«Машинное обучение»

Язык программирования Python и его (нужные нам) библиотеки

Александр Дьяконов

12 сентября 2022 года

Инструментарий

Язык программирования Python

https://www.python.org/

Библиотека для матричных вычислений и линейной алгебры

http://www.numpy.org/

Библиотека для научных вычислений

https://www.scipy.org/

Библиотека для визуализации

https://matplotlib.org/

Библиотека для машинного обучения

http://scikit-learn.org/

Библиотека для обработки данных

https://pandas.pydata.org/

Совет по инструментарию

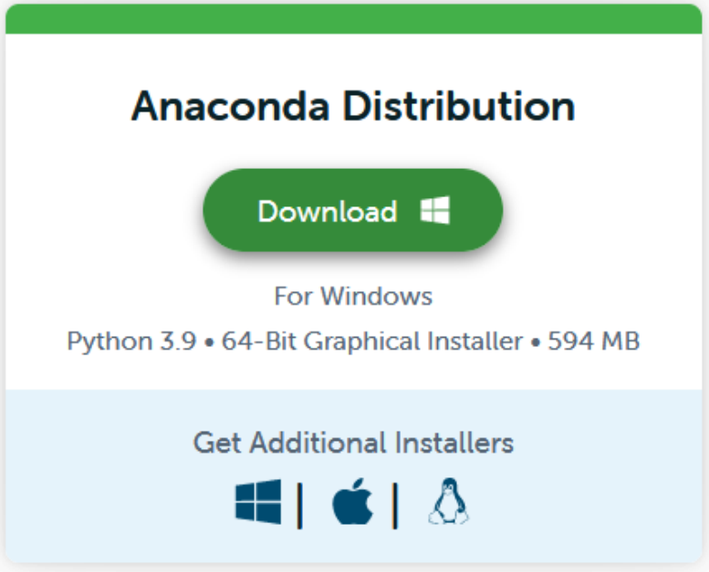
ANACONDA®

научный дистрибутив Anaconda Python
от Continuum
https://www.anaconda.com/download/

Individual Edition is now

ANACONDA DISTRIBUTION

The world's most popular open-source Python distribution platform



Совет по инструментарию

JupyterLab

Jupyter Notebook

Python, R, Julia, Scala, F#
http://jupyter.org/

Среда разработки аналогичная R-studio

эволюция IPython Notebook

для создания и обмена «ноутбуками»:

- **код**
- **полнотекстовые комментарии**
- **уравнения**
- **визуализация**

https://www.jetbrains.com/pycharm/

интегрированная среда разработки для языка программирования Python

Совет по инструментарию

Basic Numerical Integration: the Trapezoid Rule

A simple illustration of the trapezoid rule for definite integration:

$$\int_a^b f(x)\,dx \approx \frac{1}{2}\sum_{k=1}^{N}(x_k - x_{k-1})\,(f(x_k) + f(x_{k-1}))\,.$$

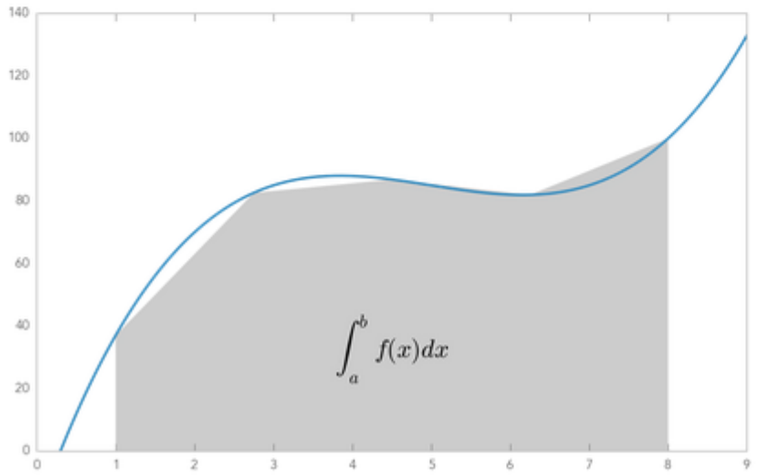
First, we define a simple function and sample it between 0 and 10 at 200 points

In [1]:
```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```
def f(x):
    return (x-3)*(x-5)*(x-7)+85

x = np.linspace(0, 10, 200)
y = f(x)
```

Choose a region to integrate over and take only a few points in that region

In [4]:
```
plt.plot(x, y, lw=2)
plt.axis([0, 9, 0, 140])
plt.fill_between(xint, 0, yint, facecolor='gray', alpha=0.4)
plt.text(0.5 * (a + b), 30,r"$\int_a^b f(x)dx$", horizontalalignment='center', fontsize=20);
```



