

## Рассмотрим темы:

- ✓ Парсинг сайтов
- ✓ Запись данных парсинга в файл
- ✓ Статистическая обработка данных
- ✓ Построение графиков

# Парсинг — это автоматизированный сбор общедоступной информации из интернета.

*Можно представить, как человек открывает браузер, ходит по сайтам и копирует с них данные. Парсинг — то же самое, только ходит не человек, а робот. Так делают поисковики, агрегаторы, скоринговые компании, злоумышленники, продающие персональные данные, и много кто ещё.*

## Парсинг подходит для быстрого сбора большого объема данных.

*Парсинг — не совсем точное название. На самом деле автоматизированный сбор информации лучше называть скрейпингом, а парсинг — это этап скрейпинга, на котором из скачанных данных извлекается нужная информация. Но в русскоязычных источниках чаще попадаете слово «парсинг»*

# Для каких целей можно использовать парсинг

- ✓ **Исследование рынка.** Парсинг позволяет быстро оценить, какие товары и цены у конкурентов.
- ✓ **Анализ динамики изменений.** Парсинг можно проводить регулярно, чтобы оценивать, как менялись какие-то показатели. Например, росли или падали цены, изменялось количество онлайн-объявлений или сообщений на форуме.
- ✓ **Устранение недочетов на собственном ресурсе.** Выявление ошибок в мета-тегах, битых ссылок, проблем с редиректами, дублирующихся элементов и т. д.
- ✓ **Поиск внешних ссылок, ведущих на вашу площадку.** Это поможет оценить работу подрядчика по линкбилдингу. Как проверять внешние ссылки и какими инструментами это делать, подробно описано в этой [статье](#) .
- ✓ **Наполнение каталога интернет-магазина.** Чтобы упростить этот процесс, часто парсят зарубежные магазины и просто переводят информацию о товарах.
- ✓ **Составление клиентской базы.** В этом случае парсят контактные данные, например, пользователей соцсетей, участников форумов и т. д. Но тут стоит помнить, что сбор информации, которой нет в открытом доступе, незаконен.
- ✓ **Сбор отзывов и комментариев на форумах, в соцсетях.**
- ✓ **Создание контента, который строится на выборке данных.** Например, результаты спортивных состязаний, инфографики по изменению цен, погоды и т. д.

# Законность парсинга

Сам по себе сбор данных из открытых источников законом не запрещен. НО искать, и использовать информацию нужно с соблюдением законодательства — и тут в силу вступают другие правовые нормы:

- ✓ Если при помощи парсеров вы полностью копируете информацию с сайтов конкурентов на собственный ресурс, это может привести к нарушению интеллектуального права.
- ✓ Чрезмерно агрессивный парсер может создать большую нагрузку на целевой сайт, которая будет выглядеть как DDOS-атака. Если вы парсите такой программой интернет-магазин, то он может стать недоступным на несколько часов, и владельцы сайта потерпят убытки. Или у владельцев могут возрасти затраты на обслуживание серверов.
- ✓ В Уголовном кодексе предусмотрена ответственность за «неправомерный доступ к охраняемой законом информации». Эта формулировка включает в себя персональные данные или коммерческую тайну.
- ✓ Согласно Закону о персональных данных, для сбора и использования даже находящихся в открытом доступе персональных данных нужно получить согласие пользователя. Строго говоря, собирать данные пользователей для запуска таргетированной рекламы — тоже незаконно. Но установить факт парсинга данных при запуске рекламы сейчас технически невозможно, поэтому многие компании продолжают использовать этот инструмент.

Вывод: парсить можно, главное, чтобы этот процесс не приводил к случаям, когда может возникнуть дополнительная ответственность.

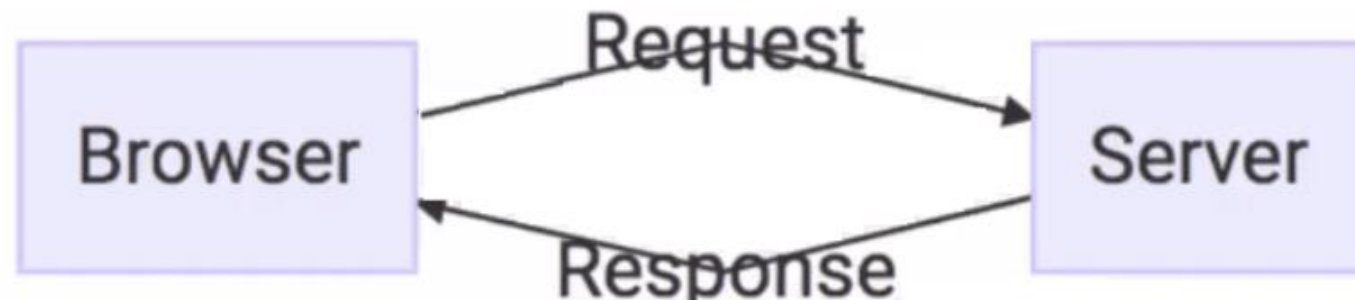
# Как происходит взаимодействие с сайтом

Когда вы вводите в браузере URL-адрес, например [www.google.com](http://www.google.com), на сервер отправляется запрос на веб-сайт, идентифицированный URL-адресом.

Затем этот сервер формирует и выдает ответ. Важным является формат этих запросов и ответов. Эти форматы определяются протоколом **HTTP — Hyper Text Transfer Protocol**.

Когда вы набираете URL в браузере, он отправляет запрос GET на указанный сервер. Затем сервер отвечает HTTP-ответом, который содержит данные в формате **HTML — Hyper Text Markup Language**. Затем браузер получает этот HTML-код и отображает его на экране.

Допустим, вы заполняете форму на веб-странице, со списком элементов. В таком случае, когда вы нажимаете кнопку «**Отправить**», на сервер отправляется HTTP-запрос **POST**.



HTTP — это протокол, который используется для определения структуры запросов (**request**) и ответов (**response**) браузера.

HTTP имеет дело главным образом с ресурсами, доступными на веб-серверах.

Ресурсы идентифицируются с помощью URI, а операции над этими ресурсами выполняются с использованием глаголов, определенных протоколом HTTP.

## Ресурс

Ресурс — это ключевая абстракция, на которой концентрируется протокол HTTP. Ресурс — это все, что вы хотите показать внешнему миру через ваше приложение.

## Методы HTTP-запроса

Метод, используемый в HTTP-запросе, указывает, какое действие вы хотите выполнить с этим запросом. Примеры:

- **GET**: получить подробную информацию о ресурсе
- **POST**: создать новый ресурс
- **PUT**: обновить существующий ресурс
- **DELETE**: Удалить ресурс

**Код состояния HTTP** — часть первой строки ответа сервера при запросах по протоколу HTTP.

Он представляет собой целое число из трёх десятичных цифр.

**Первая цифра указывает на класс состояния.** За кодом ответа обычно следует отделённая пробелом поясняющая фраза на английском языке, которая разъясняет человеку причину именно такого ответа.



Примеры:

- 102 Processing («идёт обработка»);
- 201 Created («создано»);
- 404 Not Found («не найдено»);
- 524 A Timeout Occurred («время ожидания истекло»).

Перечень всех кодов состояния в [википедии](#)

## Задача:

- ✓ Спарсить необходимые данные с сайта
- ✓ Сохранить результаты в файл
- ✓ Провести статистический анализ полученных данных
- ✓ Построить графики



## Шаг 1: Получаем HTML-страницу

**Requests** - это модуль Python, который используется для отправки всех видов HTTP-запросов.

Requests не используется как самостоятельный парсер-скрипт, но она является важной составляющей при работе со структурами URL-адресов и при получении данных.

