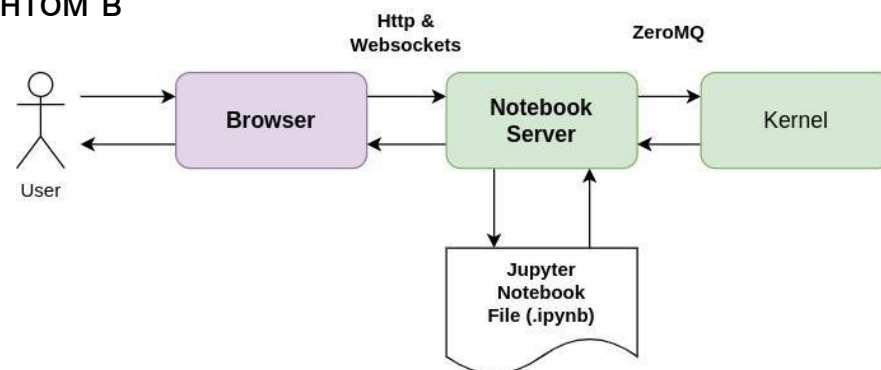


# Лекция 1: Инструменты Python 3 для задач Data mining

# Jupyter (акроним от **J**ulia, **P**ython, **R**)

- Открытое ПО для разработки, документации и выполнения кода в интерактивном режиме
- Поддерживает множество языков: R, Python, Cython, Julia, Scala, C++ (проект xeus-cling)
- В курсе рассматривается Jupyter Notebook:
  - Документы `.ipynb`: представление кода, данных в вычислениях, пояснительный текст и мультимедиа
  - Веб-приложение: управление документом в браузере
- Установка и запуск
  - Установка через `pip`  
`pip install notebook`
  - Запуск через консоль  
`[console]: jupyter notebook`

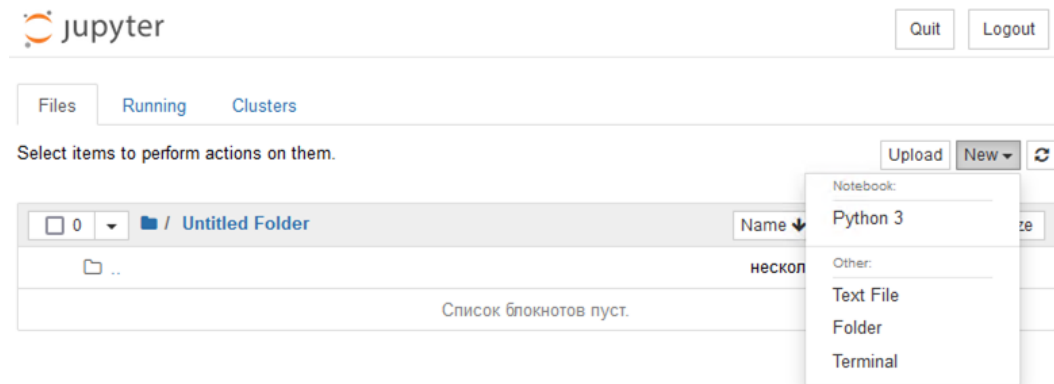


# Jupyter Notebook

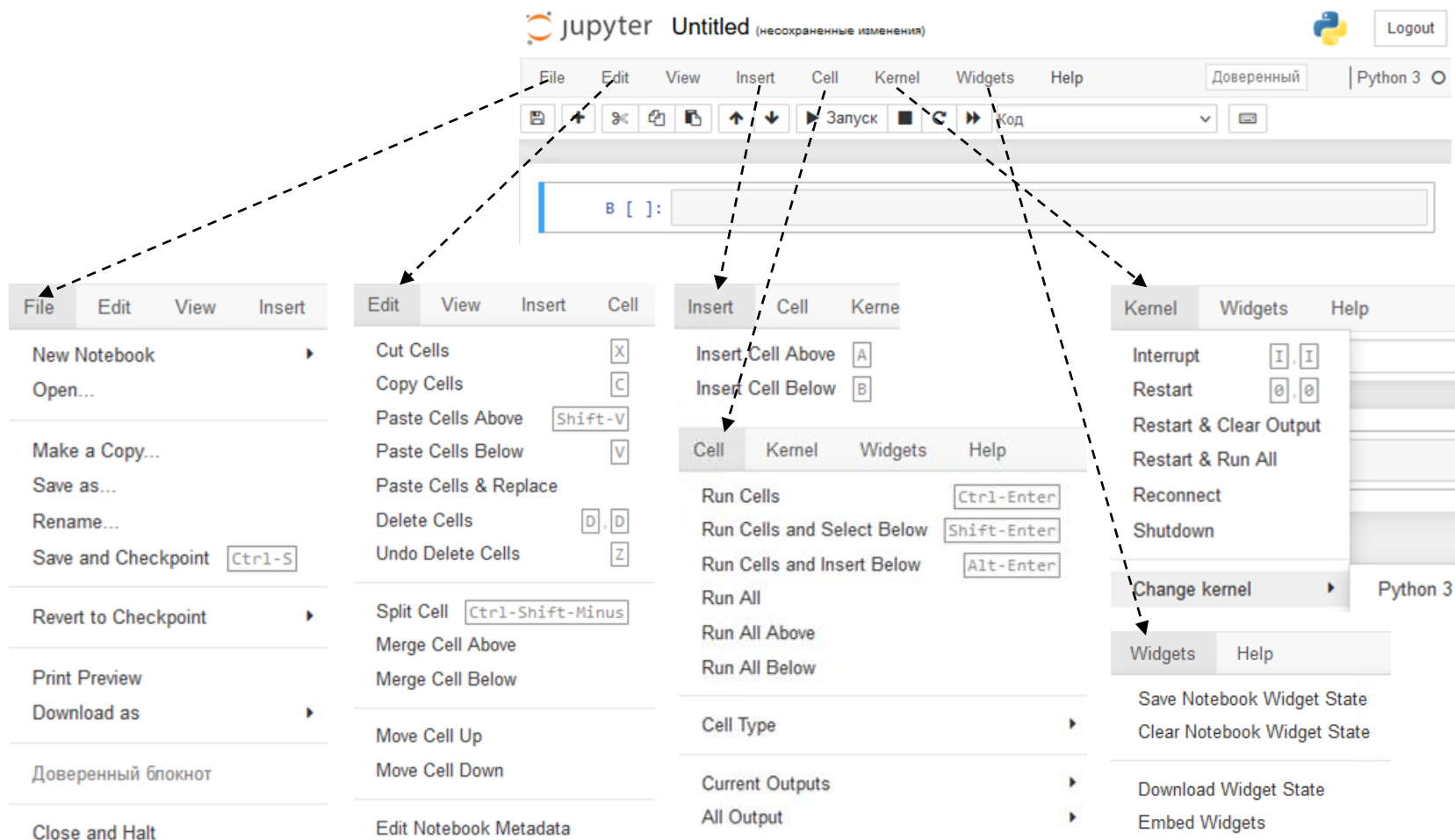
- Запуск сервера через консоль
- Окно выбора JN
- Сортировка по имени, времени модификации и размеру
- Загрузка и создание JN
- Running: список запущенных JN
- Clusters: доступ к JN на удаленном сервере

```
(base) D:\Archive>jupyter notebook
[I 11:56:03.484 NotebookApp] JupyterLab extension loaded from C:\ProgramDa
[I 11:56:03.484 NotebookApp] JupyterLab application directory is C:\Progra
[I 11:56:03.489 NotebookApp] Serving notebooks from local directory: D:\Ar
[I 11:56:03.489 NotebookApp] Jupyter Notebook 6.1.4 is running at:
[I 11:56:03.489 NotebookApp] http://localhost:8888/?token=5f0baf6d0ce75639
[I 11:56:03.490 NotebookApp] or http://127.0.0.1:8888/?token=5f0baf6d0ce7
[I 11:56:03.490 NotebookApp] Use Control-C to stop this server and shut do
[C 11:56:03.503 NotebookApp]

To access the notebook, open this file in a browser:
file:///C:/Users/vasiliev/AppData/Roaming/jupyter/runtime/nbserv
Or copy and paste one of these URLs:
http://localhost:8888/?token=5f0baf6d0ce75639a1f9bbdf9132c4af1d74f
or http://127.0.0.1:8888/?token=5f0baf6d0ce75639a1f9bbdf9132c4af1d74f
[I 11:56:04.686 NotebookApp] 302 GET /tree (127.0.0.1) 1.00ms
```



# Интерфейс JN



# Jupyter Notebook

- Запуск ячеек и порядок выполнения
- Остановка выполнения:
  - Прерывание (Interrupt)
  - Перезапуск с сохранением выходных данных (Restart), со сбросом выходных данных (Restart & Clear Output)
  - Restart + выполнение (Restart & Run All)
- Область видимости переменных
- Интерактивный ввод

The diagram illustrates the Jupyter Notebook interface and the actions available to manage the execution state. It shows three code cells, each containing the same Python code: `B [ ]: import numpy as np` followed by `np.array([1, 2, 3], dtype=int)`. The second cell has its output displayed: `Out[1]: array([1, 2, 3])`. To the right of the code cells is a toolbar with several icons. The 'Run' button (a play icon) is highlighted with a red box and labeled 'Запуск'. Below the toolbar is a 'Kernel' menu with several options: 'Interrupt', 'Restart', 'Restart & Clear Output' (highlighted with a red box), 'Restart & Run All', 'Reconnect', and 'Shutdown'. Arrows indicate the flow of actions: from the 'Run' button to the first and third code cells, and from the 'Restart & Clear Output' button to the second code cell.

This diagram illustrates variable scope and interactive input. It shows two code cells. The first cell, labeled 'B [\*]:', contains the code `a = input()`. Below it, a text input field contains the value '123'. An arrow points from this input field to the second code cell, which is labeled 'B [1]:' and contains the code `a = input()`. The output of the second cell is shown as 'Out[2]: '123'', with an arrow pointing from the output to the input field of the first cell. This demonstrates how the variable 'a' is updated in the current cell and its value is reflected in the output of the subsequent cell.

This diagram illustrates variable scope and execution order. It shows two sets of code cells. The first set, labeled 'B [1]:', 'B [ ]:', and 'B [2]:', contains the code `a = 1`, `a = 2`, and `print(a)` respectively. The output of the first cell is '1'. The second set, labeled 'B [1]:', 'B [3]:', and 'B [4]:', contains the code `a = 1`, `a = 2`, and `print(a)` respectively. The output of the second cell is '2'. An arrow points from the first set of cells to the second set, indicating the flow of execution and the state of the variable 'a' after each step.

# Jupyter Notebook

- Типы ячеек:
  - Кодовые (в зависимости от ЯП)
  - Текстовые (Markdown)
  - Иное
- Вывод графиков matplotlib  
%matplotlib inline
- Вывод в out значений последней строки (отмена вывода с ;)

```
B [2]: import numpy as np
a = np.array([1, 2, 3])
b = np.array([4, 5, 6])
a + b

Out[2]: array([5, 7, 9])

B [3]: a + b;
```

Код

Код

Markdown

Необработанный NBConvert

Заголовков

Код

Markdown

B [ ]: # python comment  $a^b$

# python comment  $a^b$

▶ Запуск

B [1]: # python comment  $a^b$

python comment  $a^b$

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt

x = np.arange(100)
y = x // 10 + np.random.rand(100)

plt.scatter(x, y)
```

# Magic command JN

- %: применение к строке
- %%: применение к ячейке
- Полный список команд: %lsmagic
- Популярные команды:
  - %env: доступ к переменным окружен
  - %run: выполнение .py или .ipynb
  - %load: загрузка кода в ячейку
  - %time: замер времени выполнения
  - %prun, %lprun, %mprun: профилировка
  - %writefile: запись ячейки в файл
  - Использование других языков: %%R ...
- !: выполнение shell-команд

```
B [4]: %lsmagic
```

```
Out[4]: Available line magics:
```

```
%alias %alias_magic %autoawait %autocall %automagic %autosave %bookmark %cd %clear %cls %colors %conda %config %connect_info %copy %ddir %debug %dhist %dirs %doctest_mode %echo %ed %edit %env %gui %hist %history %killbgscripts %ldir %less %load %load_ext %loadpy %logoff %logon %logstart %logstate %logstop %ls %lsmagic %macro %magic %matp lotlib %mkdir %more %notebook %page %pastebin %pdb %pdef %pdoc %pfile %pinfo %pinfo2 %pip %popd %pprint %precision %prun %psearch %pso
```

```
B [5]: def dummy_sum_for(n):
        res = 0
        for i in range(n):
            for j in range(i):
                res += i**j
        return res
```

```
B [7]: %timeit dummy_sum_for(10)
```

```
25.4 µs ± 109 ns per loop
```

```
B [9]: !pip install line_profiler
        %load_ext line_profiler
```

```
Collecting line_profiler
```

```
Downloading https://files.pythonhosted.org/packages/40/22/24c01f4502d70877af5159113310a3dwin\_amd64.whl (89kB)
```

```
Installing collected packages: line
Successfully installed line-profile
```

```
B [6]: dummy_sum_for(10)
```

```
Out[6]: 50971777
```

```
B [10]: %lprun -f dummy_sum_for dummy_sum_for(10)
```

```
Timer unit: 4.87619e-07 s
```

```
Total time: 7.02171e-05 s
```

```
File: <ipython-input-5-8bbe15e94459>
```

```
Function: dummy_sum_for at line 1
```

Line #	Hits	Time	Per Hit	% Time	Line Contents
1					def dummy_sum_for(n):
2	1	2.0	2.0	1.4	res = 0
3	10	11.0	1.1	7.6	for i in range(n):
4	45	47.0	1.0	32.6	for j in range(i):
5	45	83.0	1.8	57.6	res += i**j
6	1	1.0	1.0	0.7	return res

# Облачные системы выполнения JN

- Google colabotary:

- ☐ colab.research.google.com
- ☐ Система хранения: локальная + Google Drive