Совет по инструментарию

```
In [31]: [1, 2, 3]
Out[31]: [1, 2, 3]

In [32]: _[0] # предыдущая ячейка
Out[32]: 1

In [33]: sum(__) # пред-предыдущая ячейка
Out[33]: 6

In [35]: Out[31] # конкретная ячейка
Out[35]: [1, 2, 3]

In [38]: pwd # unix-dos-команды
Out[38]: u'C:\\tmp\\notebooks'
```

возможность программировать (и проводить эксперименты) в браузере

Пример решения задачи ML

https://github.com/Dyakonov/notebooks/blob/master/dj_benchmark_GMSC_01.ipynb

```
from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier(n_estimators=100,
                               max_depth=2,
                               random_state=0) # модель

model.fit(train, y) # обучение

a = model.predict_proba(test)[:,1] # предсказание
```

Пример решения соревновательной задачки

https://github.com/Dyakonov/notebooks/blob/master/mkb_benchmark.ipynb

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

In [2]:
```
data_train = pd.read_csv('train_dataset_hackathon_mkb.csv', encoding='cp1251', delimiter=';')
data_test = pd.read_csv('test_dataset_hackathon_mkb.csv', encoding='cp1251', delimiter=';')
print (data_train.shape, data_test.shape)
```

```
(17891, 124) (7330, 123)
```

In [3]:
```
def makeX(data):
    # предобработка данных
    data['CITIZENSHIP_NAME'] = data['CITIZENSHIP_NAME'].fillna(-1).map({-1: -1, 'Российская Федерация': 4, 'Таджикистан': 3, 'Казахстан':
    data['SEX_NAME'] = data['SEX_NAME'].fillna(0).map({0: 0, 'мужской': 1, 'женский': -1})
    group_names = ['OKFS_GROUP', 'OKOPF_GROUP', 'OKOGU_GROUP'] + ['WORKERSRANGE', 'OKVED_CODE']
    date_names = ['SIGN_DATE', 'DATEFIRSTREG', 'TAXREG_REGDATE', 'TAXREGPAY_REGDATE', 'BIRTHDATE']
    for name in group_names + date_names + ['id_client']:
        data[name] = data[name].fillna(-1)
        tmp = data[name].value_counts()
        tmp = tmp + 0.1 * np.random.randn(len(tmp))
        data[name] = data[name].map(tmp)
    data.fillna(-1, inplace=True)
    return data
```

In [4]:
```
data_train = makeX(data_train) # обрабатываем обучение
data_test = makeX(data_test) # обрабатываем тест
```

In [5]:
```
y = data_train.pop('TARGET').values # целевые значения
data_test = data_test[data_train.columns] # на всякий случай - вдруг, перемешаны столбцы
```

In [6]:
```
import lightgbm as lgb

model = lgb.LGBMClassifier(num_leaves=31,
                           learning_rate=0.05,
                           n_estimators=200)
```

In [7]:
```
model.fit(data_train, y)

a = model.predict_proba(data_test)[:, 1] # получаем ответ
```

In [8]:
```
df = pd.DataFrame({'id_contract': data_test.id_contract.values, 'TARGET': a})
df.to_csv('ans1.csv', sep=';', index=False) # сохраняем ответ
```

Python

«Питон» или «пайтон» – в честь комедийных серий BBC «Летающий цирк Монти-Пайтона»

Создатель – голландец Гвидо ван Россум (Guido van Rossum), 1991 г.

Особенности

- интерпретируемый
- объектно-ориентированный
- высокоуровневый язык
- встроенные высокоуровневые структуры данных
- динамическая типизация
- синтаксис прост в изучении
- поддержка модулей и пакетов (МНОГО бесплатных библиотек)
- универсальный
- интеграция с другими языками (C, C++, Java)

Важно для нас

- **Прост в освоении**
  рекомендуют и используют как первый язык программирования

- **Короткие конструкции**
  программы пишутся быстро и легко читаются

- **Есть хорошие библиотеки для ML**
  numpy, scipy, scikit-learn, pandas, matplotlib + обёртки для библиотек DL

Поддерживаемые парадигмы

- императивное (процедурный, структурный, модульный подходы) программирование
- объектно-ориентированное программирование
- функциональное программирование

PEP8 https://www.python.org/dev/peps/pep-0008/

стилистические рекомендации по оформлению кода

- отступ – 4 пробела
- длина строки < 80 символов
- переменные: `var_recommended`
- константы: `CONST_RECOMMENDED`
- ...

```
import this
```

The Zen of Python, by Tim Peters

- Beautiful is better than ugly.
- Explicit is better than implicit.
- Simple is better than complex.
- Complex is better than complicated.
- Flat is better than nested.
- Sparse is better than dense.
- Readability counts.
- Special cases aren't special enough to break the rules.
- Although practicality beats purity.
- Errors should never pass silently.
- Unless explicitly silenced.
- In the face of ambiguity, refuse the temptation to guess.
- There should be one-- and preferably only one --obvious way to do it.
- Although that way may not be obvious at first unless you're Dutch.
- Now is better than never.
- Although never is often better than *right* now.
- If the implementation is hard to explain, it's a bad idea.
- If the implementation is easy to explain, it may be a good idea.
- Namespaces are one honking great idea -- let's do more of those!