**!** Библиотека Requests не работает со страницами, написанными на JavaScript.

Пример: С помощью модуля requests получим страницу для парсинга. Для этого нужно отправить GET-запрос, а в качестве ответа получить код страницы и сохранить его:

```
import requests  
url = 'https://belstu.by/'  
data = requests.get(url)  
print(data)  
print(data.text)
```

**import requests** — импортируем библиотеку requests в код скрипта;

**url** — переменная, в которой хранится ссылка на целевую страницу;

**data** — выполняем GET-запрос и передаем в него переменную с хранящейся ссылкой;

**print(data)** — напечатает код состояния ответа;

**print(response.text)** — печатаем код полученной страницы.

## Шаг 2: Парсим страницу

Извлечь полезные данные из полученной html-страницы можно с помощью библиотек для парсинга:

- ✓ **Beautiful Soup (BS4)** – это одна из лучших библиотек Python для парсинга html и xml документов. Позволяет получить доступ напрямую к содержимому любых тегов в html.
- ✓ **Scrapy** – это Python-фреймворк, обеспечивающий быстрое создание асинхронных парсеров данных с веб-сайтов.
- ✓ **Selenium** – это набор инструментов и утилит автоматизации, обеспечивающих взаимодействие с движками браузеров. Они необходимы для поддержки некоторых важных технологий, таких как JavaScript и асинхронный AJAX.
- ✓ **Urllib3** – это ещё популярная библиотека Python, которую часто используют для web-скрапинга. Она отвечает за получение и обработку URL-запросов, с её помощью можно извлекать определённую информацию из ответов сервера, фактически это лёгкий и мощный HTTP-клиент для Python.
- ✓ **LXML** – это многофункциональная библиотека Python, позволяющая производить обработку и генерацию XML-данных на HTML-страницах. LXML подходит для синтаксического анализа содержимого и может сочетаться с другими библиотеками парсеров, например, с BeautifulSoup или со Scrapy.

Подробнее <https://blog.froxy.com/ru/python-web-scraping-libraries>

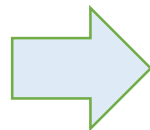
# Beautiful Soup

представляет html-документ в виде вложенной структуры данных:

```
html_doc = """
<html><head><title>The Dormouse's st
<body>
<p class="title"><b>The Dormouse's s

<p class="story">Once upon a time th
<a href="http://example.com/elsie" c
<a href="http://example.com/lacie" c
<a href="http://example.com/tillie"
and they lived at the bottom of a we

<p class="story">...</p>
"""
```



```
from bs4 import BeautifulSoup
soup = BeautifulSoup (html_doc, 'html.parser')

print(soup.prettify())
# <html>
# <head>
# <title>
#   The Dormouse's story
# </title>
# </head>
# <body>
# <p class="title">
#   <b>
#     The Dormouse's story
#   </b>
# </p>
# <p class="story">
#   Once upon a time there were three little sister
#   <a class="sister" href="http://example.com/elsi
#     Elsie
#   </a>
#
```

*Beautiful Soup анализирует документ, используя лучший из доступных парсеров. Библиотека будет использовать HTML-парсер, если вы явно не укажете, что нужно использовать XML-парсер.*

Документация по BeautifulSoup

<https://www.crummy.com/software/BeautifulSoup/bs4/doc.ru/bs4ru.html>

## Пример: Поиск элементов в объекте soup

```
from bs4 import BeautifulSoup
soup = BeautifulSoup(html_doc, 'html.parser')
```

```
soup.a
# <a class="sister" href="http://example.com/elsie" id="link1">Elsie</a>

soup.find_all('a')
# [<a class="sister" href="http://example.com/elsie" id="link1">Elsie</a>,
#  <a class="sister" href="http://example.com/lacie" id="link2">Lacie</a>,
#  <a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>]

soup.find(id="link3")
# <a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>
```

```
import requests  
from bs4 import BeautifulSoup
```

Пример работы с  
библиотекой  
BeautifulSoup

```
# GET запрос к сайту с целью получения html кода  
html = requests.get('https://sunlight.net/catalog').text
```

```
# создаём экземпляр объекта BeautifulSoup в который передаём переменную  
html.
```

```
soup = BeautifulSoup(html, 'lxml')
```

```
# получаем текст конкретного заголовка  
str=soup.find('h2', class_="cl-reviews__title").text  
print(str)
```

```
# получаем список элементов  
items = soup.find_all('div', class_='cl-item-mobile-footer')  
print(len(items))
```

*# webdriver это и есть набор команд для управления браузером*  
`from selenium import webdriver`

*# импортируем класс By, который позволяет выбрать способ поиска элемента*  
`from selenium.webdriver.common.by import By`

*# инициализируем драйвер браузера. После этой команды вы должны увидеть новое открытое окно браузера*  
`driver = webdriver.Chrome()`

*# команда time.sleep устанавливает паузу в 3 секунды, чтобы мы успели увидеть, что происходит в браузере*  
`time.sleep(3)`

`driver.get("https://suninjuly.github.io/text_input_task.html")`  
`time.sleep(3)`

*# Ищем поле для ввода текста*  
`textarea = driver.find_element(By.CSS_SELECTOR, ".textarea")`

*# Напишем текст ответа в найденное поле*  
`textarea.send_keys("get()")`  
`time.sleep(3)`

*# Найдем кнопку, которая отправляет введенное решение*  
`submit_button = driver.find_element(By.CSS_SELECTOR, ".submit-submission")`

`submit_button.click()`  
`time.sleep(3)`

`driver.quit()`

Пример работы с Selenium.  
Здесь открываем ссылку, находим  
окно ввода, вводим строку,  
нажимаем на кнопку Submit.

Selenium применяют для  
парсинга динамически  
загружаемых страниц.

## Шаг 3: Записываем результаты парсинга в файл

### Работа с файлами в Python

Python поддерживает множество различных типов файлов, но условно их можно разделить на два вида: **текстовые** и **бинарные**. Текстовые файлы - это к примеру файлы с расширением csv, txt, html, в общем любые файлы, которые сохраняют информацию в текстовом виде. Бинарные файлы - это изображения, аудио и видеофайлы и т.д. В зависимости от типа файла работа с ним может немного отличаться.


При работе с файлами необходимо соблюдать некоторую последовательность операций:

1. открытие файла с помощью метода **open()**
2. чтение файла с помощью метода **read()** или запись в файл посредством метода **write()**
3. закрытие файла методом **close()**

#### Открытие и закрытие файла

Чтобы начать работу с файлом, его надо открыть с помощью функции **open()**:

```
f = open('text.txt', 'r')
```



Первый параметр функции представляет путь к файлу. Путь файла может быть *абсолютным*, то есть начинаться с буквы диска, например, *C://somedir/somefile.txt*. Либо можно быть *относительным*, например, *somedir/somefile.txt* - в этом случае поиск файла будет идти относительно расположения запущенного скрипта Python.

```
f = open('text.txt', 'r')
```

Второй передаваемый аргумент - **mode** устанавливает режим открытия файла в зависимости от того, что мы собираемся с ним делать. Существует 4 общих режима:

- **r** (Read). Файл открывается для чтения. Если файл не найден, то генерируется исключение `FileNotFoundError`
- **w** (Write). Файл открывается для записи. Если файл отсутствует, то он создается. Если подобный файл уже есть, то он создается заново, и соответственно старые данные в нем стираются.
- **a** (Append). Файл открывается для дозаписи. Если файл отсутствует, то он создается. Если подобный файл уже есть, то данные записываются в его конец.
- **b** (Binary). Используется для работы с бинарными файлами. Применяется вместе с другими режимами - `w` или `r`.

После завершения работы с файлом его обязательно нужно закрыть методом **close()**. Данный метод освободит все связанные с файлом используемые ресурсы.

Например:

```
somefile = open("hello.txt", "r")  
...  
somefile.close()
```



При открытии файла или в процессе работы с ним можно столкнуться с различными исключениями, например, к нему нет доступа и т.д. В этом случае программа выпадет в ошибку, а ее выполнение не дойдет до вызова метода close, и соответственно файл не будет закрыт.

В этом случае можно обрабатывать исключения:

```
try:
    somefile = open("hello.txt", "w")
    try:
        somefile.write("hello world")
    except Exception as e:
        print(e)
    finally:
        somefile.close()
except Exception as ex:
    print(ex)
```

В данном случае вся работа с файлом идет во вложенном блоке try. И если вдруг возникнет какое-либо исключение, то в любом случае в блоке finally файл будет закрыт.