```
from google.colab import drive
drive.mount ('/content/drive')
```
- ☐ Тайм-аут простоя - 90 минут, абсолютный - 12 часов

The logo for Google Colab, featuring the word "colab" in a bold, sans-serif font. The "col" is in a lighter yellow-orange color, and "ab" is in a darker orange color.

- Kaggle Kernels:

- ☐ Среда с встроенными наборами данных
- ☐ Тайм-аут простоя - 30 минут, абсолютный - 9 часов

The logo for Kaggle, featuring the word "kaggle" in a bold, sans-serif font. The "k" is in a light blue color, and "aggle" is in a darker blue color.

- Ограниченный доступ к GPU и TPU
- Интеграция с GitHub
- Чистая среда после перезагрузки





# Особенности языка Python

Python 3 – интерпретируемый динамически-типизированный язык программирования.

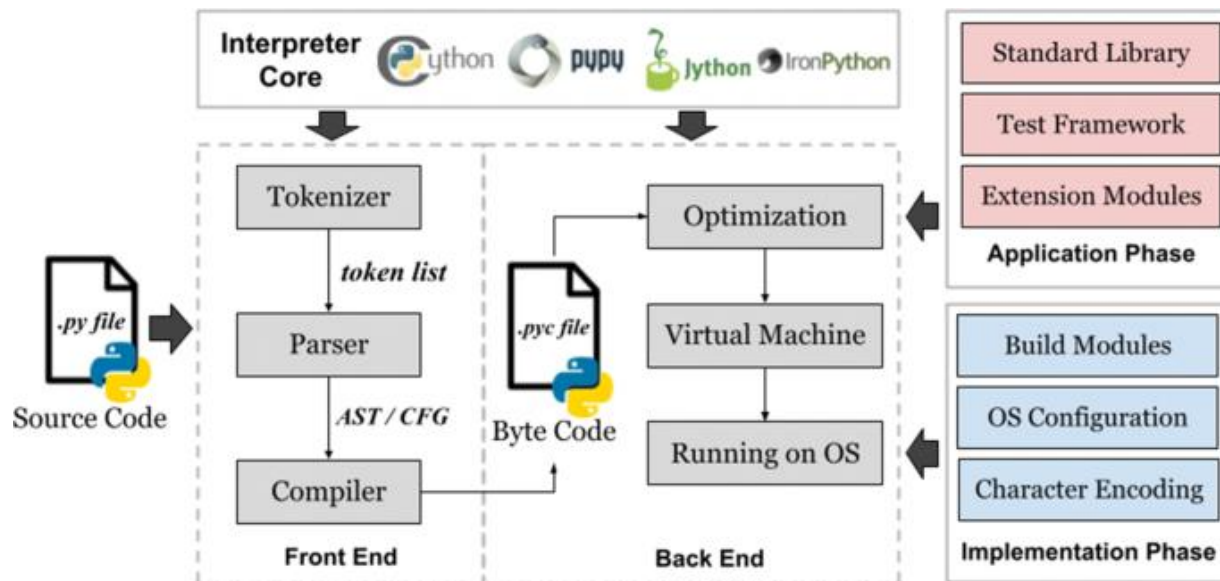
Преимущества:

- Возможность интерактивного программирования;
- Кроссплатформенность;
- Нет необходимости компилировать программу перед выполнением;
- Автоматическое управление памятью.

Недостатки:

- Производительность;
- Для сбора неиспользуемой памяти используется сборщик мусора;
- Динамическая типизация может приводить к ошибкам.

# CPython

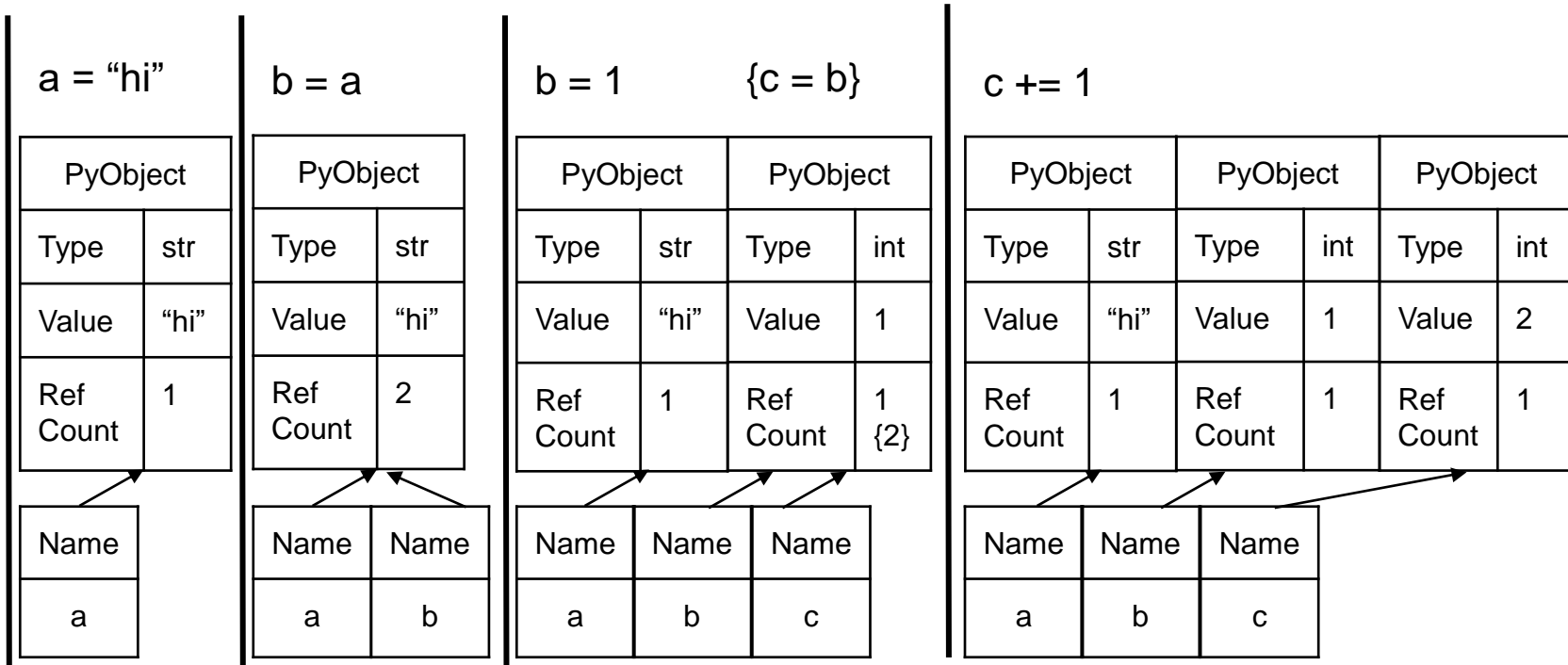


- Интерпретатор + компилятор в байткод
- global interpreter lock (GIL) – сложности при параллельных вычислениях
- интерфейсы для доступа к внешнему функционалу через bindings

# Переменные

## Динамическая типизация

```
a = "hi"  
b = a  
b = 1  
c = b  
c += 1  
  
a, b, c  
  
('hi', 1, 2)
```



# Переменные

## ■ Рекомендация типов

```
a: str = "hi"  
b: int = 1  
c: int = b
```

## ■ Проваерка типов

```
type(a), type(b), type(c)
```

```
(str, int, int)
```

```
print(isinstance(a, str))  
print(isinstance(b, int))  
print(isinstance(c, float))
```

```
True  
True  
False
```

## ■ Сравнение объектов: id(), is

```
a = 1000  
b = 1000  
c = b  
print(id(a), id(b), id(c), sep="\n")
```

```
140176963252720  
140176963253552  
140176963253552
```

```
print(a == b == c)  
print(a is b) # id(a) == id(b)  
print(a is c) # id(a) == id(c)  
print(b is c) # id(b) == id(c)
```

```
True  
False  
False  
True
```

## ■ Object (общий базовый класс)

```
print(isinstance(a, object))  
print(isinstance(b, object))  
print(isinstance(c, object))
```

```
True  
True  
True
```

```
print(isinstance(int, object))  
print(isinstance(type, object))  
print(isinstance(object, object))
```

```
True  
True  
True
```

# Базовые типы данных

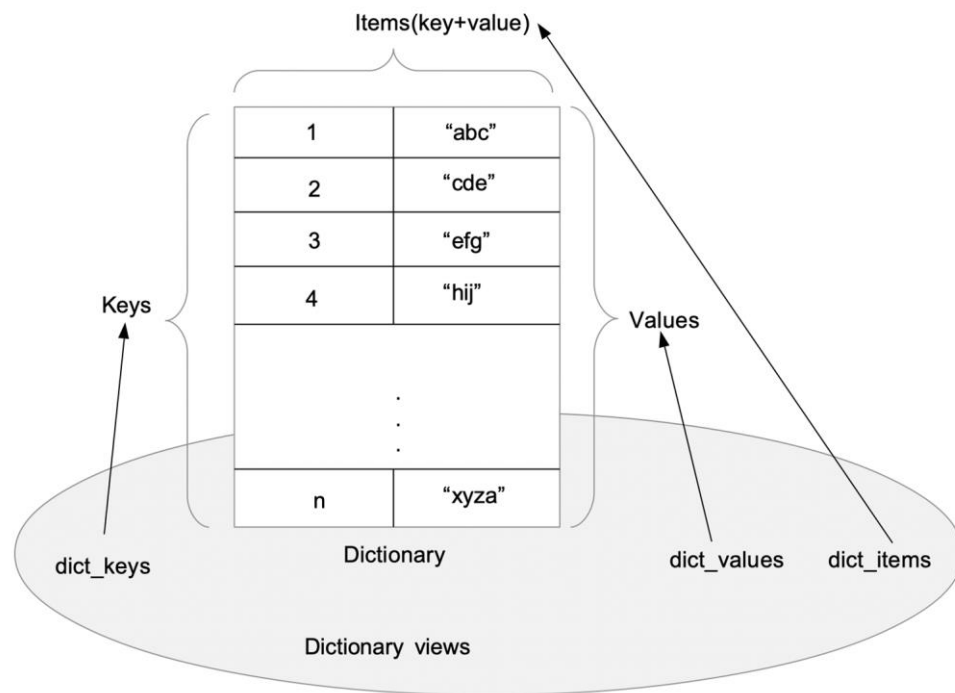
Тип	Числовой	Строковый	Логический
Init	<ul style="list-style-type: none"><li>• int a = 2</li><li>• float b = 2.0</li><li>• complex c = 1 + 2j</li></ul>	<ul style="list-style-type: none"><li>• str s = "hello"</li><li>• f-string sf = f"{a}_{b}" = "2_2.0"</li><li>• unicode su = u"привет"</li></ul>	<ul style="list-style-type: none"><li>• True 4 == 4 bool("hi")</li><li>• False bool("") bool(0)</li></ul>
Op	<ul style="list-style-type: none"><li>• Бинарные +, -, *, / //, %, ** ==, !=, &lt;, &gt;  , ^, &amp;, &gt;&gt;, &lt;&lt;</li><li>• Унарные +=, -=, *=, /=, ~</li></ul>	<p>s = "H" + "i" s * 3 = "HiHiHi"</p> <p>s.upper() = "HI" s.lower() = "hi" s.find("i") = 1 s.find("t") = -1 len(s) = 2</p>	<ul style="list-style-type: none"><li>• bool -&gt; int True + 4 = 5 5 * False = 0</li><li>• int -&gt; bool or, and, not</li><li>• is (object identity)</li><li>• in "gg" in "eggs" = True</li></ul>

# Структуры данных (tuple, list, set)

Тип	Кортеж (tuple) <ul style="list-style-type: none"><li>- разнотипные элементы</li><li>- порядок учтен</li></ul>	Список (list) <ul style="list-style-type: none"><li>- однотипные элементы</li><li>- порядок учтен</li></ul>	Множество (set) <ul style="list-style-type: none"><li>- однотипные элементы</li><li>- порядок не учтен</li></ul>
Init	<ul style="list-style-type: none"><li>• ()</li><li>• tuple(range(10))</li><li>• tuple("hello")</li></ul>	<ul style="list-style-type: none"><li>• []</li><li>• list(range(10))</li><li>• list("hello")</li></ul>	<ul style="list-style-type: none"><li>• {}</li><li>• set(range(10))</li><li>• set("hello")</li></ul>
Op	<ul style="list-style-type: none"><li>• Доступ к элементу a[i]</li><li>• Поиск по значению a.index(i)</li><li>• Срезы a[start:end:step]</li><li>• Агрегаты: len(), sum()</li></ul>	<ul style="list-style-type: none"><li>• Доступ и поиск элемента</li><li>• Срезы</li><li>• List comprehension<ul style="list-style-type: none"><li>[i**2 for i in range(10)]</li><li>[i for i in [1, 2, 3] if i &gt; 1]</li><li>[i if ~i else 2 for i in [0, 2]]</li></ul></li><li>• Агрегаты: len(), sum()</li><li>• Методы изменения<ul style="list-style-type: none"><li>l.sort()</li><li>l.append(val),</li><li>l1.extend(l2)</li><li>l.remove(val), l.pop(ind)</li><li>l.reverse(), .clean()</li></ul></li></ul>	<ul style="list-style-type: none"><li>• Вхождение элемента (in)</li><li>• Set comprehension</li><li>• Агрегаты: len(), sum()</li><li>• Методы изменения<ul style="list-style-type: none"><li>add(), remove(), pop()</li></ul></li><li>• Бинарные операции<ul style="list-style-type: none"><li>s1.union(s2) или  </li><li>s1.intersection(s2) или &amp;</li><li>s1.difference(s2) или -</li></ul></li></ul>

# Структуры данных (dict)

- Инициализация  
`{}`, `{'one': 1, 'two': 2, 'three': 3}`  
`{i,j for i, j in zip([1, 2, 3], [4, 5, 6])}`
- Ключи: `d.keys()`, `list(d)`
- Значения: `d.values()`
- Пары ключ-значение: `d.items()`
- Изменение словаря:  
`d[key] = value` или `d.update(d2)`
- Доступ по ключу:  
`d[key]` или `d.get(key, default)`
- Удаление ключа:  
`del d[key]` или `d.pop(key)`



# Управляющие конструкции

## ■ Циклы:

**while** условие:

    #тело цикла

**else:**

    #выполняется 1 раз если не  
    #было **break**

**for** переменная **in** итератор:

    #тело цикла

**else:**

    #выполняется 1 раз если не  
    #было **break**

**continue** – следующая итерация

## ■ Условия:

**if** условие:

    #операторы

**elif** условие :

    #операторы

...

**else:**

    #операторы

## ■ Тернатрный **if**:

значение **if** условие **else** значение



# Функции

## ■ Определение функции

```
def f(x, y):  
    return x + y  
  
print("int:", f(1, 2))  
print("str:", f("a", "b"))  
print("list:", f([0], [1]))
```

```
int: 3  
str: ab  
list: [0, 1]
```

## ■ lambda функции

```
dist = lambda x, y: (x**2 + y**2)**(0.5)  
  
dist(0, 0) # 0  
dist(1, 0) # 1  
dist(8, 6) # 10
```

## ■ Полезные функции:

- eval – вычисление выражений, print – вывод в том числе форматированный, input – ввод данных

## ■ Передача параметров

```
def power(x=1, y=1):  
    return x**y
```

```
power() # 1  
power(1, 2) # 1  
power(x=2) # 2  
power(y=2, x=2) # 4
```

```
def params_to_tuple(*args):  
    return args
```

```
def params_to_dict(**kwargs):  
    return kwargs
```

```
params_to_tuple(1, 2, 3) # (1, 2, 3)  
params_to_tuple(x=1, y=2) # TypeError: unexpected argument 'x'  
params_to_dict(x=1, y=2) # {'x': 1, 'y': 2}  
params_to_dict(1, 2, 3) # TypeError: 0 positional arguments
```

# Функции map, reduce, filter

## ■ map (func, \*iters)

- Применяет func к каждому iters

```
l1 = range(10)
l2 = range(-10, 0)
```

```
print(list(l1))
print(list(l2))
```

```
print(map(lambda x: x**2, l1))
print(list(map(lambda x: x**2, l1)))
print(map(lambda x, y: x*y, l1, l2))
print(list(map(lambda x, y: x*y, l1, l2)))
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[-10, -9, -8, -7, -6, -5, -4, -3, -2, -1]
<map object at 0x7f257135eb50>
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
<map object at 0x7f257135eb50>
[0, -9, -16, -21, -24, -25, -24, -21, -16, -9]
```

```
s = "1 2 3 4 5"
list(map(int, s.split()))
```

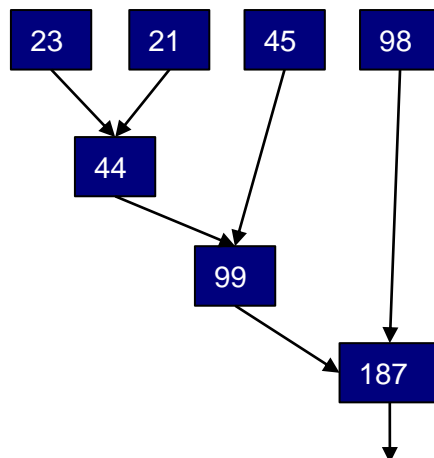
```
[1, 2, 3, 4, 5]
```

## ■ reduce (func, \*iters)

- Применяет func последовательно к паре (старое зн. и каждый iters)

```
l = [23, 21, 45, 98]
reduce(lambda a, b: a + b, l)
```

187



## ■ filter (func, \*iters)

- Применяет логическую func к каждому iters и оставляет только True

