string))
```

4. Метод **findall(pattern, string)** возвращает список *всех подстрок* в *string*, удовлетворяющих шаблону *pattern*. А не только первую подходящую подстроку, как *search()*.

```
import re
print(re.findall(pattern, string))
```

Метод **findall()** удобен тем, что можно сразу посчитать количество повторяющихся подстрок в строке функцией *len()*:

```
import re
print(len(re.findall(pattern, string)))
```

Парсинг HTML

Достать данные из строки, которая содержит код страницы, вручную сложно. Чтобы решить проблему, обратимся к возможностям библиотеки **BeautifulSoup.** Методы библиотеки *BeautifulSoup* превращают HTML-файл в древовидную структуру. После этого нужный контент можно отыскать по тегам и атрибутам.

```
from bs4 import BeautifulSoup
soup = BeautifulSoup(req.text, 'lxml')
```

Первый аргумент — это данные, из которых будет собираться древовидная структура. Второй аргумент — синтаксический анализатор, или парсер. Он отвечает за то, как именно из кода веб-страницы получается «дерево». Парсеров много, они создают разные структуры из одного и того же HTML-документа. За высокую скорость работы мы выбрали анализатор **lxml**. Есть и другие, например, *html.parser*, *xml* или *html5lib*.

После превращения кода страницы в дерево, можно искать данные различными методами. Первый метод поиска называется **find()**. В HTML-документе он находит первый элемент, имя которого ему передали в качестве аргумента, и возвращает его весь, с тегами и контентом.

```
tag_content = soup.find(tag)
```

Чтобы посмотреть контент без тега, вызывают метод **text**. Результат возвращается в виде строки:

```
tag_content.text
```

Существует и другой метод поиска — **find_all**. Этот метож находит *все* вхождения определённого элемента в HTML-документе и возвращает список:

```
tag_content = soup.find_all(tag)
```

Методом *text* вычленим только контент из тегов:

```
for tag_content in soup.find_all(tag):
    print(tag_content.text)
```

У методов *find()* и *find_all()* есть дополнительный фильтр поиска элементов страницы — параметр **attrs**. Он используется для поиска по идентификаторам и классам. Их имена уточняют в панели разработчика.

Параметру *attrs* передают словарь с именами и значениями атрибутов:

```
soup.find(tag, attrs={"attr_name": "attr_value"})
```