Однако есть и более удобная конструкция - конструкция with:

```
with open("hello.txt", "w") as file_obj:
    file_obj.write("hello world")
```

Эта конструкция определяет для открытого файла переменную file\_obj и выполняет набор инструкций. После их выполнения файл автоматически закрывается. Даже если при выполнении инструкций в блоке with возникнут какие-либо исключения, то файл все равно закрывается.

# Чтение и запись файлов CSV

Формат **CSV** (Comma Separated Values) является наиболее распространенным форматом импорта и экспорта для электронных таблиц и баз данных. Каждая строка в файле csv представляет отдельную запись или строку, которая состоит из отдельных столбцов, разделенных запятыми (или точкой с запятой).

```
1 import csv
2
3 FILENAME = "users.csv"
4
5 users = [
6     ["Tom", 28],
7     ["Alice", 23],
8     ["Bob", 34]
9 ]
10
11 with open(FILENAME, "w", newline="") as file:
12     writer = csv.writer(file)
13     writer.writerows(users)
14
15
16 with open(FILENAME, "a", newline="") as file:
17     user = ["Sam", 31]
18     writer = csv.writer(file)
19     writer.writerow(user)
```

Для упрощения работы с этим форматом предоставляется специальный встроенный модуль **csv**.

объект **writer** возвращается функцией **csv.writer(file)**. В эту функцию передается открытый файл.

Запись производится с помощью метода **writer.writerows(users)**. Этот метод принимает набор строк. В нашем случае это двухмерный список.

Если необходимо добавить одну запись, которая представляет собой одномерный список, то в этом случае можно вызвать метод **writer.writerow(user)**

Для чтения из файла нужно создать объект **reader**:

```
1 import csv
2
3 FILENAME = "users.csv"
4
5 with open(FILENAME, "r", newline="") as file:
6     reader = csv.reader(file)
7     for row in reader:
8         print(row[0], " - ", row[1])
```

Если кодировка не совпадает с ASCII, то нужно явным образом указать кодировку с помощью параметра **encoding**:

```
def save_file(items, path): 1 usage
    with open(path, 'w', newline='', encoding="utf-8") as file:
        writer = csv.writer(file, delimiter=';')
        writer.writerow(['Name', 'Price'])
```

## Работа со словарями

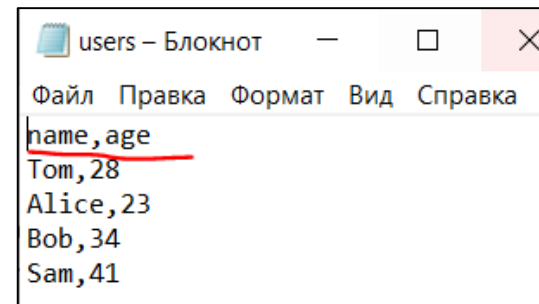
Модуль csv имеет специальные дополнительные возможности для работы со словарями. В частности, функция **csv.DictWriter()** возвращает объект writer, который позволяет записывать в файл. А функция **csv.DictReader()** возвращает объект reader для чтения из файла:

```
1 import csv
2
3 FILENAME = "users.csv"
4
5 users = [
6     {"name": "Tom", "age": 28},
7     {"name": "Alice", "age": 23},
8     {"name": "Bob", "age": 34}
9 ]
10
11 with open(FILENAME, "w", newline="") as file:
12     columns = ["name", "age"]
13     writer = csv.DictWriter(file, fieldnames=columns)
14     writer.writeheader()
15
16     # запись нескольких строк
17     writer.writerows(users)
18
19     user = {"name": "Sam", "age": 41}
20     # запись одной строки
21     writer.writerow(user)
22
23 with open(FILENAME, "r", newline="") as file:
24     reader = csv.DictReader(file)
25     for row in reader:
26         print(row["name"], "-", row["age"])
27
```

В этом случае каждая строка представляет собой отдельный словарь, и кроме того, производится запись и заголовков столбцов с помощью метода `writeheader()`, а в метод `csv.DictWriter` в качестве второго параметра передается набор столбцов.

При чтении строк, используя названия столбцов, можно обратиться к отдельным значениям внутри строки: `row["name"]`.

```
with open(FILENAME, "r", newline="") as file:
    reader = csv.DictReader(file)
    for row in reader:
        print(row["name"])
```



```
=====
Tom - 28
Alice - 23
Bob - 34
Sam - 41
>>>
```

```
def save_file(items,path):  
    with open(path, 'w', newline="",encoding="utf-8") as file:  
        writer = csv.writer(file, delimiter=';')  
        writer.writerow(['Name', 'Price'])  
        for item in items:  
            writer.writerow([item['name'], item['price']])
```

Запись файла

```
html = requests.get('https://sunlight.net/catalog').text  
soup = BeautifulSoup(html,'xml')  
items = soup.find_all('div', class_='cl-item-mobile-footer')
```

```
jewelry=[]  
for item in items:  
    soup = BeautifulSoup(html)  
    name=item.find('div', class_='cl-item-mobile-footer__subname').get_text(strip=True)  
    price = item.find('div', class_='cl-item-mobile-footer-price').find_all('span')[0].text
```

```
    jewelry.append({  
        'name': name,  
        'price': price  
    })
```

```
save_file(jewelry,"catalog.csv")
```

## Шаг 4: Анализируем результаты

Нам понадобятся две библиотеки Python:

- **NumPy** (Numerical Python) используется для обработки больших массивов данных. Он упрощает математические операции и их векторизацию на массивах.
- **Pandas** упрощает работу с табличными данными.



# Что такое NumPy?



NumPy — библиотека с открытым исходным кодом для Python, реализующая множество математических операций для работы с векторами, матрицами и массивами. Важное преимущество NumPy перед собственной реализацией массивов (например на списках) - **это векторные операции, которые происходят гораздо быстрее, последовательных.**

Фактически, NumPy - это основная математическая библиотека для работы с данными (если вы решаете задачи машинного обучения или анализа данных). Именно NumPy, а не встроенный Math.

NumPy лежит в основе других важных библиотек: Pandas (работа с табличными данными), SciPy (работы с методами оптимизации и научными расчётами), Matplotlib (построение графиков) и т.д.

Pandas — это библиотека Python, предоставляющая широкие возможности для анализа данных.



Pandas спроектирована на основе библиотеки NumPy. Такой выбор делает pandas совместимой с большинством других модулей.

Еще одно важное решение — разработка специальных структур для анализа данных. Вместо того, чтобы использовать встроенные в Python или предоставляемые другими библиотеками структуры, были разработаны две новых **Series** и **DataFrame**.

Данные, часто хранятся в форме табличек — например, в форматах .csv, .tsv или .xlsx. С помощью библиотеки Pandas такие табличные данные очень удобно загружать, обрабатывать и анализировать.

В связке с библиотеками Matplotlib и Seaborn Pandas предоставляет широкие возможности визуального анализа табличных данных.



# Структуры данных в pandas

- **Series**
- **Dataframe**

**Series** — это объект библиотеки pandas для представления одномерных структур данных.

**Series** состоит из двух связанных между собой массивов. Основной содержит данные (данные любого типа NumPy), а в дополнительном, index, хранятся метки.

Series	
index	value
0	12
1	-4
2	7
3	9

Создать структуру **Series** можно на базе различных типов данных:

- словарей Python;
- списков Python;
- массивов из numpy: ndarray;
- скалярных величин.

**DataFrame** – это двумерная структура данных, представляющая собой таблицу, каждый столбец которой содержит данные одного типа. Можно представлять её как словарь объектов типа Series. Структура DataFrame отлично подходит для представления реальных данных.

В отличие от Series у которого есть массив индексов с метками, ассоциированных с каждым из элементов, Dataframe имеет сразу два таких. Первый ассоциирован со строками (рядами) и напоминает таковой из Series. Каждая метка ассоциирована со всеми значениями в ряду. Второй содержит метки для каждой из колонок.