```
l = [i**2 for i in range(20)]
```

```
filtered = filter(lambda el: el % 2 == 0, lst)
print(filtered)
print(list(filtered))
```

```
<filter object at 0x7f2583e97950>
[0, 4, 16, 36, 64, 100, 144, 196, 256, 324]
```

```
l1 = ["100,,0,0", "99,,,0", "25,0,,,"]
```

```
all_el = map(lambda l: l.split(","), l1)
print(list(all_el))
```

```
filt_el = map(lambda l: list(filter(len, l.split(","))), l1)
print(list(filt_el))
```

```
[['100', '', '0', '0'], ['99', '', '', '0'], ['25', '0', '', '', '']]
[['100', '0', '0'], ['99', '0'], ['25', '0']]
```

# Неопределенность, функции как генераторы

## ■ None

- «Неопределенность»

```
x = None
y = None
print(id(x), id(y))
print(x is y)
print(x is None)
```

```
94078386057008 94078386057008
True
True
```

## ■ Pass

- «Заглушка»

```
def f1(x, y):
    x + y

def f2(x, y):
    pass

print(f1(1, 2))
print(f2(1, 2))
```

```
None
None
```

## ■ Итераторы vs Генераторы (динамические итераторы пока возврат через yield)

- После return прекращает выполнение функции
- После yield передает управление вызывающей области
- В отличие от списка, в каждый отдельный момент удерживает только возвращаемое значение
- По сравнению с функцией, сохраняет локальные переменные при прекращении работы

```
def tic_tac(x):
    while(x > 0):
        if x % 2 == 0:
            yield "Tic"
        else:
            yield "Tac"
        x -= 1
```

```
for i in tic_tac(6):
    print(i, end=">")
```

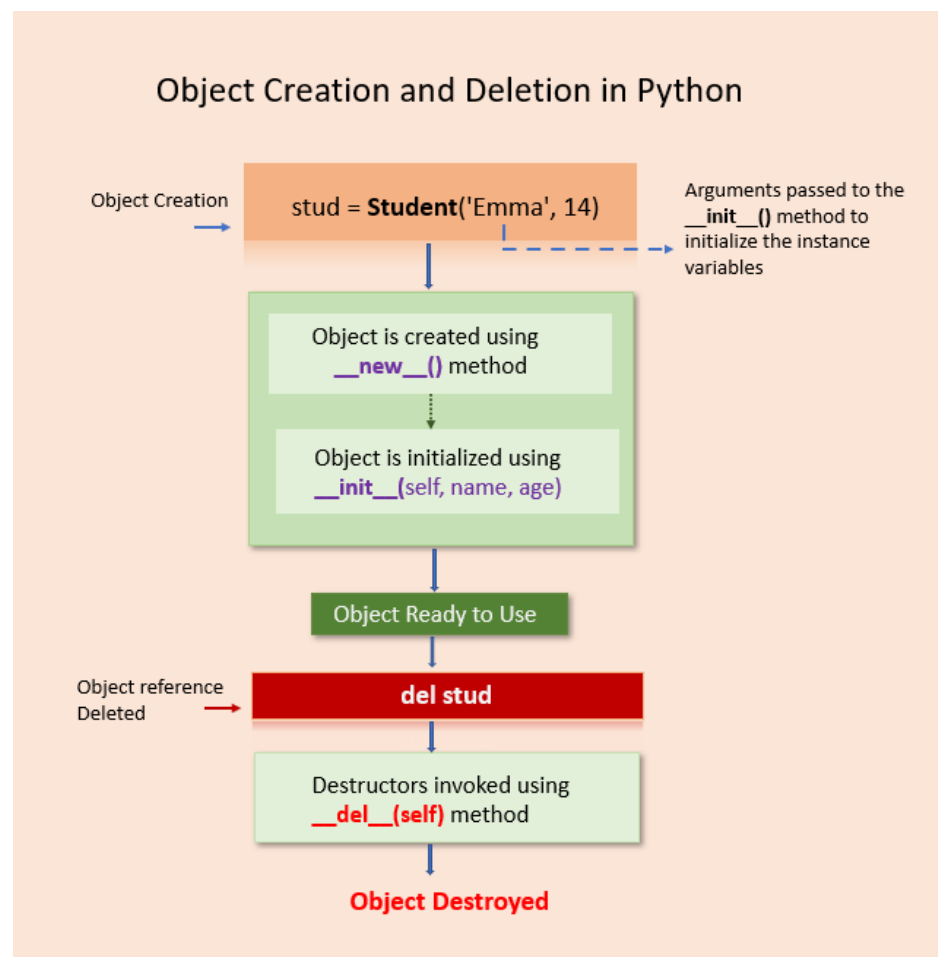
Tic->Tac->Tic->Tac->Tic->Tac->

```
a = (x**2 for x in range(5))
next(a) # 0
next(a) # 1
next(a) # 4
next(a) # 9
next(a) # 16
next(a) # StopIteration
```

# Классы

Определение класса:

- Конструктор класса: `__new__()`
- Инициализация: `__init__(self)`
- Деструктор класса: `__del__(self)`
- Ссылка на свой PyObject: `self`



# Пример класса

Определение класса и методов:

- Инициализация: `__init__(self)`
- Статические атрибуты:  
width, height
- Динамические атрибуты:  
title, sentences
- Переопределение функций
- Неявное наследование от object  
Аналогично **PythonSlide(object)**

```
class PythonSlide:
    width = 1920
    height = 1080

    def __init__(self, title):
        self.title = title
        self.sentences = []

    def add_sentence(self, s):
        self.sentences.append(s)

    def __str__(self):
        return f"{self.title} ({self.width}x{self.height})"

slide = PythonSlide("Классы")
print(slide) # == str(slide)
print(slide.title, slide.sentences)
print(slide.width, slide.height)

print(isinstance(a, PythonSlide))
print(isinstance(a, object))

slide.add_sentence("__init__")
slide.add_sentence("methods")
slide.add_sentence("__str__")
print(slide.sentences)
```

```
Классы (1920x1080)
Классы []
1920 1080
False
True
['__init__', 'methods', '__str__']
```

# Инкапсуляция и наследование

## ■ Изменение атрибутов

```
print(slide.width) # 1920
delattr(PythonSlide, "width")
print(slide.width) # AttributeError: width

slide.width = 1000
print(slide.width) # 1000
```

```
class StoreSecret(object):
    def __init__(self):
        self.__secret = 100

    def get_secret(self):
        return self.__secret
```

```
ss = StoreSecret()
print(ss.get_secret()) # 100
print(ss.__secret) # AttributeError:
print(ss._StoreSecret__secret) # 100
```

## ■ Наследование и super()

```
class A(object):
    def __init__(self):
        print("A init")

class B(A):
    def __init__(self):
        print("B init")
        super().__init__()

b = B()
```

B init  
A init

```
class LogCount(object):
    def __init__(self):
        self.i = 0

    def increase(self):
        self.i += 1

    def print(self):
        self.increase()
        print(f"***{self.i}***")
```

```
class LogWrap(LogCount):
    def increase(self):
        self.i += 2

    def print(self):
        super().print()
        print(f"==={self.i}===")

b = LogWrap()
b.print()

***2***
===2===
```

## ■ Список доступных атрибутов и методов

dir(slide)

```
['_class_',
 '_delattr_',
 '_dict_',
 '_dir_',
 '_doc_',
 '_eq_',
 '_format_',
 ...]
```

object.\_\_dict\_\_

```
mappingproxy({'__repr__': <slot wrapper '__repr__' of 'object' objects>,
 '__hash__': <slot wrapper '__hash__' of 'object' objects>,
 '__str__': <slot wrapper '__str__' of 'object' objects>,
 '__getattr__': <slot wrapper '__getattr__' of 'object' objects>,
 '__setattr__': <slot wrapper '__setattr__' of 'object' objects>,
 '_delattr_': <slot wrapper '_delattr_' of 'object' objects>,
 '_lt_': <slot wrapper '_lt_' of 'object' objects>,
 '_le_': <slot wrapper '_le_' of 'object' objects>,
 ...})
```

# Сериализация объектов

- Импорт сторонних библиотек (joblib)
- Доступ к внешним функциям
- Чтение/запись с файлами
- Сериализация объектов класса
- Сохранение и запись объекта

```
class LogCount(object):  
    def __init__(self):  
        self.i = 0  
  
    def increase(self):  
        self.i += 1  
  
    def print(self):  
        self.increase()  
        print(f"***{self.i}***")
```

```
class LogWrap(LogCount):  
    def increase(self):  
        self.i += 2  
  
    def print(self):  
        super().print()  
        print(f"==={self.i}===")  
  
b = LogWrap()  
b.print()
```

```
***2***  
===2===
```

```
import joblib  
  
joblib.dump(b, open('./file.txt', 'wb'))
```

```
b2 = joblib.load(open('./file.txt', 'rb'))  
b2.print()
```

```
***4***  
===4===
```

# Наследование и специальные ВОЗМОЖНОСТИ

- Приведение типов:
  - определить методы `__str__`, `__bool__`, `__int__` и другие
- Перегрузка операторов:
  - определить методы `__add__`, `__mul__`, `__sub__` и другие
- Множественное наследование:
  - список в скобках в операторе **class**, поиск метода по сигнатуре сначала в объекте, потом в родителях с учетом порядка в списке
- Декораторы:
  - Расширение возможностей функций, а значит и методов классов, а значит и объектов без изменения кода исходной функции

```
def decorator_function(func):  
    def wrapper():  
        print('Функция-обёртка!')  
        print('Оборачиваемая функция: {}'.format(func))  
        print('Выполняем обёрнутую функцию...')  
        func()  
        print('Выходим из обёртки')  
    return wrapper
```

```
@decorator_function  
def hello_world():  
    print('Hello world!')
```

```
Функция-обёртка!  
Оборачиваемая функция: <function hello_world at 0x000002AD640B70D0>  
Выполняем обёрнутую функцию...  
Hello world!  
Выходим из обёртки
```



# Обработка исключений

- Исключение – объект
- «Бросить» исключение **raise**, в том числе свой объект (наследник класса Exception)
- «Поймать» исключение по типам:

**try:**

    #код где ловим

**exception** класс\_исключения\_1:

    #выполняется обработка произошедшего исключения типа 1

...

**exception** класс\_исключения\_K:

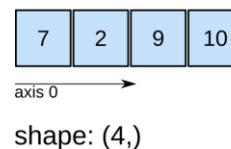
    #выполняется обработка произошедшего исключения типа K

# Numpy (Numerical Python)

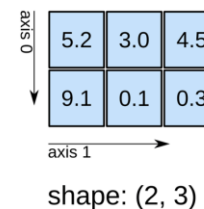
- Открытая библиотека для работы с многомерными массивами
- Используется во многих библиотеках Python: Pandas, SciPy, Scikit-learn
- Ядро написано на Си (код оптимизирован под большинство CPU). Эффективна за счет кэширования и векторизации.
- Установка и импорт библиотеки
  - Установка через pip  
`pip install numpy`
  - Импорт через python  
`import numpy as np`



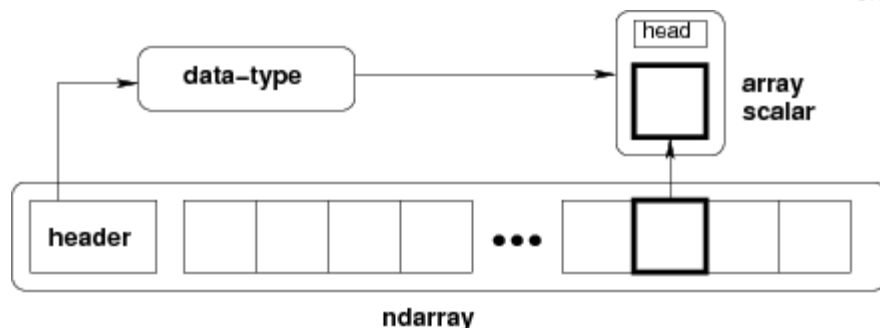
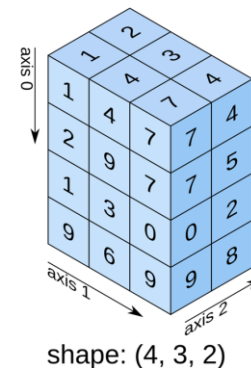
1D array



2D array



3D array



# Numpy ndarray

## ■ Как хранилище одномерных данных

## ■ Создание:

□ Из python-объектов (список, кортеж): `np.array(obj)`

□ Из интервала значений с заданным:

**шагом:** `np.arange(start, end, step)`

**количеством точек:** `np.linspace(start, end, size)`

□ С заданными значениями:

■ **Нулей:** `np.zeros(shape)`, **Единиц:** `np.ones(shape)`

■ **Случайные значения:** `np.empty(shape)`

■ **Кастомные значения:** `np.full(shape, value)`

## ■ Ссылки на объект (для копирования `.copy()`)

```
a = np.array([1, 2])
b = a
b *= 3
a, b
```

(array([3, 6]), array([3, 6]))

```
a = np.array([1, 2])
b = a.copy()
b *= 3
a, b
```

(array([1, 2]), array([3, 6]))

```
lst = [1, 2, 3, 4, 5]
tpl = (1, 2, 3, 4, 5)
a_l = np.array(lst)
a_t = np.array(tpl)
a_l, a_t
```

(array([1, 2, 3, 4, 5]),  
array([1, 2, 3, 4, 5]))

```
r = np.arange(5)
l = np.linspace(0, 4, 5)
r, l
```

(array([0, 1, 2, 3, 4]),  
array([0., 1., 2., 3., 4.]))

```
z = np.zeros((5))
o = np.ones((5))
e = np.empty((5)) # мусор
f = np.full((5), -9999)
print(z, o, e, f, sep = "\n")
```

[0. 0. 0. 0. 0.]  
[1. 1. 1. 1. 1.]  
[1. 1. 1. 1. 1.]  
[-9999 -9999 -9999 -9999 -9999]

# Numpy ndarray

- Форма массива (shape) - кортеж длин осей
- Создание:
  - Из python-объектов: `np.array(obj)`
  - Из начальных и конечных значений с числом точек
  - С заданными значениями
  - Добавление размерности (`np.newaxis` + `np.repeat`)
- Изменение формы:

- `arr.reshape()`
- `arr.ravel()`
- **inplace**  
`arr.resize()`

```
r = np.arange(6)
a = r.reshape(6, 1)
b = r.reshape(3, 2)
c = r.reshape(2, 3)
d = r.reshape(1, 2, 3)
r, a, b, c, d
```

```
np.ravel(a) # == r
np.ravel(b) # == r
np.ravel(c) # == r
np.ravel(d) # == r
```

```
(array([0, 1, 2, 3, 4, 5]),
 array([[0],
        [1],
        [2],
        [3],
        [4],
        [5]]),
 array([[0, 1],
        [2, 3],
        [4, 5]]),
 array([[0, 1, 2],
        [3, 4, 5]]),
 array([[0, 1, 2],
        [3, 4, 5]]))
```

```
lst1 = [0, 1, 2]
lst2 = [3, 4, 5]
lst_lst = [lst1, lst2]
np.array(lst_lst)
```

```
array([[0, 1, 2],
       [3, 4, 5]])
```

```
np.linspace([2,3],[4,6], num=3)
```

```
array([[2. , 3. ],
       [3. , 4.5],
       [4. , 6. ]])
```

```
z = np.zeros((2, 2))
o = np.ones((2, 2))
e = np.empty((2, 2)) # мусор
f = np.full((2, 2), -9999)
```

```
h = np.arange(6)[np.newaxis, :]
r1 = np.repeat(h, 2, axis=0)
h, r1
```

```
(array([[0, 1, 2, 3, 4, 5]]),
 array([[0, 1, 2, 3, 4, 5],
        [0, 1, 2, 3, 4, 5]]))
```

# Типизация

- При создании выбирает общий тип
- Рекомендуется указывать тип через **dtype**:
  - Внутренний тип numpy: int, bool, float
  - Общий формат: “порядок”, “тип”, “размер”  
Например: “<i4” (big-endian byte np.int32)
  - Кастомный тип (object или иное)
- Иные значения float: nan (!=), inf (1/0)
- Все элементы должны иметь один тип
- Изменение типа через arr.astype(new type)

```
a = np.array([2, 3, 4], dtype=int)
print(a)
print(a.astype(float))
print(a.astype(str))
```

```
[2 3 4]
[2. 3. 4.]
['2' '3' '4']
```

```
i = np.array([2, 3, 4]) # int
s = np.array([2, 3, "array"]) # str
o = np.array([2, [123], "array"]) # object
i, s, o
```

```
(array([2, 3, 4]),
 array(['2', '3', 'array'], dtype='<U21'),
 array([2, list([123]), 'array'], dtype=object))
```

Тип	Numpy type	Код
Логический	bool	b
Числовой	byte, short, intc, int8-64, uint8-64	i1-i8, u1-u8
Веществ.	float16-64, double	f, d, f2-f8
Строковый	str, char	s, u
Иное	Кастомный (default: object)	-

# Взаимодействие

## ■ Базовые операции

### □ Доступ к атрибутам

```
a = np.arange(100).reshape((2,5,10))
print(a.shape) # Форма
print(a.ndim)  # Размерность
print(a.dtype) # Тип
print(a.size)  # Количество элементов
```

```
(2, 5, 10)
3
int64
100
```

### □ Доступ к элементу a[i, j, k]

### □ Срезы a[start: end: step]

```
a[0, 0, 0] # Первый элемент
a[0, :, :] # Первая строка
a[:, 0, :] # Первый столбец
a[::-1]    # Симметрия по 0й оси
a[:, ::-1] # Симметрия по 1й оси
```

## ■ Структурированные массивы как набор именованных столбцов

## ■ Хранение данных разных типов

```
dots = np.array([( 'A', 0.5), ( 'B', 0.2),
                  ( 'C', 0.19), ( 'D', 0.11)],
                dtype=[('name', 'U1'), ('val', 'f2')])
dots['name'], dots['val']
```

```
(array([ 'A', 'B', 'C', 'D'], dtype='<U1'),
 array([0.5 , 0.2 , 0.19, 0.11], dtype=float16))
```

```
y = np.empty(dtype=[("value", np.float64),
                    ("measure", '<U2')], # СИ
              shape=4)
y["value"] = np.array([10, 0.2, 5e-4, 40000])
y["measure"] = np.array(["V", "KV", "MV", "mV"])

y["value"], y["measure"]
```

```
(array([1.e+01, 2.e-01, 5.e-04, 4.e+04]),
 array(['V', 'KV', 'MV', 'mV'], dtype='<U2'))
```

# Операции

## ■ Поэлементные

- Унарные: `exp(A)`, `sqrt(A)` ...
- Бинарные: `+`, `-`, `*`, `/`

## ■ Матричные

- Агрегаты: `A.max(axis)`, `A.mean(axis)` ...
- Унарные: `A.T`, `trace(A)`
- Бинарные: `A.dot(B)`

```
A = np.linspace(10, 100, 4).reshape((2,2))
B = np.ones((2,2), dtype=int)

print("sqrt(A):\n", np.sqrt(A))

print("max:", A.max())
print("max by row:", A.max(axis=1))

print("Elementwise sum:\n", A + B)
print("Elementwise prod:\n", A * B)
print("Matrix prod:\n", A.dot(B)) # or A @ B
```

```
sqrt(A):
[[ 3.16227766  6.32455532]
 [ 8.36660027 10.        ]]
max: 100.0
max by row: [ 40. 100.]
Elementwise sum:
[[ 11.  41.]
 [ 71. 101.]]
Elementwise prod:
[[ 10.  40.]
 [ 70. 100.]]
Matrix prod:
[[ 50.  50.]
 [170. 170.]]
```

## ■ Изменение формы:

- Фильтрация: `a[cond]`
- Объединение: `vstack()`, `hstack()`, ...
- Разделение: `vsplit()`, `hsplit()`, ...

```
A = np.arange(5)
B = np.zeros(5, dtype=int)
ind = np.arange(0, 5, 2)
```

```
B[ind] = 1
print("A:", A)
print("B:", B)
print("A[B]:", A[B])
print("A[A < 3]:", A[A < 3])
```

```
C = np.hstack([A, B])
D = np.vstack([A, B])
print("AB:", C)
print("[A,B]:\n", D)
```

```
print("AB -> A, B:\n", np.hsplit(C, 2))
print("[A,B] -> A, B:\n", np.vsplit(D, 2))
```

```
A: [0 1 2 3 4]
B: [1 0 1 0 1]
A[B]: [1 0 1 0 1]
A[A < 3]: [0 1 2]
AB: [0 1 2 3 4 1 0 1 0 1]
[A,B]:
[[0 1 2 3 4]
 [1 0 1 0 1]]
AB -> A, B:
[array([0, 1, 2, 3, 4]),
 array([1, 0, 1, 0, 1])]
[A,B] -> A, B:
[array([[0, 1, 2, 3, 4]]),
 array([[1, 0, 1, 0, 1]])]
```

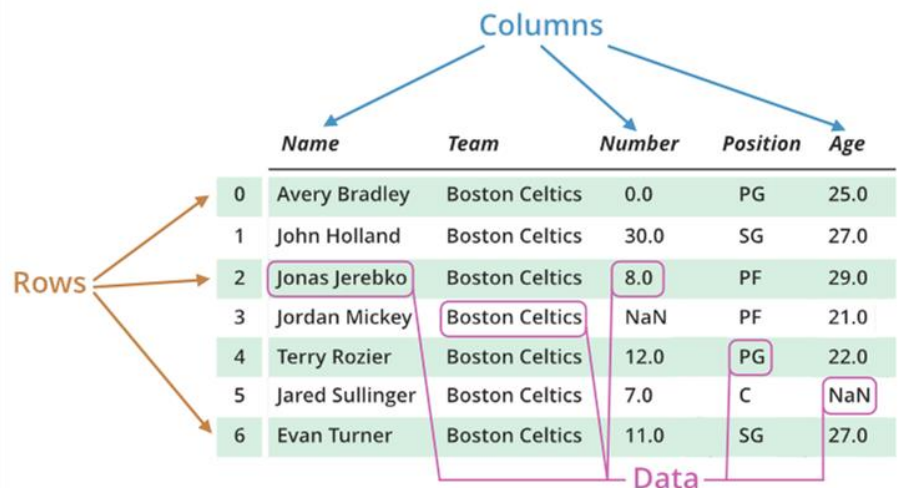
# Pandas



- Открытая библиотека для хранения и обработки табличных данных
- Основные структуры данных:
  - Series - близка к словарям и спискам Python, используется в качестве столбцов в таблице
  - DataFrame - интерфейс близок к обычным таблицам БД. Поддерживает хранение данных разных типов.
- Установка и импорт библиотеки
  - Установка через pip

```
pip install pandas
```
  - Импорт через python

```
import pandas as pd
```



The diagram illustrates the structure of a DataFrame. A tree diagram at the top shows "Columns" branching into "Name", "Team", "Number", "Position", and "Age". Below this is a table with 7 rows and 5 columns. The first column is labeled "Rows" with arrows pointing to each row. The table contains the following data:

	Name	Team	Number	Position	Age
0	Avery Bradley	Boston Celtics	0.0	PG	25.0
1	John Holland	Boston Celtics	30.0	SG	27.0
2	Jonas Jerebko	Boston Celtics	8.0	PF	29.0
3	Jordan Mickey	Boston Celtics	NaN	PF	21.0
4	Terry Rozier	Boston Celtics	12.0	PG	22.0
5	Jared Sullinger	Boston Celtics	7.0	C	NaN
6	Evan Turner	Boston Celtics	11.0	SG	27.0

A pink box labeled "Data" at the bottom right encompasses the data cells of the table, specifically highlighting the values 8.0, NaN, 12.0, 7.0, and 11.0 in the "Number" column, and PG, C, and NaN in the "Position" column.



# Pandas Series

- Одномерный размеченный массив
- Создание Series(data, index=index):
  - Из python-объектов (список, словарь)
  - Из numpy array
- Операции с значениями повторяют numpy
- Операции с индексами:
  - Типы индексов: int, float, str, datetime
  - Доступ по ключу или списку
  - Проверка вхождения индекса
  - Добавление значений по индексу
- Атрибуты:
  - Доступ к данным: s.index, s.array, s.values
  - Свойства: s.dtype, s.shape, s.ndim, s.hasnans

```
d = {"k1": 1, "k2": 0, "k3": 2}
print(pd.Series(d))

print(pd.Series(np.random.randn(4),
               index=["a", "b", "c", "d"]))
```

```
k1    1
k2    0
k3    2
dtype: int64

a    0.685094
b    1.491304
c   -0.555919
d   -0.367133
dtype: float64
```

```
s = pd.Series(np.arange(1, 6))

print(s[0])
print(s[:3])
print(np.exp(s))
```

```
1
0    1
1    2
2    3
dtype: int64

0    2.718282
1    7.389056
2   20.085537
3   54.598150
4  148.413159
dtype: float64
```

```
s = pd.Series({"a1": 1, "a2": 2})
print("a1:", s["a1"])
s["a3"] = 3
print("a3:", s["a3"])

print(f'by index [a3,a2]:\n{s[["a3", "a2"]]}')
print("a3 in s:", "a3" in s)
print("a4 in s:", "a4" in s)
```

```
a1: 1
a3: 3

by index [a3,a2]:
a3    3
a2    2

dtype: int64
a3 in s: True
a4 in s: False
```

# Pandas Dataframe

- Набор Series разных типов (SQL таблица)
- Создание DataFrame(data, index=index):
  - Из python-объектов (словарь, набор словарей)
  - Из 2D numpy ndarray
  - Из pandas Series
- Атрибуты из Series + columns
- Доступ к строкам и столбцам:
  - Добавление столбца: df[name] = vals
  - Удаление: del df[name] или df.drop(name, axis=1)
  - Доступ по имени столбца: df[column\_name]
  - Через имена: df.loc[[row\_names], [column\_names]]
  - Через номера: df.iloc[[row\_n], [column\_n]]
  - При одном столбце преобразуется к Series

```
d1 = {"width": [30, 50, 10],  
      "height": [200, 170, 180]}  
df = pd.DataFrame(d1, index=["A", "B", "C"])
```

```
d2 = [{"a": 1, "b": 2},  
      {"a": -1, "b": 10, "c": 20}]  
pd.DataFrame(d2)
```

```
arr = np.arange(9).reshape((3, 3))  
pd.DataFrame(arr)
```

```
df.loc[["A", "B"], "height"]  
df.iloc[[0, 1], 1]
```

```
df["tall"] = df["height"] >= 180  
df["wide"] = df["width"] > 30  
wide = df.pop("wide")  
del df["tall"] # Equal first df
```

	width	height
A	30	200
B	50	170
C	10	180

	a	b	c
0	1	2	NaN
1	-1	10	20.0

	0	1	2
0	0	1	2
1	3	4	5
2	6	7	8

```
A    200  
B    170  
Name: height,  
dtype: int64
```

	width	height	tall	wide
A	30	200	True	False
B	50	170	False	True
C	10	180	True	False

# Базовые операции

- Вывод первых строк: `df.head(n)`
- Сортировка: `df.sort_values(col)`
- Отмена копирования: **inplace**
- Фильтрация:
  - Выбор строк и столбцов
  - Одиночное условие: `df[col] <op> value`
  - Фильтрация пропусков: `.isna()`, `.notna()`
  - Вхождение в интервал: `.isin(...)`
  - Объединение условий: `&`, `|`
- Написание строковых запросов:
  - Метод `.query(cond)`
  - Вместо `df[col]` указывается только имя
  - `isin` -> `in`

```
arr = np.random.randint(10, size=90)
df = pd.DataFrame(arr.reshape((30, 3)),
                  columns=["x", "y", "z"])
df.head(3)
```

	x	y	z
0	9	0	9
1	4	1	8
2	6	0	1

```
df.sort_values("y", inplace=True)
df.head(3)
```

	x	y	z
20	7	0	5
27	6	0	8
5	4	0	8

```
df[df["x"] > 3].head(3)
df[df["y"].isin([1, 2])].head(3)
```

	x	y	z
0	9	0	9
1	4	1	8
2	6	0	1

	x	y	z
1	4	1	8
4	8	2	0
9	3	2	2

```
df[(df["x"] > 3) & (df["y"].isin([1, 2]))]
```

```
df.query("(x > 3) & (y in [1, 2])")
```

	x	y	z
1	4	1	8
4	8	2	0
19	7	2	0

# Операции со значениями

- Приведение типов: `df[col].astype()`

```
arr = np.random.randn(8) # random float
df = pd.DataFrame(arr.reshape((2, 4)),
                  columns=["i", "n", "f", "o"])
df["i"] = df["i"].astype(int)
```

	i	n	f	o		i
0	0.202126	-0.105390	0.435208	1.240217	→	0
1	1.166177	-0.511592	-0.642082	-1.305375		1

- Математические операции

```
# Базовые: +, -, *, /, сравнения
df["i+n"] = df["i"] + df["n"]
df["f>=o"] = df["f"] >= df["o"]
# Совместимость с функциями numpy
df["exp(n)"] = np.exp(df["n"])
# Агрегаты: .min(), .max(), .sum()
df["all_sum"] = df.sum(axis=1)
```

	i+n	f>=o	exp(n)	all_sum
0	-0.105390	False	0.899974	2.904372
1	0.488408	True	0.599540	1.467056

- Применение func: `apply(func, axis)`  
by row (axis=0), by col (axis=1)

- Маппинг: `.map(dict)`

```
df["f>=o"].map({False:"no", True:"yes"})
```

0 no  
1 yes  
Name: f>=o,  
dtype: object

- Сэмплинг: `df.sample(n)`

```
df.sample(n=3, replace=True)
```

	i	n	f	o	f*o	f>=o	exp(n)	i+n	all_sum
1	1	-0.511592	-0.642082	-1.305375	0.838158	True	0.599540	0.488408	1.467056
0	0	-0.105390	0.435208	1.240217	0.539752	False	0.899974	-0.105390	2.904372
1	1	-0.511592	-0.642082	-1.305375	0.838158	True	0.599540	0.488408	1.467056

# Операции с таблицами

## ■ UNION: `pd.concat([dfs..])`

```
A = pd.DataFrame([{"a1": 0, "a2": 0},  
                  {"a1": 10, "a2": 20}])  
B = pd.DataFrame([{"b1": 1, "b2": 1},  
                  {"b1": 11, "b2": 21}])
```

	a1	a2	b1	b2
0	0	0	0	1
1	10	20	1	11

### □ Горизонтальная: `axis=1`

```
pd.concat([A, B], axis=1)
```

	a1	a2	b1	b2
0	0	0	0	1
1	10	20	11	21

### □ Вертикальная: `axis=0`

```
pd.concat([A, B], axis=0)
```

	a1	a2	b1	b2
0	0.0	0.0	NaN	NaN
1	10.0	20.0	NaN	NaN
0	NaN	NaN	1.0	1.0
1	NaN	NaN	11.0	21.0

## ■ JOIN: `pd.merge(df1, df2, on, how)`

### □ LEFT OUTER JOIN

```
pd.merge(A, B, left_on="a1",  
         right_on="b1", how="left")
```

	a1	a2	b1	b2
0	0	0	NaN	NaN
1	10	20	NaN	NaN

### □ RIGHT JOIN

```
pd.merge(A, B, left_on="a1",  
         right_on="b1", how="right")
```

	a1	a2	b1	b2
0	NaN	NaN	1	1
1	NaN	NaN	11	21

### □ INNER JOIN