DataFrame			
index	columns		
	color	object	price
0	blue	ball	1.2
1	green	pen	1.0
2	yellow	pencil	0.6
3	red	paper	0.9
4	white	mug	1.7

DataFrame												
	date	number_of_game	day_of_week	v_name	v_league	v_game_number	h_name	h_league	h_game_number	v_score	h_score	length_outs
0	01871054	0	Thu	CL1	na	1	FW1	na	1	0	2	54.0
1	18710505	0	Fri	BS1	na	1	WS3	na	1	20	18	54.0
2	18710506	0	Sat	CL1	na	2	RC1	na	1	12	4	54.0

IntBlock

	0	1	2	3	4	5
0	01871054	0	1	1	0	2
1	18710505	0	1	1	20	18
2	18710506	0	2	1	12	4

ObjectBlock

	0	1	2	3	4
0	Thu	CL1	na	FW1	na
1	Fri	BS1	na	WS3	na
2	Sat	CL1	na	RC1	na

FloatBlock

	0
0	54.0
1	54.0
2	54.0

Внутреннее представление данных разных типов в pandas  
Внутри pandas столбцы данных группируются в блоки со значениями одинакового типа. На рисунке пример того, как в pandas хранятся первые 12 столбцов объекта DataFrame.

```
In [ ]: import pandas as pd
import numpy as np
```

```
In [6]: ave_data = np.array(np.arange(24)).reshape((6,4))
dates = pd.date_range('20210101', periods=6)
ave_df = pd.DataFrame(ave_data, index=dates, columns=list('1234'))
ave_df
```

Out[6]:

	1	2	3	4
2021-01-01	0	1	2	3
2021-01-02	4	5	6	7
2021-01-03	8	9	10	11
2021-01-04	12	13	14	15
2021-01-05	16	17	18	19
2021-01-06	20	21	22	23


```
In [7]: ave_df['4']
```

```
Out[7]: 2021-01-01    3
2021-01-02    7
2021-01-03   11
2021-01-04   15
2021-01-05   19
2021-01-06   23
Freq: D, Name: 4, dtype: int32
```

```
In [9]: ave_df[1:4]
```

Out[9]:

	1	2	3	4
2021-01-02	4	5	6	7
2021-01-03	8	9	10	11
2021-01-04	12	13	14	15

Getting started **User Guide** API reference Development Release notes

10 minutes to pandas

Intro to data structures

Essential basic functionality

IO tools (text, CSV, HDF5, ...)

Indexing and selecting data

MultiIndex / advanced indexing

Merge, join, concatenate and compare

Reshaping and pivot tables

Working with text data

Working with missing data

Duplicate Labels

Categorical data

Null and integer data types

# 10 minutes to pandas

This is a short introduction to pandas, geared mainly for new users. You can see more complex recipes in the [Cookbook](#).

Customarily, we import as follows:

```
In [1]: import numpy as np
In [2]: import pandas as pd
```

## Object creation

See the [Data Structure Intro](#) section.

Creating a **Series** by passing a list of values, letting pandas create a default integer index:

# Пример на датасете Титаник

```
In [1]: import pandas as pd
```

```
In [2]: titanic = pd.read_csv("data/titanic.csv")
```

## Как рассчитать сводную статистику?

### Агрегированная статистика

Какой средний возраст пассажиров Титаника?

```
In [6]: titanic["Age"].mean()
```

```
Out[6]: 29.69911764705882
```

Каков средний возраст и стоимость билета пассажиров «Титаника»?

```
In [7]: titanic[["Age", "Fare"]].median()
```

```
Out[7]: Age      28.0000  
Fare      14.4542  
dtype: float64
```

## Функция `pandas.DataFrame.describe` рассчитывает параметры описательной статистики

In [8]: `titanic[["Age", "Fare"]].describe()`

Out[8]:

	Age	Fare
<b>count</b>	714.000000	891.000000
<b>mean</b>	29.699118	32.204208
<b>std</b>	14.526497	49.693429
<b>min</b>	0.420000	0.000000
<b>25%</b>	20.125000	7.910400
<b>50%</b>	28.000000	14.454200
<b>75%</b>	38.000000	31.000000
<b>max</b>	80.000000	512.329200

Вместо predefined статистики можно определить конкретные комбинации агрегированной статистики для заданных столбцов с помощью `DataFrame.agg()` метода:

[10]: `titanic.agg(  
 {  
 "Age": ["min", "max", "median"],  
 "Fare": ["min", "max", "median", "mean"],  
 }  
)`

[10]:

	Age	Fare
<b>min</b>	0.42	0.000000
<b>max</b>	80.00	512.329200
<b>median</b>	28.00	14.454200

Каков средний возраст пассажиров Титаника мужчинами и женщинами?

```
In [11]: titanic[["Sex", "Age"]].groupby("Sex").mean()
```

```
Out[11]:
```

	Age
Sex	

female	27.915709
male	30.726645

Если не указывать столбцы, то mean-метод применяется к каждому столбцу, содержащему числовые данные:

```
In [12]: titanic.groupby("Sex").mean()
```

```
Out[12]:
```

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
Sex							

female	431.028662	0.742038	2.159236	27.915709	0.694268	0.649682	44.479818
male	454.147314	0.188908	2.389948	30.726645	0.429809	0.235702	25.523893



## Шаг 5: Визуализируем результаты



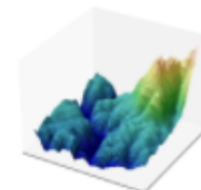
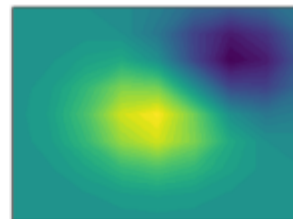
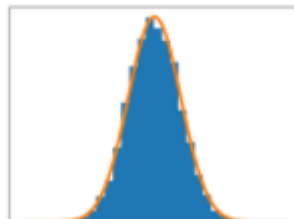
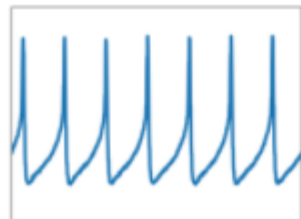
<https://matplotlib.org/>

Монтаж   Документация   Примеры   Учебники   Содействие

[домой](#) | [содержание](#) » [Matplotlib: построение графиков Python](#)

## Matplotlib: Визуализация с помощью Python

Matplotlib - это комплексная библиотека для создания статических, анимированных и интерактивных визуализаций в Python.



Matplotlib делает легкие вещи легкими, а сложные - возможными.



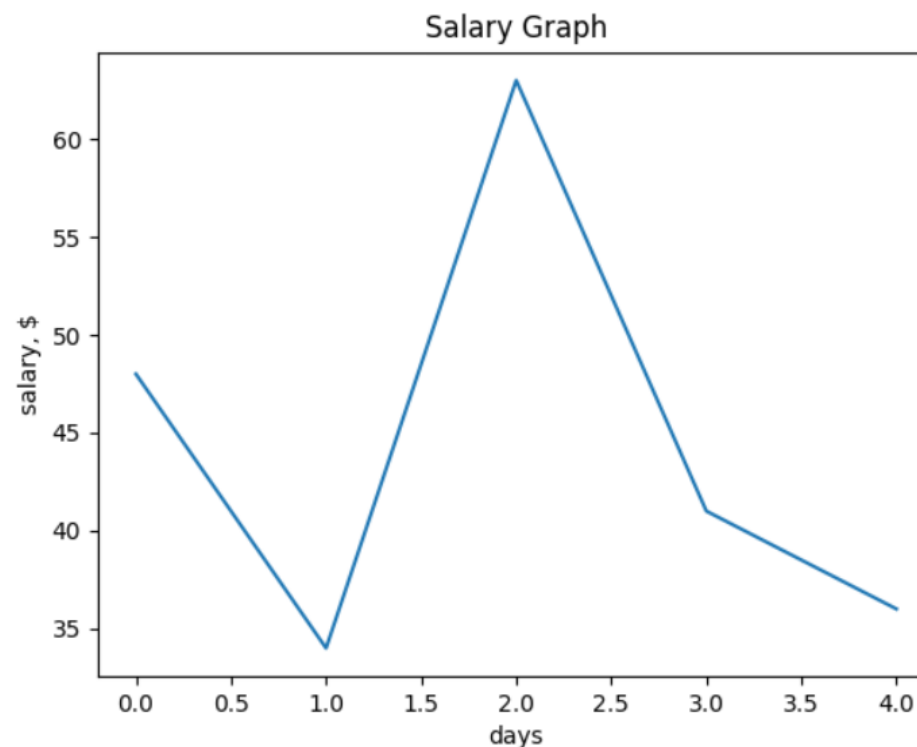
Функция **plot()** является универсальной функцией и принимает произвольное количество аргументов.

```
import matplotlib.pyplot as plt
plt.plot([1, 2, 3, 4]) ← y
plt.ylabel('some numbers')
plt.show()
```