```
pd.merge(A, B, left_on="a1", right_on="b1"
```

	a1	a2	b1	b2
--	----	----	----	----

### □ FULL JOIN

```
pd.merge(A, B, left_on="a1",  
         right_on="b1", how="outer")
```

	a1	a2	b1	b2
0	0.0	0.0	NaN	NaN
1	10.0	20.0	NaN	NaN
2	NaN	NaN	1.0	1.0
3	NaN	NaN	11.0	21.0

# Разведочный анализ данных

- Импорт и экспорт данных:
  - Чтение: `read_csv`, `read_excel`, `read_json`
  - Запись: `to_csv`, `to_excel`, `to_json`
- Набор данных по ирисам Фишера
- Базовая информация: `head`, `columns`
- Подробная информация:
  - Статистика (`count`, `mean`, `std`..): `df.describe()`
  - Размер, тип по каждому признаку: `df.info()`

```
iris = pd.read_csv("Iris.csv")
print(iris.columns)
iris.head()
```

```
Index(['Id', 'SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm',
      'PetalWidthCm', 'Species'],
      dtype='object')
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

```
iris.describe()
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	75.500000	5.843333	3.054000	3.758667	1.198667
std	43.445368	0.828066	0.433594	1.764420	0.763161
min	1.000000	4.300000	2.000000	1.000000	0.100000
25%	38.250000	5.100000	2.800000	1.600000	0.300000
50%	75.500000	5.800000	3.000000	4.350000	1.300000
75%	112.750000	6.400000	3.300000	5.100000	1.800000
max	150.000000	7.900000	4.400000	6.900000	2.500000

```
iris.info()
```

```
Data columns (total 6 columns):
#   Column          Non-Null Count  Dtype
---  -
0   Id              150 non-null    int64
1   SepalLengthCm   150 non-null    float64
2   SepalWidthCm    150 non-null    float64
3   PetalLengthCm   150 non-null    float64
4   PetalWidthCm    150 non-null    float64
5   Species         150 non-null    object
dtypes: float64(4), int64(1), object(1)
memory usage: 7.2+ KB
```

# Разведочный анализ данных

- Дополнительная информация:
  - Флаг повторяющихся строк: `df.duplicated()`
  - Количество уникальных: `df.nunique()`
  - Удаление пропусков: `df.dropna()`
  - Таблица частотности: `value_counts()`
- Удаление пропусков: `df.dropna()`
- Анализ групп содержит шаги:
  - Разделение данных на группы по критерию:  
Метод `df.groupby()`
  - Применение функции к каждой группе:  
Встроенные методы или `df.apply(func)`
  - Объединение результатов:  
Метод `grouped_df.agg(funcs)`

```
iris.nunique()
```

```
Id          150
SepalLengthCm  35
SepalWidthCm  23
PetalLengthCm  43
PetalWidthCm  22
Species       3
dtype: int64
```

```
iris["Species"].value_counts()
```

```
Iris-setosa      50
Iris-versicolor  50
Iris-virginica   50
Name: Species, dtype: int64
```

```
grouped = iris.groupby("Species")
grouped.mean()
```

Species				
	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
Iris-setosa	5.006	3.418	1.464	0.244
Iris-versicolor	5.936	2.770	4.260	1.326
Iris-virginica	6.588	2.974	5.552	2.026

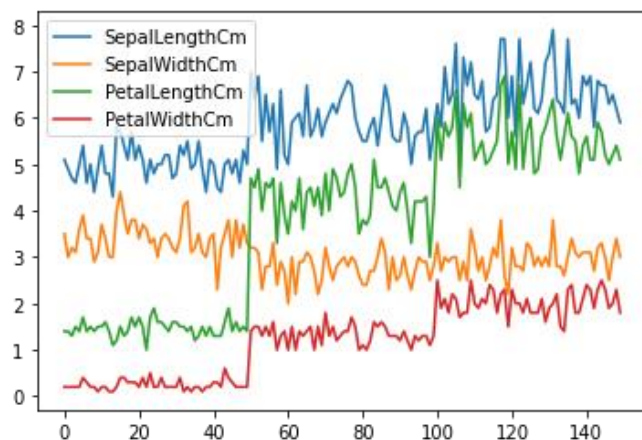
```
grouped.agg([np.mean, np.std])
```

Species								
	SepalLengthCm		SepalWidthCm		PetalLengthCm		PetalWidthCm	
	mean	std	mean	std	mean	std	mean	std
Iris-setosa	5.006	0.352490	3.418	0.381024	1.464	0.173511	0.244	0.107210
Iris-versicolor	5.936	0.516171	2.770	0.313798	4.260	0.469911	1.326	0.197753
Iris-virginica	6.588	0.635880	2.974	0.322497	5.552	0.551895	2.026	0.274650

# Разведочный анализ данных

- Расчет корреляций: `df.corr(method)`  
Method: pearson, kendall, spearman
- Ящик с усами: `.boxplot()`
- Визуализация набора: `.plot()`

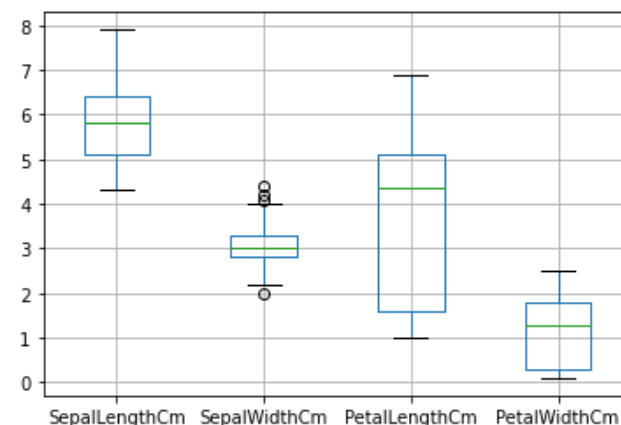
```
iris.loc[:,cols].plot()
```



```
iris.corr()
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
Id	1.000000	0.716676	-0.397729	0.882747	0.899759
SepalLengthCm	0.716676	1.000000	-0.109369	0.871754	0.817954
SepalWidthCm	-0.397729	-0.109369	1.000000	-0.420516	-0.356544
PetalLengthCm	0.882747	0.871754	-0.420516	1.000000	0.962757
PetalWidthCm	0.899759	0.817954	-0.356544	0.962757	1.000000

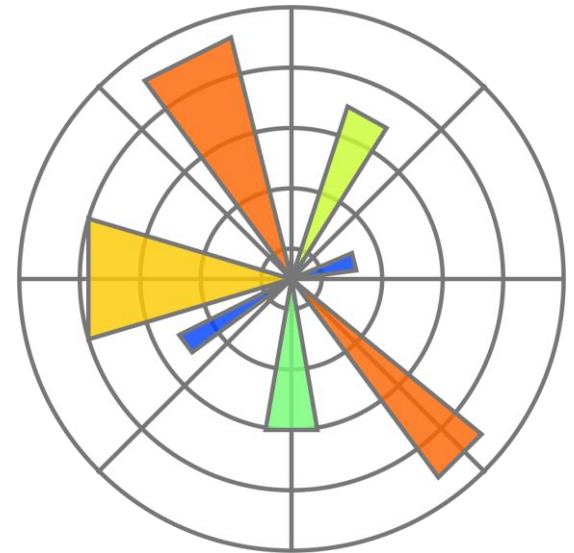
```
cols = ["SepalLengthCm", "SepalWidthCm",  
        "PetalLengthCm", "PetalWidthCm"]  
stud_bplt = iris.loc[:,cols].boxplot()  
stud_bplt.plot()
```





# Визуализация с matplotlib

- Мощная низкоуровневая библиотека для создания статических, анимированных и интерактивных визуализаций на Python
- Pyplot
  - Графический модуль на основе MATLAB
  - Использует интерфейс на основе состояний и ООП
- Установка и импорт библиотеки
  - Установка через pip  
`pip install matplotlib`
  - Импорт через python  
`import matplotlib.pyplot as plt`



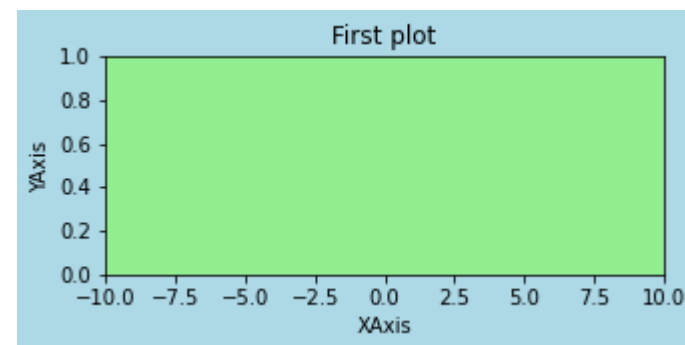
# ОСНОВНЫЕ КОМПОНЕНТЫ

- Figure - общая область рисунка:
  - Конструктор **plt.figure()** определяет холст (длина по x, y; цвет; разрешение)
  - Содержит контейнеры Axes.
- Axes - область отображения графиков:
  - Определяет атрибуты: XAxis, YAxis, title...
  - Для добавления axes (или subplot):  
**fig.add\_subplot([nrow][ncol][num\_cell])**
- Создание графиков императивное:
  - Создание регистрируется в pyplot
  - Для сохранения в файл **plt.savefig(path)**
  - Для отображения **plt.show()**, но график удаляется из памяти pyplot

```
fig = plt.figure(figsize=(5, 2))
ax = fig.add_subplot(111)

fig.set(facecolor = 'lightblue')
ax.set(facecolor = 'lightgreen')
ax.set_xlim([-10, 10])
ax.set_title('First plot')
ax.set_xlabel('XAxis')
ax.set_ylabel('YAxis')

plt.show()
```



# Отображение данных (axes)

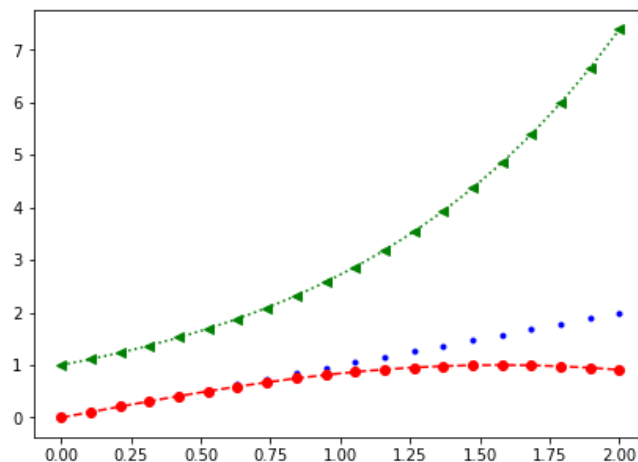
- Изображение линий `plot(x, y, color, marker)`:
  - С одним `y` определяет `x` через `range(N)`
  - Для пар `x, y` определяет точки вида **marker** ('.' - точка; 'o' - круг; 'v', '^', '<', '>' - треугольники; ...)
  - Соединяет точки линиями **linestyle** ('-' - solid; '--' - dashed; ':' - dotted, '' - empty) с шириной **linewidth**
  - Цвет по полному названию или первой букве
- Можно использовать один параметр формат `'[marker][line][color]'`
- Работа с индексируемыми данными: `plot('xlabel', 'ylabel', data=obj)`

```
fig = plt.figure(figsize=(7, 5))
ax = fig.add_subplot(111)

x = np.linspace(0, 2, 20)

ax.plot(x, x, marker=".", linestyle="", color="b")
ax.plot(x, np.sin(x), "o--r")
ax.plot(x, np.exp(x), "<:g")

plt.show()
```



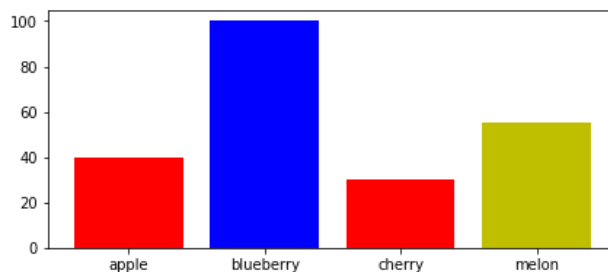
# Отображение данных (axes)

- Изображение точек scatter (x, y, s, c):
  - Близок к графику plot с **linestyle** = ""
  - Аналогичен по параметрам x, y, marker, linewidths
  - **S** - размер маркера (массив или константа)
  - **C** - цвет маркера (массив или константа)
- Изображение столбцов bar (x, y, h, w), где h (height), w (width) - высота и ширина столбцов (массив или константа)

```
fig = plt.figure(figsize=(7, 3))
ax = fig.add_subplot(111)

fruits = ['apple', 'blueberry', 'cherry', 'melon']
counts = [40, 100, 30, 55]
bar_colors = ['r', 'b', 'r', 'y']
ax.bar(fruits, counts, color=bar_colors)

plt.show()
```



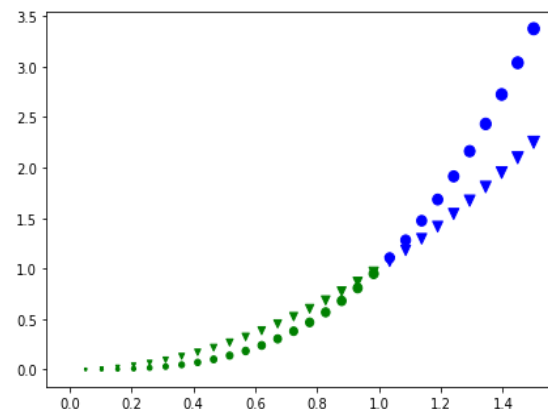
```
fig = plt.figure(figsize=(7, 5))
ax = fig.add_subplot(111)

x = np.linspace(0, 1.5, 30)
size = x * 50

y1 = x**2
y2 = x**3
col = np.where(y1 < y2, "b", "g")

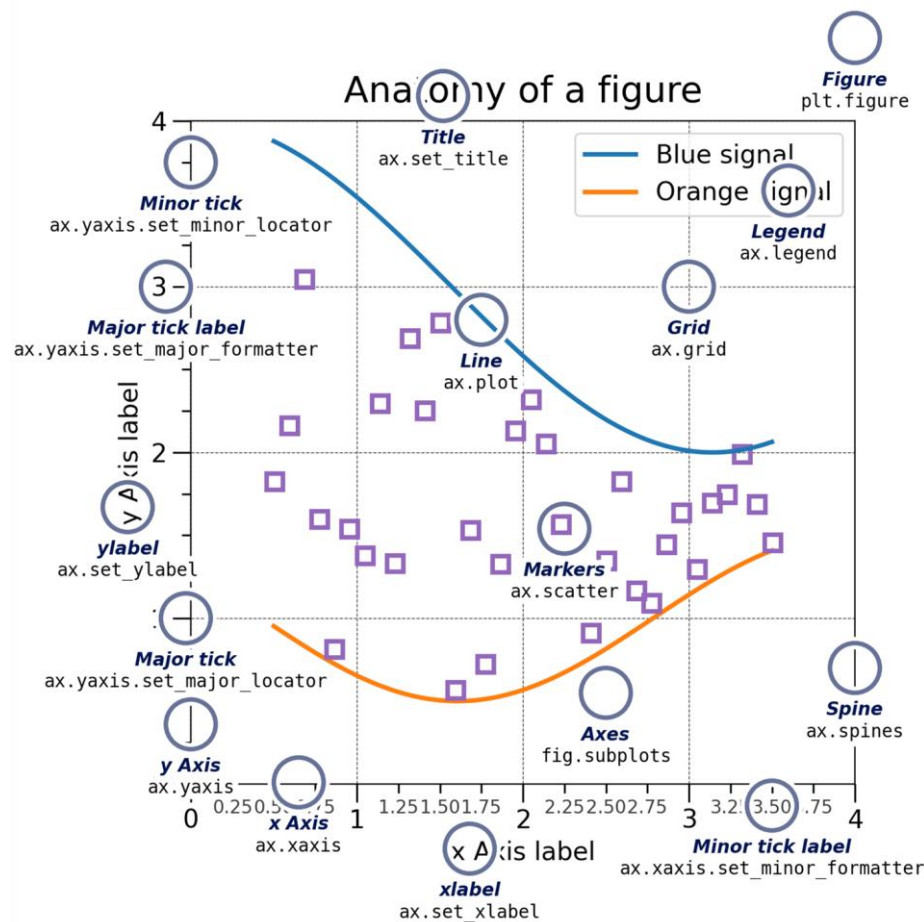
ax.scatter(x, y1, s=size, c=col, marker="v")
ax.scatter(x, y2, s=size, c=col, marker="o")

plt.show()
```



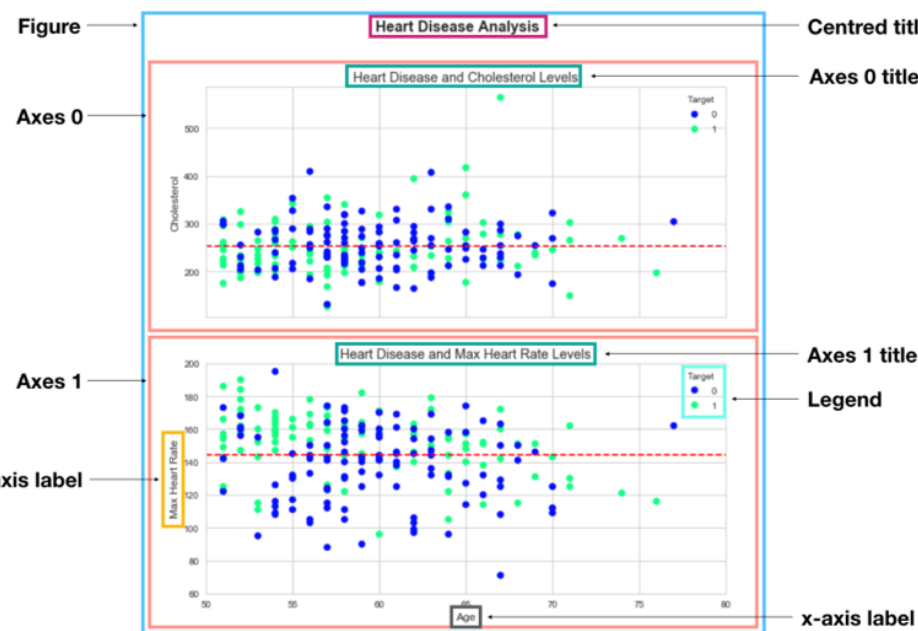
# Кастомизация графиков

- Общие параметры:
  - Title - заголовок графика
  - Grid - сетка с шагом
  - Legend - окно с сопоставлением меток (при установке **label** в графиках)
  - alpha - прозрачность (0 - бесцветный)
- Параметры осей:
  - xlabel, ylabel - метки осей
  - xticks, yticks - деления (числа и строки)
  - xlim, ylim - границы делений
  - xscale, yscale - масштабирование осей
  - xmargin, ymargin - отступ от краев



# Группировка add\_subplot

- В контейнер figure можно поместить несколько axes
- Первый способ:
  - Создать объект figure
  - Создать axes с помощью `fig.add_subplot([nrow][ncol][num_cell])`
  - Полотно разбивается на ячейки согласно `[nrow][ncol]` и выделяет место для axes в ячейке `[num_cell]`
  - Индексирование ячеек построчно слева-направо (1 - левый верх)
- Разбиения могут не совпадать



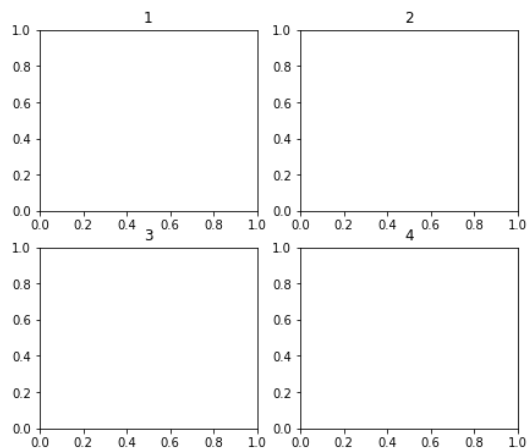
# Примеры add\_subplot

```
fig = plt.figure(figsize=(7,6))

ax_1 = fig.add_subplot(2, 2, 1)
ax_2 = fig.add_subplot(2, 2, 2)
ax_3 = fig.add_subplot(2, 2, 3)
ax_4 = fig.add_subplot(2, 2, 4)

ax_1.set(title = '1')
ax_2.set(title = '2')
ax_3.set(title = '3')
ax_4.set(title = '4')

plt.show()
```

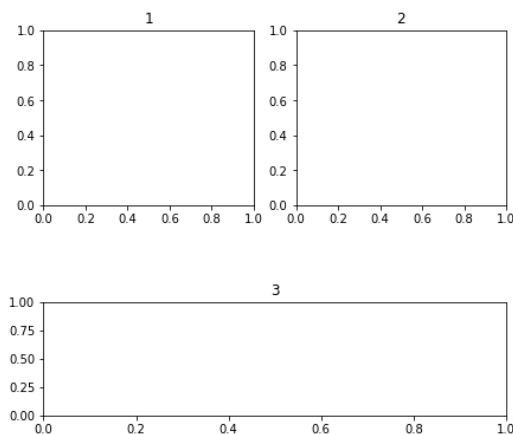


```
fig = plt.figure(figsize=(7,6))

ax_1 = fig.add_subplot(2, 2, 1)
ax_2 = fig.add_subplot(2, 2, 2)
ax_3 = fig.add_subplot(3, 1, 3)

ax_1.set(title = '1')
ax_2.set(title = '2')
ax_3.set(title = '3')

plt.show()
```

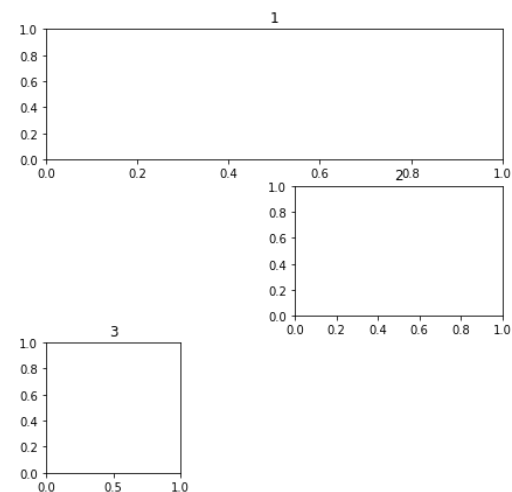


```
fig = plt.figure(figsize=(7, 7))

ax_1 = fig.add_subplot(3, 1, 1)
ax_2 = fig.add_subplot(3, 2, 4)
ax_3 = fig.add_subplot(3, 3, 7)

ax_1.set(title = '1')
ax_2.set(title = '2')
ax_3.set(title = '3')

plt.show()
```



# Группировка subplots

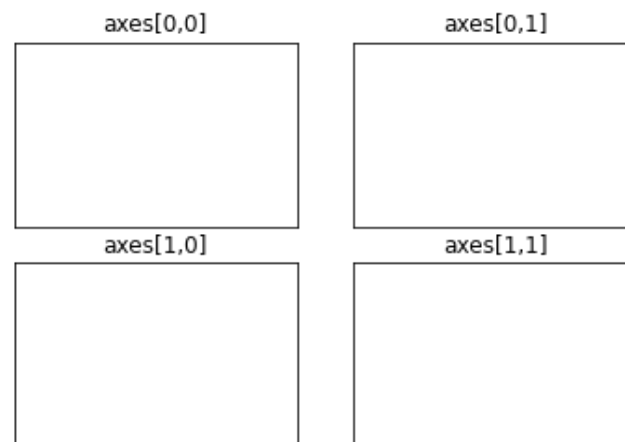
- Второй способ: subplots
- Возвращает кортеж figure и axes по фиксированной сетке
- Индексация по numpy массиву (где axes [0,0] - левый верхний угол)
- Можно перебирать элементы через итератор numpy flat
- По умолчанию создает 1 axes

```
fig, axes = plt.subplots(nrows=2, ncols=2)

axes[0,0].set(title='axes[0,0]')
axes[0,1].set(title='axes[0,1]')
axes[1,0].set(title='axes[1,0]')
axes[1,1].set(title='axes[1,1]')

for ax in axes.flat: # numpy итератор
    ax.set(xticks=[], yticks=[])

plt.show()
```

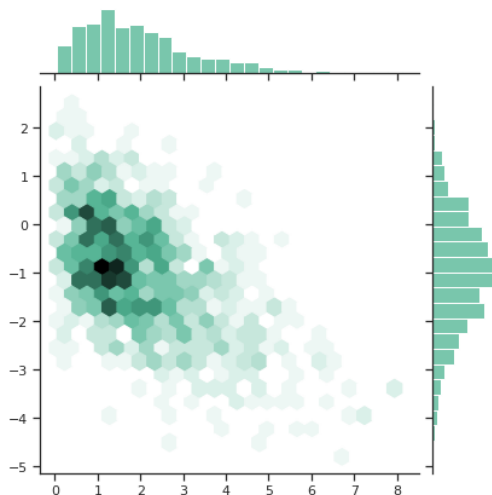




# Расширенные возможности matplotlib

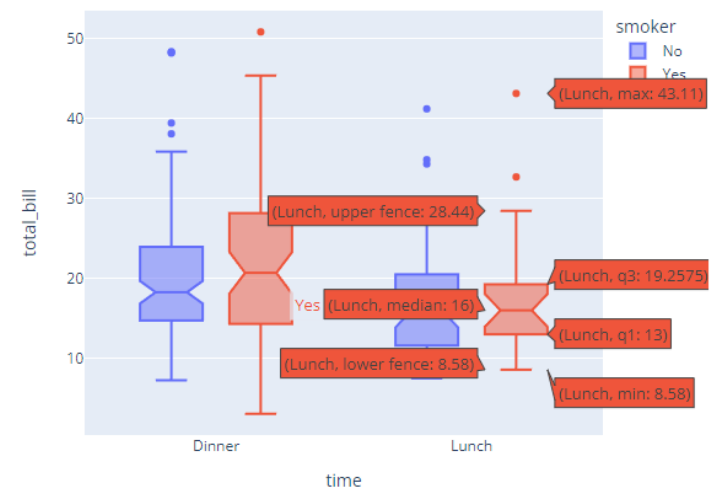
## ■ Seaborn:

- Абстракция для упрощения графиков
- Работает вместе с matplotlib
- Содержит множество стилей
- Тесно взаимодействует с pandas
- Включает наборы для тренировки



## ■ Plotly:

- Front-End на JS
- Back-End на Python (на основе Seaborn)
- Возможность изменять графики “на лету”, построение интерактивных отчетов (dashboard)



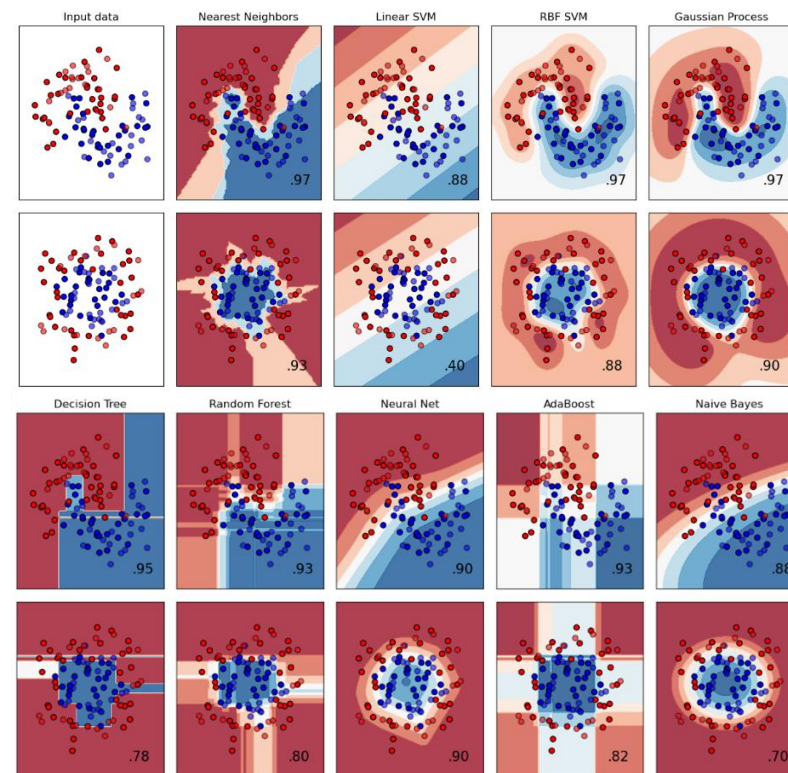
# Scikit-learn (sklearn)



- Широкая библиотека машинного обучения
- Решение различных задач анализа данных с помощью общего API
- В основном написан на Python + Cython
- Хорошая интеграция с библиотеками Numpy, Pandas, Matplotlib, Plotly
- Установка и импорт библиотеки
  - Установка через pip

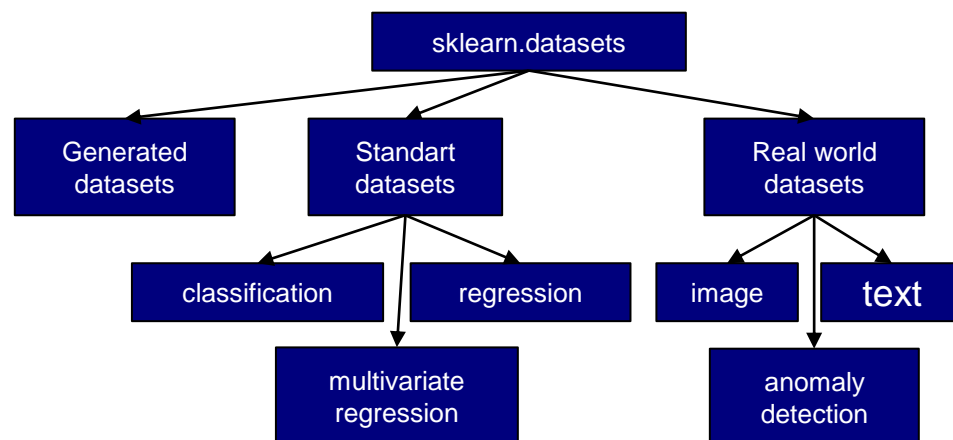
```
pip install scikit-learn
```
  - Импорт через python

```
from sklearn import ...
```