x                      y

```
plt.plot([1, 2, 3, 4], [1, 4, 9, 16])
```

```
1 import matplotlib.pyplot as plt
2
3 x_list=list(range(0,5))
4 y_list= [48,34,63,41,36]
5
6 plt.title('Salary Graph')
7 plt.xlabel('days')
8 plt.ylabel('salary, $')
9 plt.plot(x_list,y_list)
10
11 plt.show()
```



# Маркеры

```
import matplotlib.pyplot as plt

x_list=list(range(0,5))
y_list= [48,34,63,41,36]

plt.title('Salary Graph')
plt.xlabel('days')
plt.ylabel('salary, $')
plt.plot(x_list,y_list,marker='o')

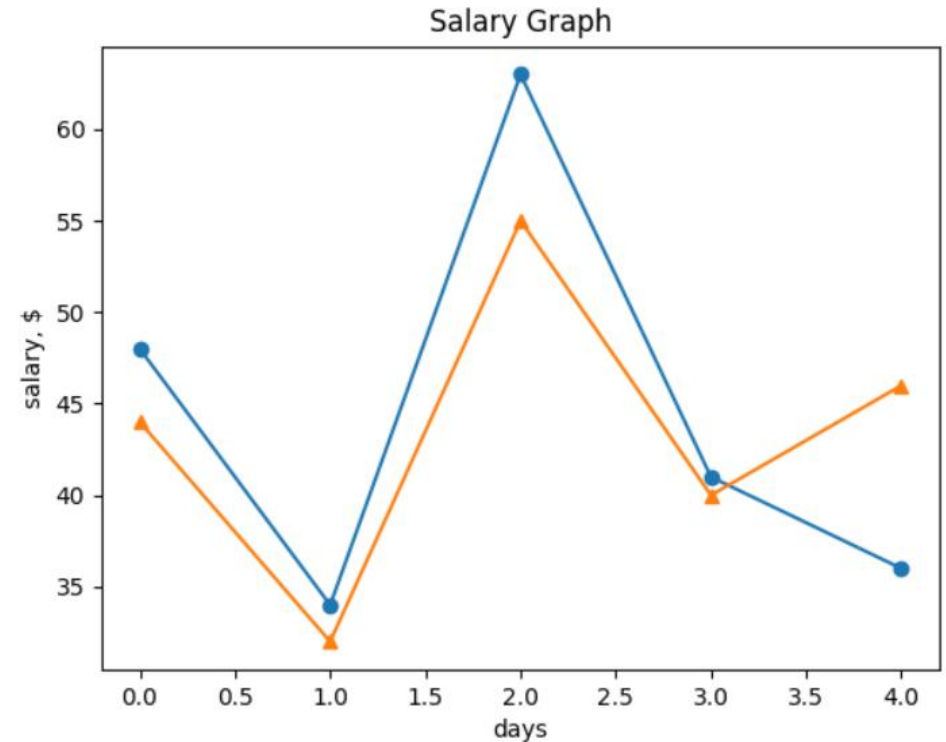
plt.show()
```

marker	symbol	description
"."	•	point
","	.	pixel
"o"	●	circle
"v"	▼	triangle_down
"^"	▲	triangle_up
"<"	◀	triangle_left
">"	▶	triangle_right
"1"	⋈	tri_down
"2"	⋈	tri_up
"3"	↙	tri_left
"4"	↘	tri_right
"8"	●	octagon
"s"	■	square
"p"	⬠	pentagon
"p"	⊕	plus (filled)
"*"	★	star

[https://matplotlib.org/stable/api/markers\\_api.html](https://matplotlib.org/stable/api/markers_api.html)

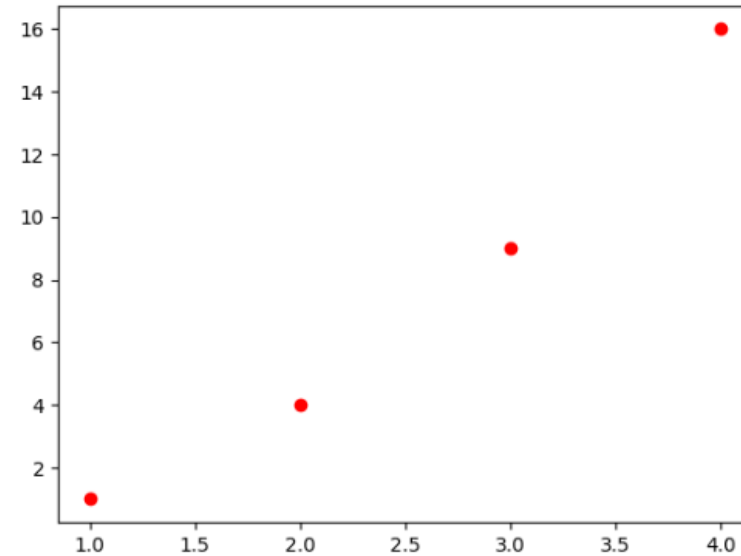
## Два графика в одних осях

```
1 import matplotlib.pyplot as plt
2
3 x_list=list(range(0,5))
4 y1_list= [48,34,63,41,36]
5 y2_list= [44,32,55,40,46]
6
7 plt.title('Salary Graph')
8 plt.xlabel('days')
9 plt.ylabel('salary, $')
10 plt.plot(x_list,y1_list,marker='o')
11 plt.plot(x_list,y2_list,marker='^')
12 plt.show()
```



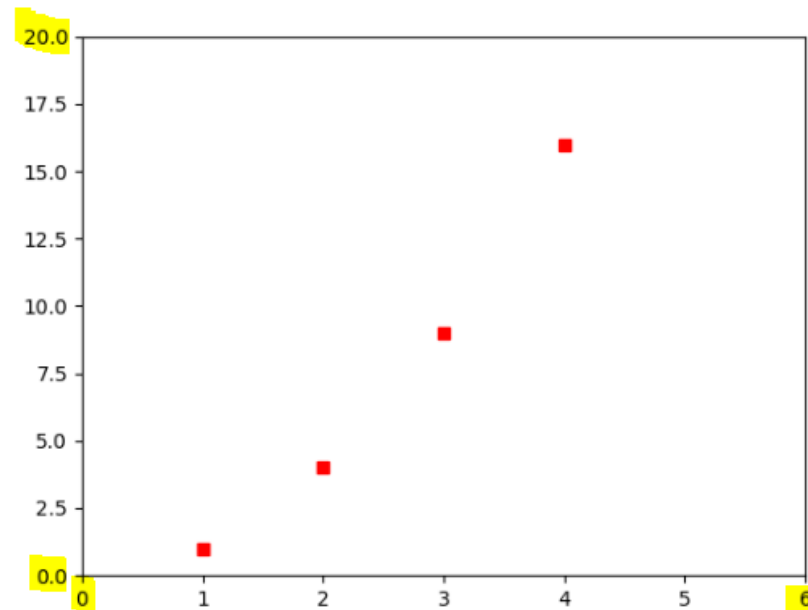
```
>>> plot(x, y)           # plot x and y using default line style and color
>>> plot(x, y, 'bo')     # plot x and y using blue circle markers
>>> plot(y)              # plot y using x as index array 0..N-1
>>> plot(y, 'r+')        # ditto, but with red plusses
```

```
1 import matplotlib.pyplot as plt
2
3 plt.plot([1, 2, 3, 4], [1, 4, 9, 16], 'ro')
4
5 plt.show()
```