# sklearn.datasets

- Доступ к наборам через API
- Стандартные наборы данных:
  - Небольшой размер (встроены в sklearn)
  - Загрузка: `load_<dataset_name>`
- Внешние наборы данных:
  - Большой размер (выгрузка из репозитория)
  - Загрузка: `fetch_<dataset_name>`
- Возвращают словарь-like с атрибутами: `data`, `target`, `DESCR`, ...
- Некоторые наборы имеют параметр `return_X_y` (возвращает кортеж (`data`, `target`))



```
from sklearn.datasets import load_iris

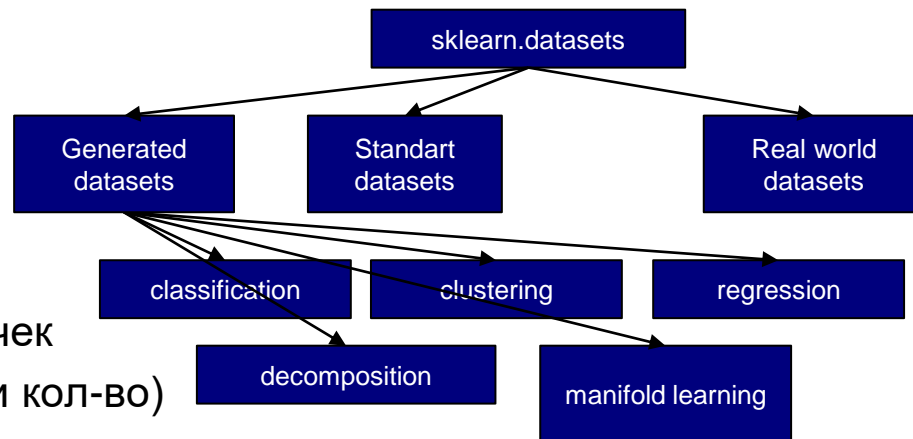
iris = load_iris()
X = iris.data
print("Матрица данных:\n", iris.data[:5])
print("Целевой класс:", iris.target[:5])

print("Признаки:", iris.feature_names)
print("Целевой признак:", iris.target_names)
```

```
Матрица данных:
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]]
Целевой класс: [0 0 0]
Признаки: ['sepal length (cm)', 'sepal width (cm)',
           'petal length (cm)', 'petal width (cm)']
Целевой признак: ['setosa' 'versicolor' 'virginica']
```

# sklearn.datasets

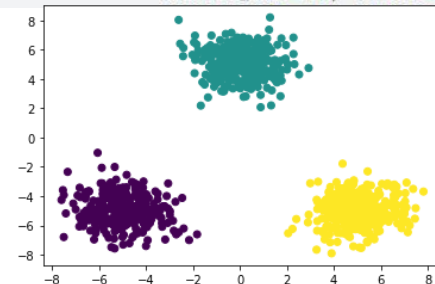
- Примеры генерации наборов
- Shape: **n\_samples**, **n\_features**
- `X, y = make_blobs():`
  - Генерирует гауссовы области точек
  - **centers** (координаты центров или кол-во)
- `X, y = make_classification():`
  - Кластеры гауссовых точек вокруг вершин гиперкуба + шум
  - **n\_informative**, **n\_redundant** (лин. комб информативных), **n\_repeat**, **n\_classes**
- `X, y, coef = make_regression():`
  - **y** генерируется по случайным данным смещенной линейной регрессией + шум
  - **n\_targets** (размерность), **noise**, **bias**



```
from sklearn.datasets import make_blobs

n_samples = 1000
centers = [(-5, -5), (0, 0), (5, 5)]

X, y = make_blobs(n_samples=n_samples, n_features=2,
                  cluster_std=1.0, centers=centers)
```



# Estimator API

Основные шаги использования API:

1. Выбор класса модели scikit-learn
2. Выбор гиперпараметров (в объекте или параметрах конструктора)
3. Упорядочивание данных: X, y
4. Обучение модели (fit)
5. Применение модели (predict, predict\_proba, transform)

Также, возможны комбинированные методы: fit\_predict(data), fit\_transform(data)

```
# Пример псевдокода
estimator = Estimator(param1=1, param2=2)
estimator.param1

estimator.fit(data)
estimator.estimated_param_

# Если прогнозирование
estimator.predict(new_data)
# Если преобразование
estimator.transform(new_data)
```

# sklearn.preprocessing (предобработка данных)

- Обучение: `fit, fit_transform`
- Преобразование: `transform, inverse_transform`
- Стандартизация:
  - `StandardScaler`:  $(x - \text{mean}(x))/\text{std}(x)$
  - `RobustScaler`:  $(x - \text{median}(x))/\text{IQR}(x)$
- Нормализация:
  - `MinMaxScaler`:  $(x - \min(x))/(\max(x) - \min(x))$
  - `MaxAbsScaler`:  $x/\max(|x|)$
  - `Normalizer(norm)` # l1, l2, max
- Преобразование:
  - `QuantileTransformer(n_quantiles)`
  - `PowerTransformer` # yeo-johnson, box-cox
  - `KBinsDiscretizer`:  $x \rightarrow 0..k-1$
  - `Binarizer`:  $x \rightarrow 0,1$

```
X = np.array([[1, 10, 100, 200, 1000]]).T

StandardScaler().fit_transform(X)
RobustScaler().fit_transform(X)
MinMaxScaler().fit_transform(X)
MaxAbsScaler().fit_transform(X)
Normalizer().fit_transform(X)
QuantileTransformer().fit_transform(X)
PowerTransformer().fit_transform(X)
KBinsDiscretizer(encode="ordinal").fit_transform(X)
Binarizer(threshold=20).fit_transform(X)
```

<b>StandardScaler:</b> [[[-0.69493808] [-0.67099305] [-0.43154271] [-0.16548679] [ 1.96296062]]	<b>MaxAbsScaler:</b> [[[0.001] [0.01 ] [0.1  ] [0.2  ] [1.   ]]	<b>PowerTransformer:</b> [[[-1.42849354] [-0.76770092] [ 0.21971925] [ 0.55897206] [ 1.41750316]]
<b>RobustScaler:</b> [[[-0.52105263] [-0.47368421] [ 0.   ] [ 0.52631579] [ 4.73684211]]	<b>Normalizer:</b> [[[1.] [1.] [1.] [1.] [1.]]]	<b>KBinsDiscretizer:</b> [[[0.] [1.] [2.] [3.] [4.]]]
<b>MinMaxScaler:</b> [[[0.   ] [0.00900901] [0.0990991 ] [0.1991992 ] [1.   ]]	<b>QuantileTransformer:</b> [[[0.  ] [0.25] [0.5  ] [0.75] [1.   ]]	<b>Binarizer:</b> [[[0] [0] [1] [1] [1]]]

# Предобработка данных

## ■ Кодирование категориальных признаков:

- Модуль `sklearn.preprocessing`
- `LabelEncoder()`
- `OrdinalEncoder(categories)`
- `OneHotEncoder(categories, min_frequency)`
- Параметр `handle_unknown` (`error`, `ignore`, ...)

```
X = [['male', 'sin', 'fit'],  
      ['female', 'cos', 'predict'],  
      ['female', 'sin', 'transform']]  
  
OneHotEncoder().fit_transform(X).toarray()  
  
[[0., 1., 0., 1., 1., 0., 0.],  
 [1., 0., 1., 0., 0., 1., 0.],  
 [1., 0., 0., 1., 0., 0., 1.]]
```

## ■ Обработка пропусков:

- Модуль `sklearn.impute`
- `SimpleImputer(missing_values, strategy)`
- `KNNImputer(n_neighbors, weights)`
- `MissingIndicator(missing_values)`

```
X = np.array([[1, np.nan], [np.nan, 3], [7, 6]])  
imp = SimpleImputer(missing_values=np.nan,  
                    strategy='mean')  
imp.fit_transform(X)  
  
knn_imp = KNNImputer(n_neighbors=1,  
                    weights="uniform")  
knn_imp.fit_transform(X)
```

```
[[1. , 4.5],  
 [4. , 3. ],  
 [7. , 6. ]]  
  
[[1. , 6. ],  
 [7. , 3. ],  
 [7. , 6. ]]
```

# Предобработка признакового пространства

## ■ Генерация признаков:

- Модуль `sklearn.preprocessing`
- `PolynomialFeatures(degree, interaction_only)`
- `SplineTransformer(degree, n_knots)`
- `FunctionTransformer(func, inverse_func)`

```
X = np.arange(6).reshape(3, 2)
PolynomialFeatures(2).fit_transform(X)
[[ 1.,  0.,  1.,  0.,  0.,  1.],
 [ 1.,  2.,  3.,  4.,  6.,  9.],
 [ 1.,  4.,  5., 16., 20., 25.]]

SplineTransformer(2, 1).fit_transform(X)
[[ 1.,  0.,  1.,  0. ],
 [ 0.5, 0.5, 0.5, 0.5],
 [ 0.,  1.,  0.,  1. ]]
```

## ■ Уменьшение размерности:

- Модуль `sklearn.decomposition`
- `PCA(n_components)` # число или `mle`
- `TruncatedSVD(n_components, algorithm)`
- Атрибуты: **`explained_variance_`**, **`singular_values_`**
- `NMF(n_components, init, solver)`
- `LatentDirichletAllocation(n_components)`
- Общий атрибут: **`components_`**

```
X = np.arange(100).reshape(10, 10)

PCA(n_components=2).fit_transform(X)
[[ 1.42302495e+02,  1.67908651e-14],
 [ 1.10679718e+02, -2.41419792e-15],
 [ 7.90569415e+01, -3.65037002e-15],
 [ 4.74341649e+01, -1.51614199e-15],
 [ 1.58113883e+01, -2.24513984e-16],
 [-1.58113883e+01,  2.24513984e-16],
 [-4.74341649e+01,  1.51614199e-15],
 [-7.90569415e+01,  3.65037002e-15],
 [-1.10679718e+02,  2.41419792e-15],
 [-1.42302495e+02,  7.91882609e-15]]
```



# Предобработка признакового пространства

```
X = np.arange(6).reshape(3, 2)
print("X.shape:", X.shape)
print("X:\n", X)

pf = PolynomialFeatures(2)
pol_X = pf.fit_transform(X)
print("pol_X.shape:", pol_X.shape)
print("pol_X:\n", pol_X)

pca = PCA(n_components=2)
decom_X = pca.fit_transform(pol_X)
print("decom_X.shape:", decom_X.shape)
print("decom_X:\n", decom_X)

print("components_:\n",
      np.round(pca.components_, 3))
print("explained_variance_:\n",
      pca.explained_variance_)
print("singular_values_:\n",
      pca.singular_values_)
```

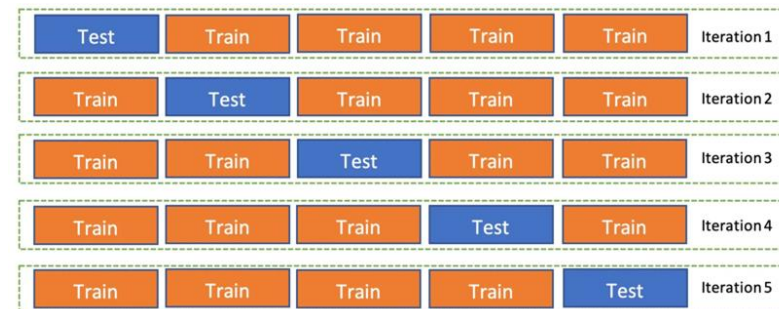
```
X.shape: (3, 2)
X:
[[0 1]
 [2 3]
 [4 5]]
pol_X.shape: (3, 6)
pol_X:
[[ 1.  0.  1.  0.  0.  1.]
 [ 1.  2.  3.  4.  6.  9.]
 [ 1.  4.  5. 16. 20. 25.]]
decom_X.shape: (3, 2)
decom_X:
[[-15.51982135 -0.68445502]
 [-4.51113148  0.99147675]
 [ 20.03095282 -0.30702173]]
components_:
[[-0.    0.107  0.107  0.457  0.564  0.671]
 [ 0.    0.488  0.488 -0.612 -0.124  0.364]]
explained_variance_:
[331.22711642  0.77288358]
singular_values_:
[25.73818628  1.24328885]
```

# sklearn.model\_selection (подготовка выборок)

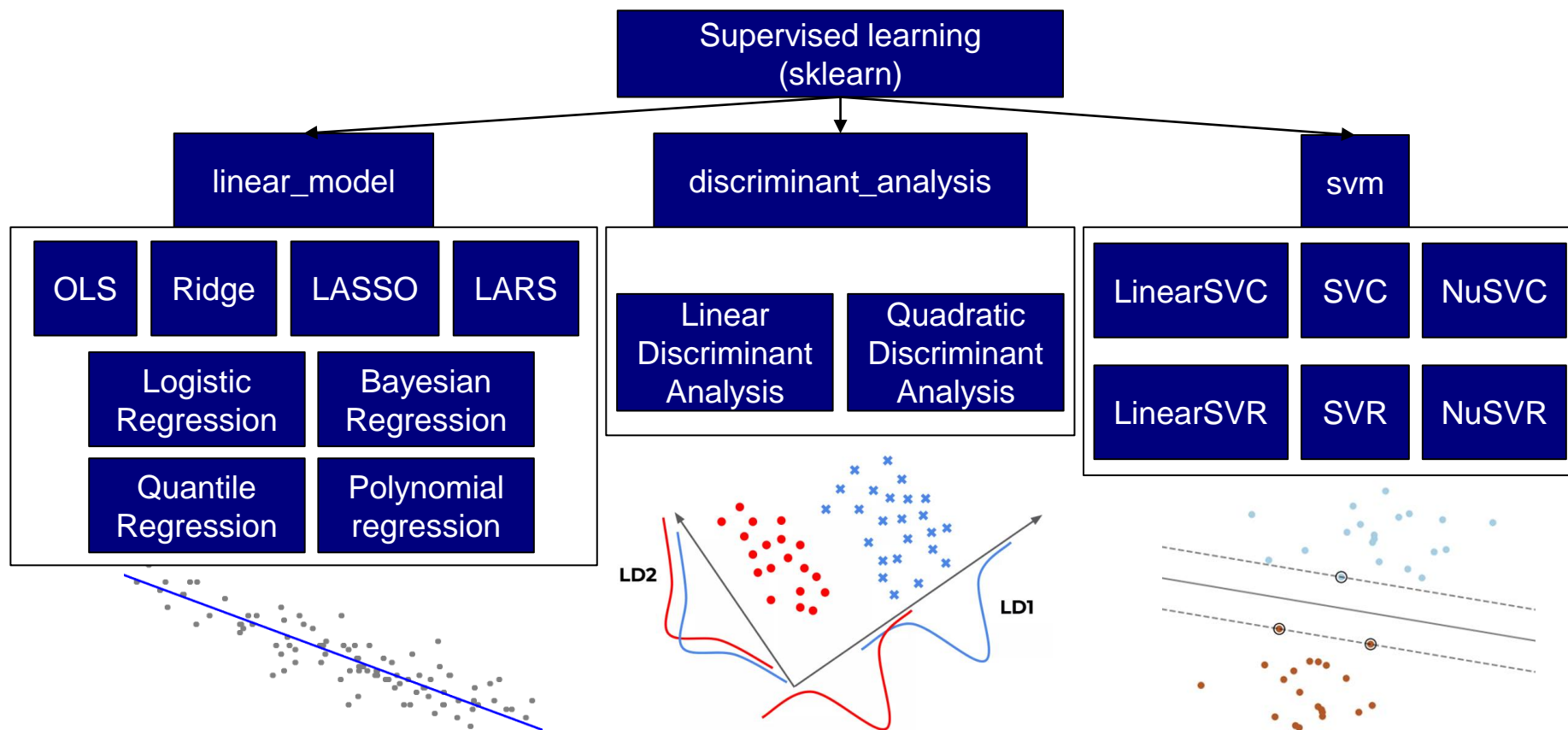
- `train_test_split(*arrays, test_size, train_size)`
  - Принимает последовательность индексируемых массивов одинаковой длины
  - Разбивает массивы на 2 подмножества
  - Возвращает список длиной  $2 * \text{len}(\text{arrays})$ : `arr1_train, arr1_test, arr2_train, arr2_test ...`
  - Параметр `stratify`
- Классы генерации разбиений:
  - Генерация выборок методом `split()`
  - `ShuffleSplit(n_splits, test_size, train_size)`
  - `KFold(n_splits)`
  - `LeaveOneOut()`
  - `Stratified<ИмяКласса>()`

```
X, y = np.arange(20).reshape((10, 2)), range(10)
X_tr, X_tst, y_tr, y_tst = train_test_split(X, y, test_size=0.33)
```