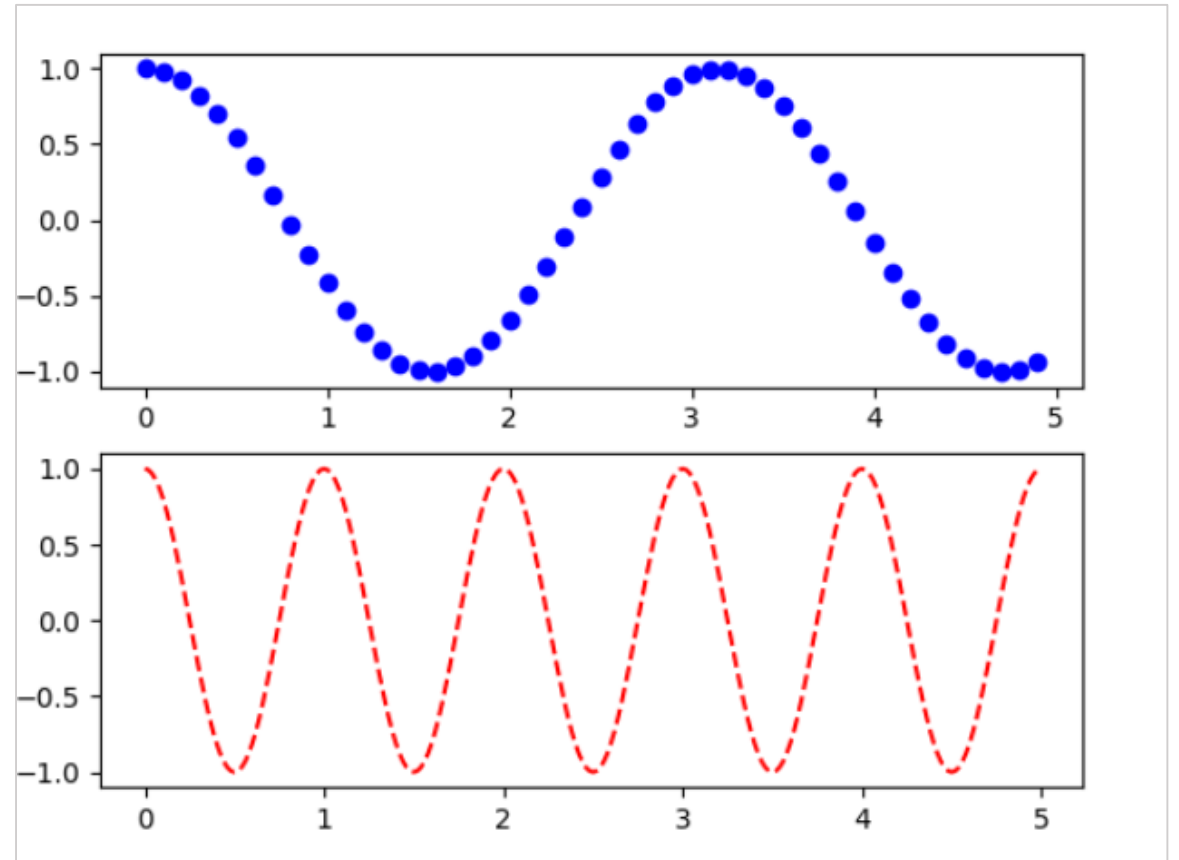
## Форматирование диапазона осей

```
1 import matplotlib.pyplot as plt
2
3 plt.plot([1, 2, 3, 4], [1, 4, 9, 16], 'rs')
4 plt.axis([0, 6, 0, 20])
5
6 plt.show()
```



# Размещение графиков

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 def f(t):
5     return np.cos(2*np.pi*t2)
6
7 t1 = np.arange(0.0, 5.0, 0.1)
8 t2 = np.arange(0.0, 5.0, 0.02)
9
10 plt.figure()
11 plt.subplot(211)
12 plt.plot(t1, np.cos(2*t1), 'bo')
13
14 plt.subplot(212)
15 plt.plot(t2, f(t2), 'r--')
16 plt.show()
```



Параметры **subplot()**:

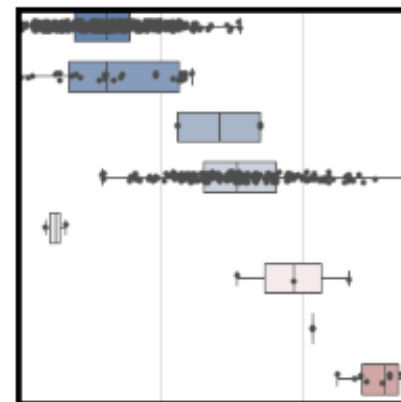
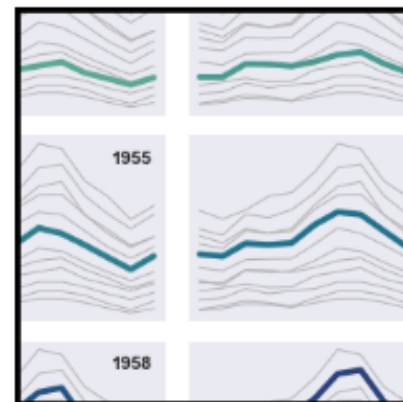
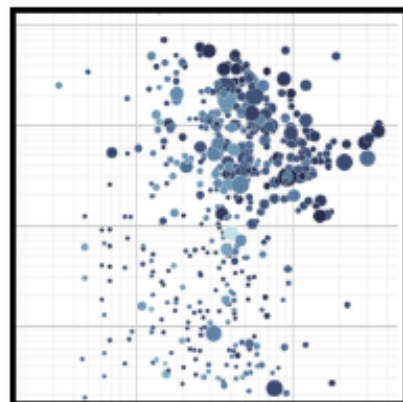
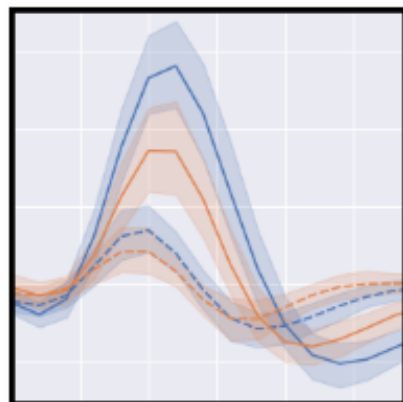
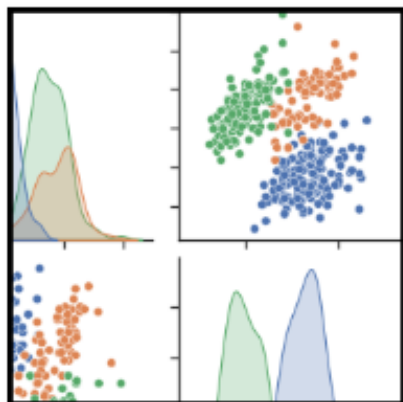
1. количество строк;
2. количество столбцов
3. индекс ячейки.

**Seaborn** — это библиотека для создания статистических графиков на Python. Она основывается на matplotlib и тесно взаимодействует со структурами данных pandas.



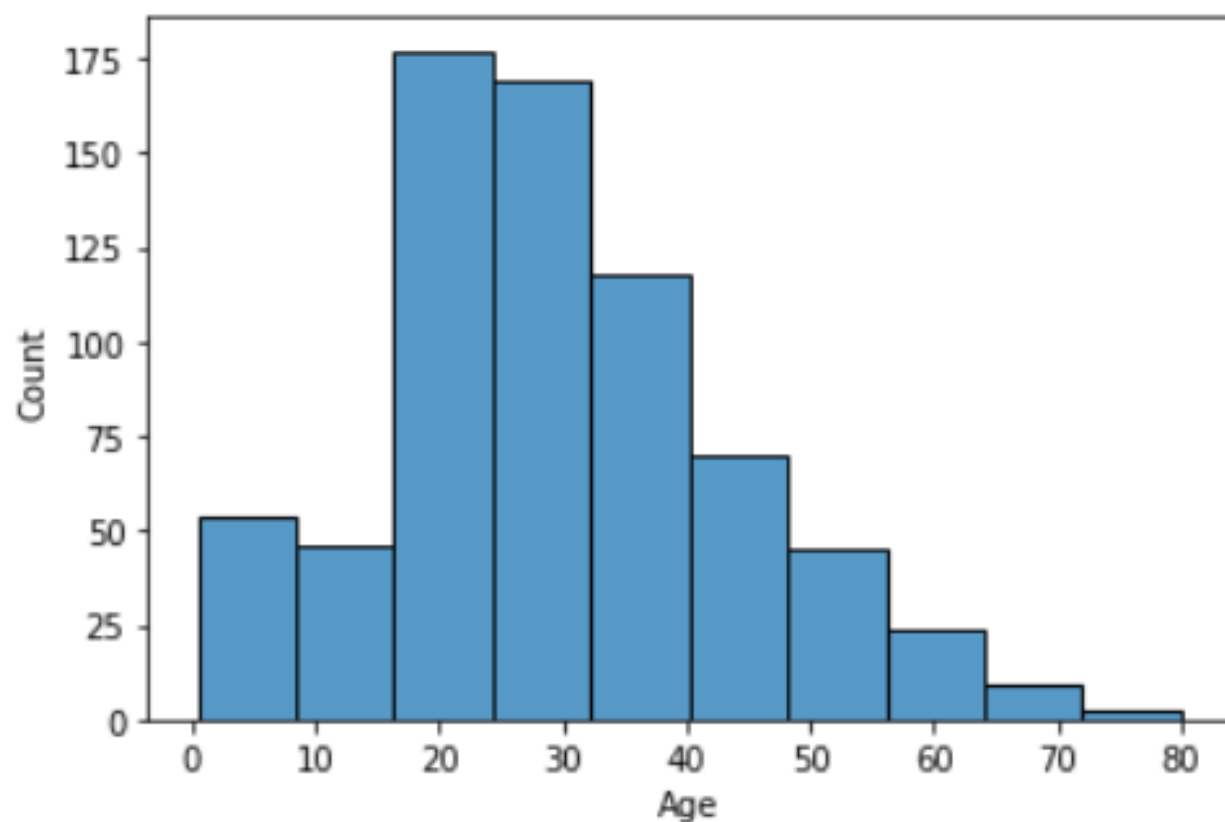
[Installing](#) [Gallery](#) [Tutorial](#) [API](#) [Releases](#) [Citing](#) [FAQ](#)

## seaborn: statistical data visualization



```
In [4]: import seaborn as sns
sns.histplot(data=titanic["Age"],bins=10)
```

Out[4]: <AxesSubplot:xlabel='Age', ylabel='Count'>



```
In [36]: s.describe()
```

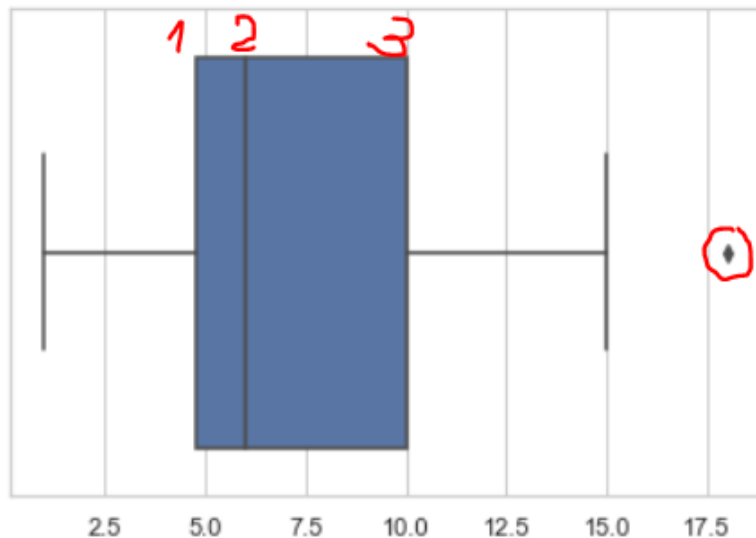
```
Out[36]: count    20.000000  
mean      7.250000  
std       4.191156  
min       1.000000  
25%      4.750000 1  
50%      6.000000 2  
75%     10.000000 3  
max      18.000000  
dtype: float64
```