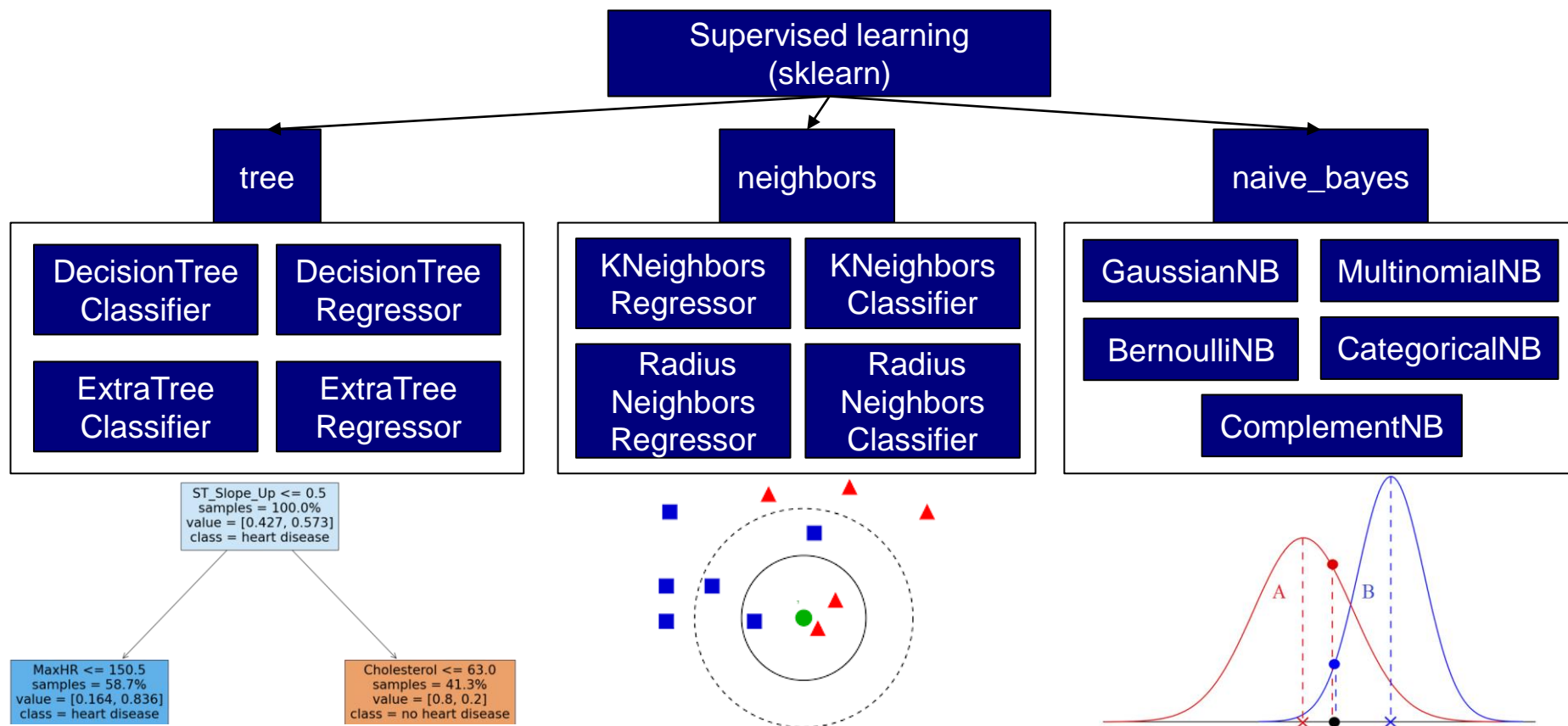
X:	Train X:	Test X:
[[ 0  1]	[[10 11]	[[ 4  5]
[ 2  3]	[12 13]	[ 0  1]
[ 4  5]	[18 19]	[14 15]
[ 6  7]	[ 6  7]	[ 2  3]
[ 8  9]	[ 8  9]	Test y:
[10 11]	[16 17]]	[2, 0, 7, 1]
[12 13]	Train y:	
[14 15]	[5, 6, 9, 3, 4, 8]	
[16 17]		
[18 19]]		
y:		
range(0, 10)		



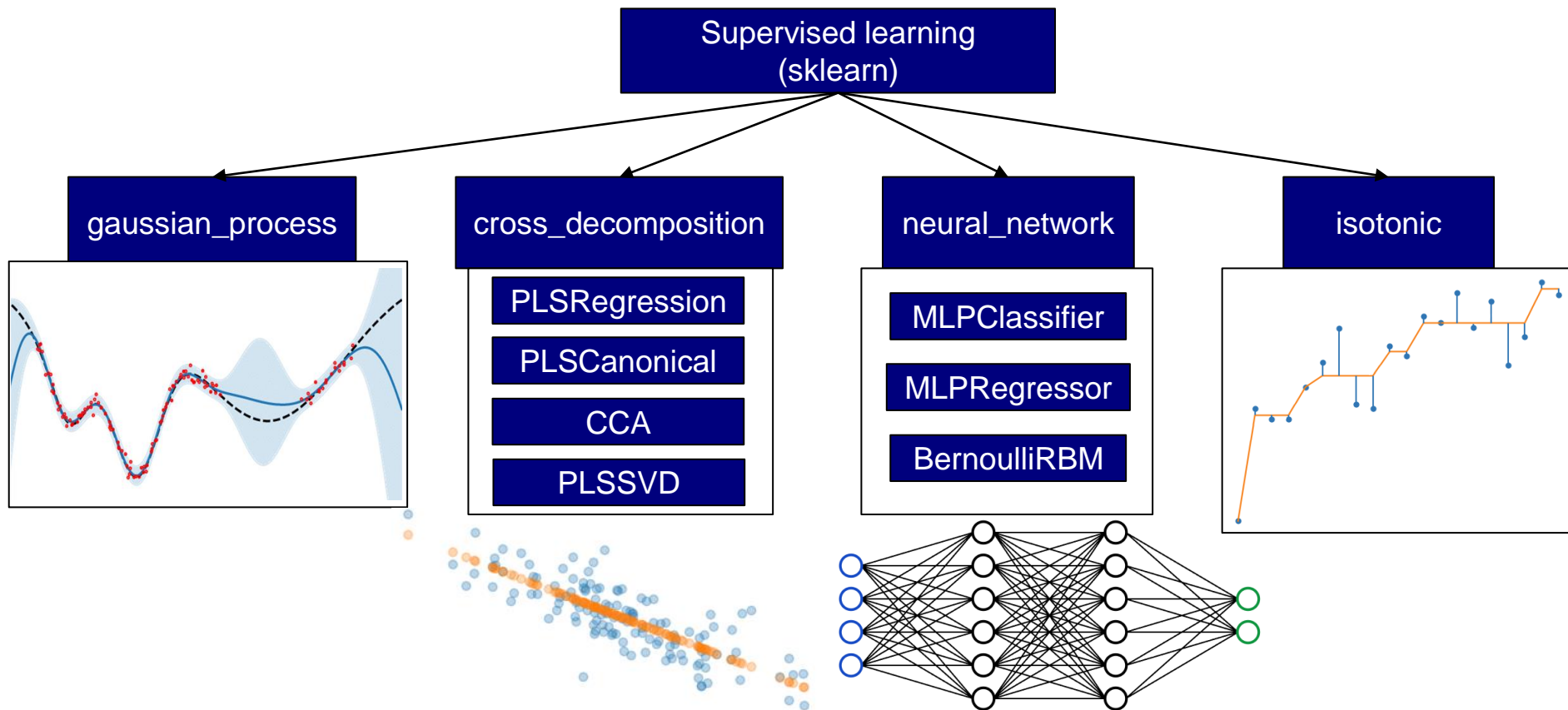
# Обучение с учителем



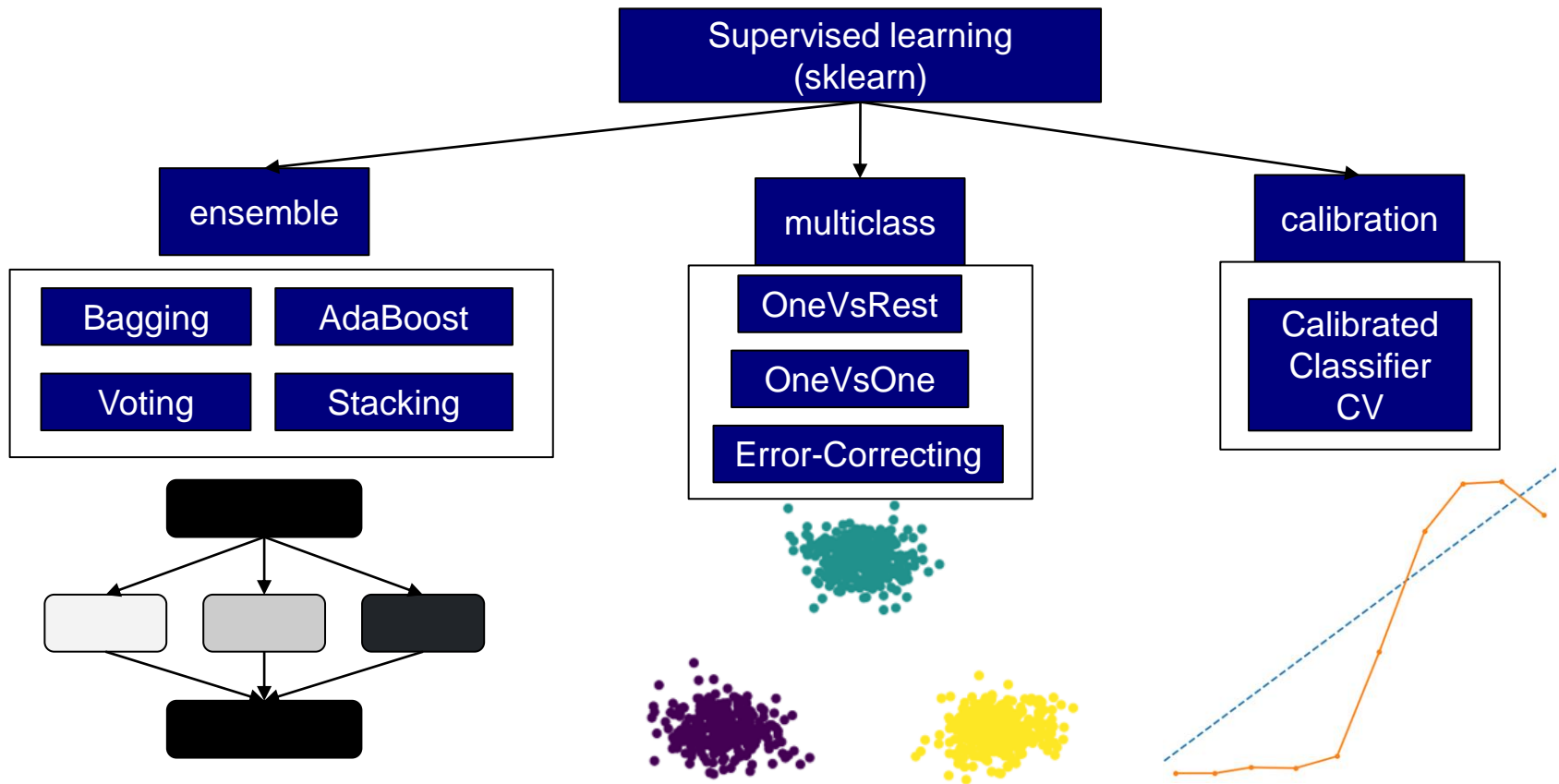
# Обучение с учителем



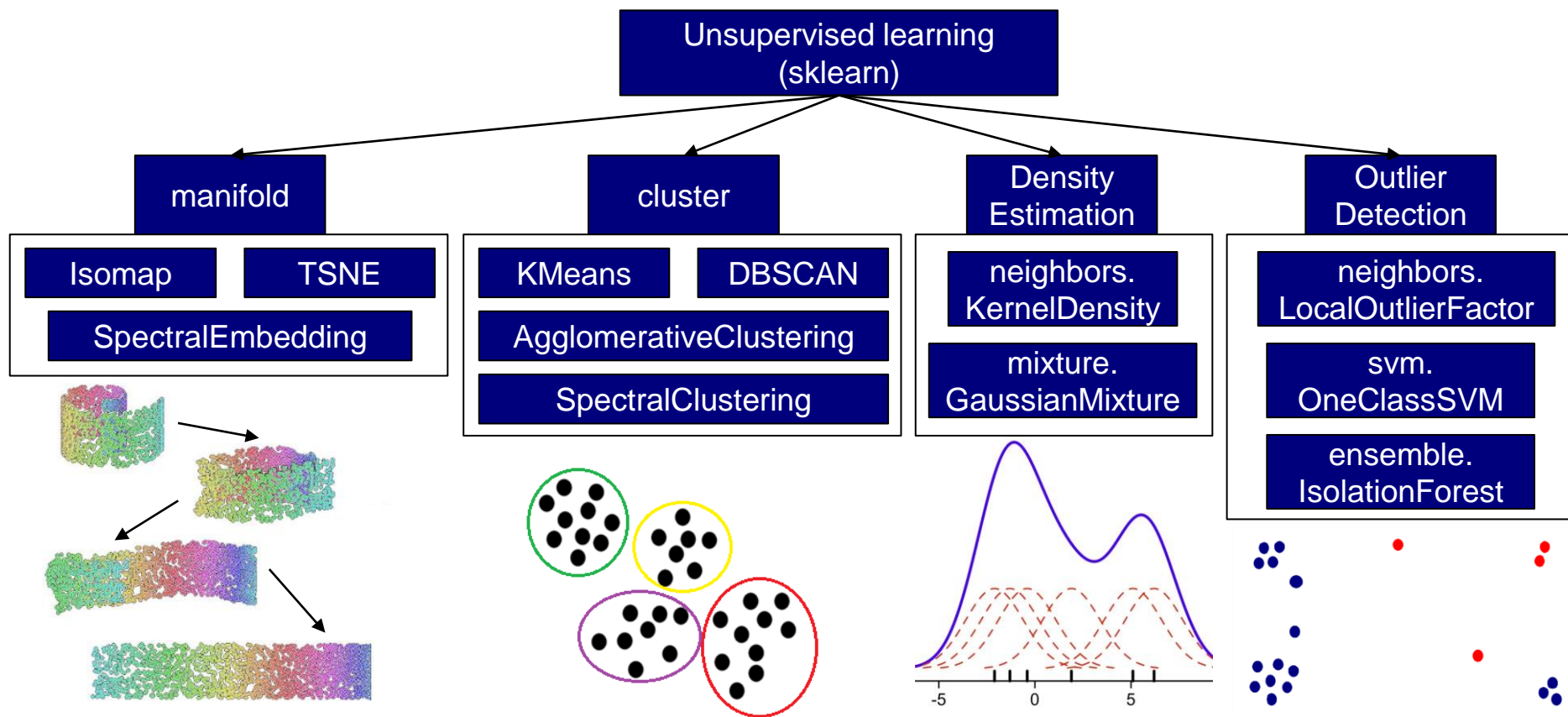
# Обучение с учителем



# Метамодели (на основе внешнего Estimator)



# Обучение без учителя



# Валидация моделей

- Метрики
  - Модуль sklearn.metrics
  - Общее API: `metric(y_true, y_pred, *params)`
- Мета-функции валидации моделей:
  - Общие параметры: `estimator`, `X`, `groups`, `scoring`
  - `cross_validate()`: оценка качества и времени
  - `cross_val_predict()`: генерация CV прогнозов
  - `cross_val_score()`: оценки на CV (вызов `score`)
  - `learning_curve()`: генерация кривых обучения
- Экспорт и импорт моделей аналогично через `joblib`: `dump`, `load` (лекция 1)