```
In [34]: s.median()
```

```
Out[34]: 6.0
```

```
In [48]: sns.boxplot(x=s)
```

```
Out[48]: <AxesSubplot:>
```



## График box-plot

Центром ящика является медиана наших данных или второй квартиль, верхняя граница = 3-й квартиль, а нижняя граница = 1-й квартиль.

### Почему некоторые точки на графике отображены отдельно?

Если мы посчитаем разность между 3-м и 1-м квартилем - это межквартильный размах (мера изменчивости).

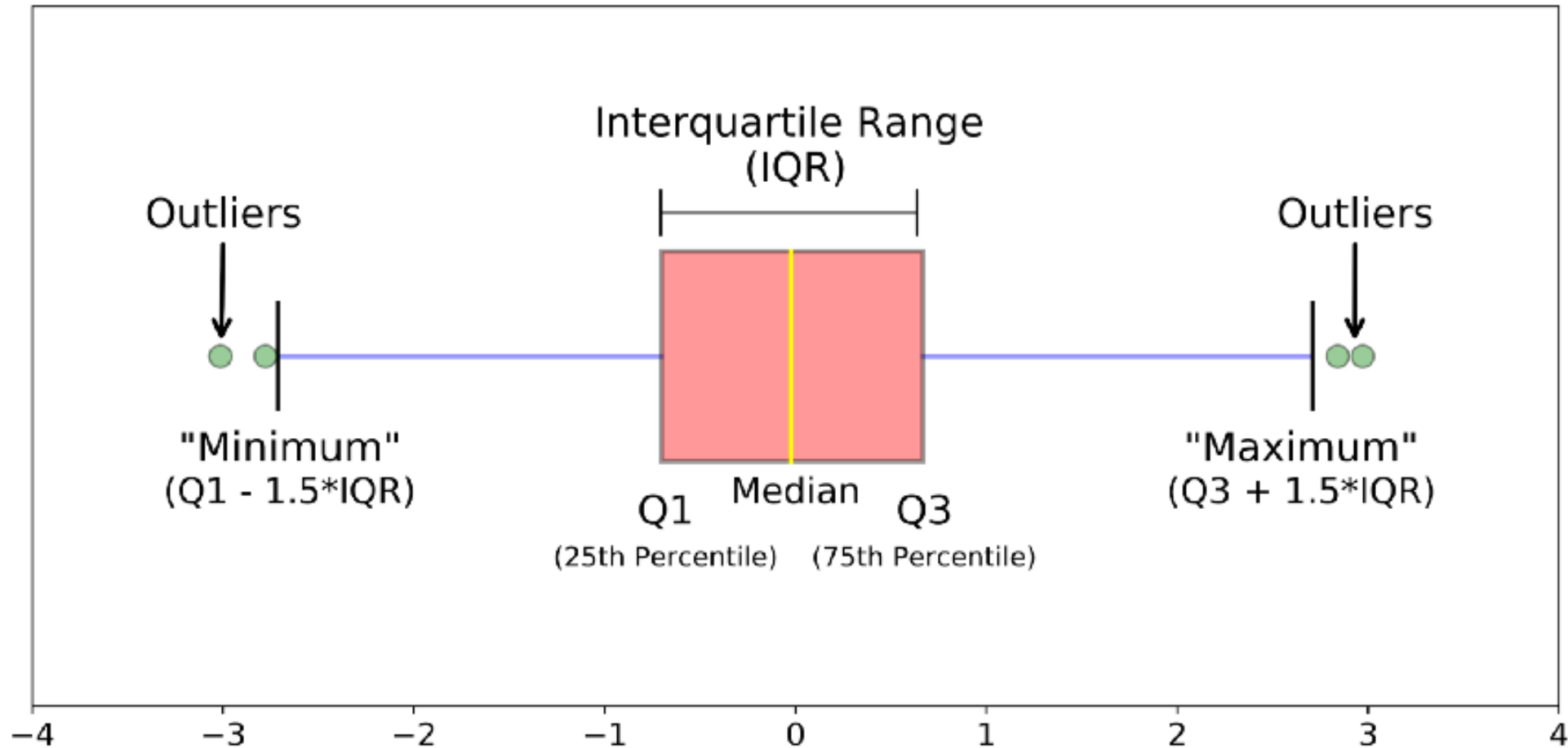
Чем выше межквартильный размах, тем больше вариативность нашего признака.

Отложим мысленно 1,5 межквартильного размаха вверх и вниз от 1-го и 3-го квартилей. Те значения признака, которые последними принадлежат этому промежутку и будут границами усов.

Точки, которые превосходят полтора межквартильного размаха - наносятся на график отдельно.



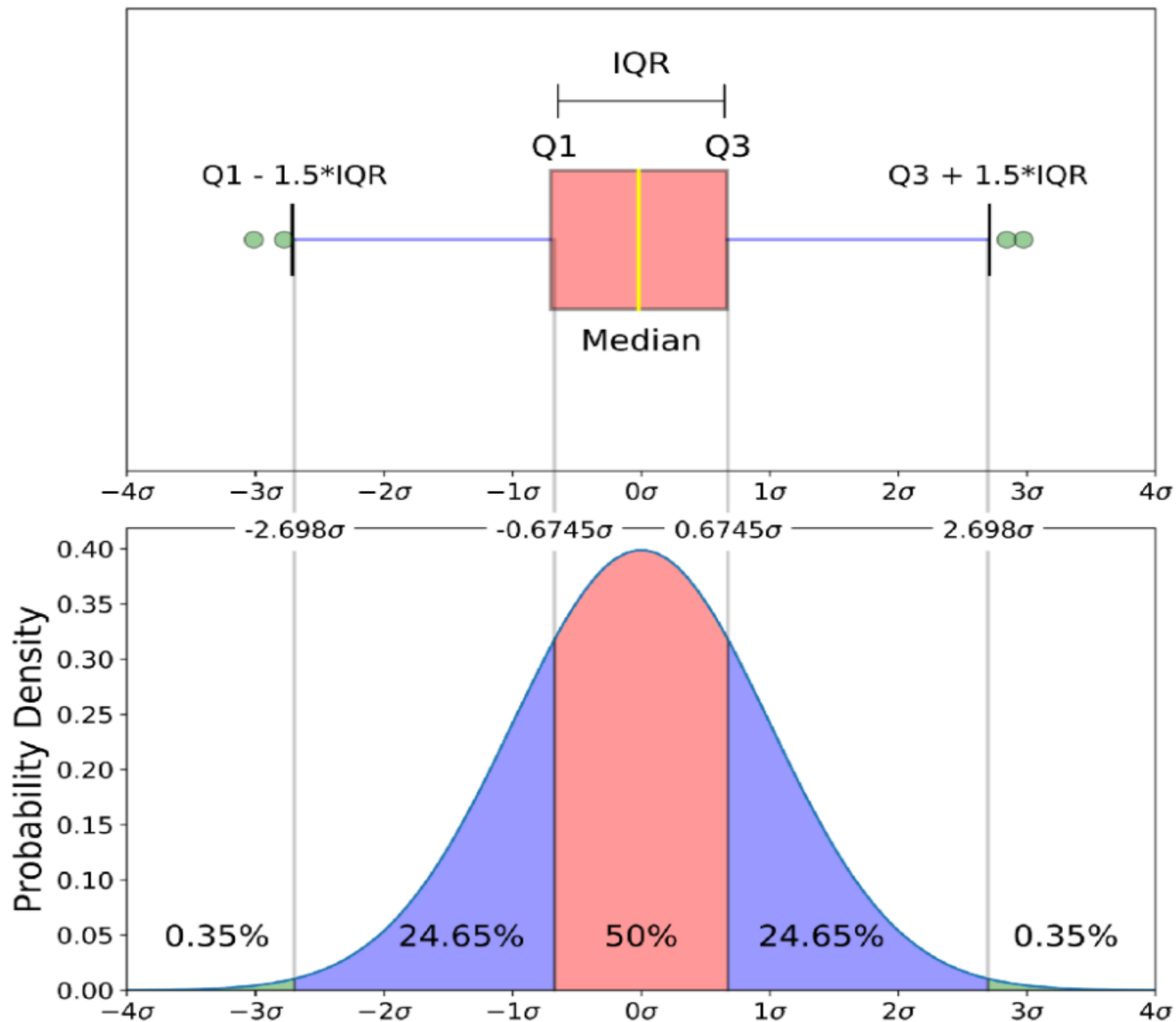
# box plot Ящик с усами Диаграмма размаха



IQR - размах

Q1 – 25 перцентиль (1й квартиль)

Q3 – 75 перцентиль (3й квартиль)

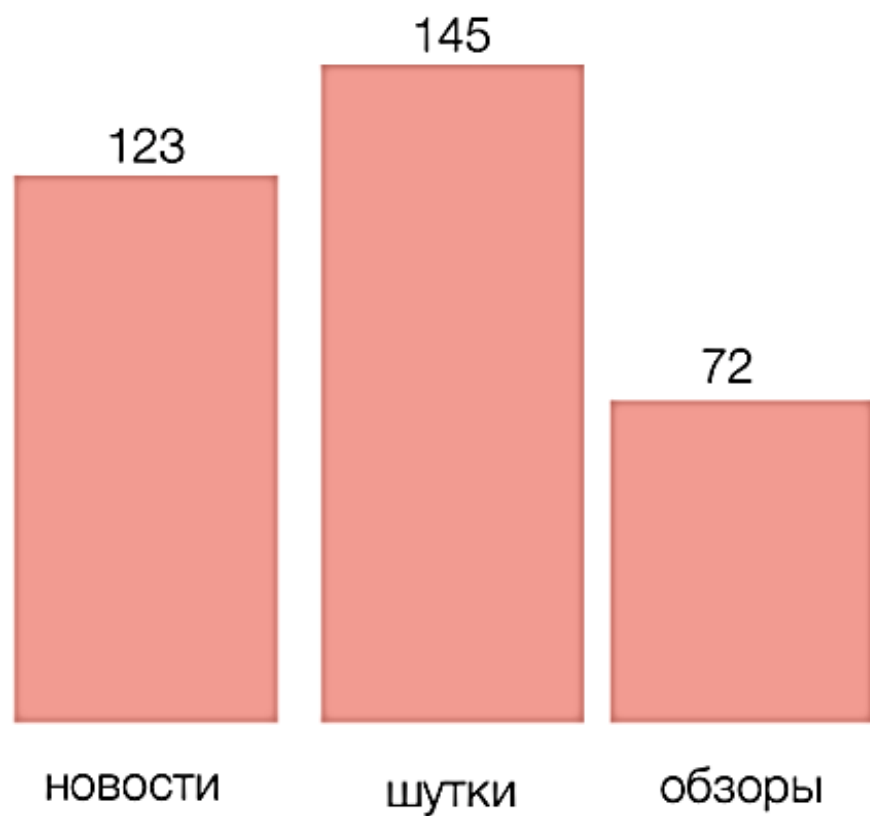


Box-plot наглядно визуализирует распределение случайной величины.

Из рисунка видно, что ящик содержит 50 процентов всех данных, а за пределами усов содержится 0,35 процентов, т.е. очень маленькое количество данных, которые можно расценить как выбросы.

## Пример: анализ эффективных постов блога

Сумма лайков по категориям



Распределение лайков по категориям