Переменные – названия переменных

- содержат буквы, цифры, знак подчёркивания
  - с подчёркиванием есть тонкости
- не начинаются с цифры
- чувствительны к регистру

ключевые слова

```
help("keywords")
```

```
False       class       from        or
None        continue    global      pass
True        def         if          raise
and         del         import      return
as          elif        in          try
assert      else        is          while
async       except      lambda      with
await       finally     nonlocal    yield
break       for         not
```

Основы Python: условный оператор, функция

```
# функция
def sgn(x):
    """
    функция 'знак числа'
    +1 - для положительного аргумента
    -1 - для отрицательного аргумента
    0 - для нуля
    Пример: sgn(-2.1) = -1
    """
    # if - условный оператор
    if x > 0:
        a = +1
    elif x < 0:
        a = -1
    else:
        a = 0
    return a

sgn(2.1), sgn(0), sgn(-2)
(1, 0, -1)

  "nonzero" if x != 0 else "zero" # другой вариант условного оператора
```

многострочных комментариев нет – часто используются строки

но важны отступы (4 пробела)
нет операторных скобок и end

обратите внимание на двоеточие

после return скобки не обязательны

для помощи – `help (sgn)`
выведется оранжевый текст

Основы Python: цикл for, вывод

```
# for - цикл
for i in range(1, 4):
    s = ""
    for j in range(1, 4):
        s += ("%i " % (i * j))
    print (s)
```

```
1 2 3
2 4 6
3 6 9
```

```
# можно много по чему итерироваться
for i in [10, 20]:
    for j in 'ab':
        print (i, j)
```

```
(10, 'a')
(10, 'b')
(20, 'a')
(20, 'b')
```

range – это итератор (см. дальше)

после «:» должно быть 4 пробела

«+» – конкатенация строк (см. дальше)

нет явного счётчика (см. дальше)

как и ожидается, есть

`continue`

`break`

Интересно: есть и такие сокращения операций (работают с числами)

«+=» «-=» «*=» «/=»

Основы Python: цикл while, ввод

```
# while - цикл

s = input("Введите строку:")

while s: # s != "":
    print (s)
    s = s[1:-1]
```

```
Введите строку:12345
12345
234
3
```

`input` – **ввод именно строки**

(в Python3 !)

`[1:-1]` – **«слайсинг»:**

без первой и последней букв

(см. дальше)

Нет цикла с постусловием!

Пример решения задачи на Python

```
import math

def primes(N):
    """Возвращает все простые от 2 до N"""
    sieve = set(range(2, N))
    for i in range(2, round(math.sqrt(N))):
        if i in sieve:
            sieve -= set(range(2 * i, N, i))
    return sieve

primes(20)
```

```
{2, 3, 5, 7, 11, 13, 17, 19}
```

Вывести простые числа

Здесь задействован тип «множество»

(см. дальше)

Можно переносить строки с помощью «\», иногда просто переносить

```
x = 1 + 2 + 3 + 4 +\
5 + 6 + 7 + 8
```

```
x = (1 + 2 + 3 + 4 +
5 + 6 + 7 + 8)
```

Где один пробел – можно много

«Сложные» условия

```
x = 4
if 3 < x < 5: # можно без скобок
    print ('четыре')
```
```
четыре
```

```
if 3 < x and x < 5:
    print ('четыре')
```
```
четыре
```

```
# проверка списка на пустоту
if not lst:
    ...
```

объект `False`, если он пуст

Строка

в Python3 – неизменяемый массив символов Юникода

«рекурсивная структура данных» –
каждый символ объект типа str длины 1
нет понятия символ (это одноэлементная строка)

```
# задание одной и той же строки
s1 = "string"
s2 = 'string'
s3 = """string"""
s4 = 'st' 'rin' 'g' # будет склейка (аналогично +)
s5 = 'st' + 'rin' + 'g'

s = u'\u043f\u0440\u0438\u0432\u0435\u0442' # можно убрать u
print(s)
```

```
привет
```

Операции над строками

```
print ('A' + 'B') # конкатенация
```

```
AB
```

```
print ('A' * 3) # повтор
```

```
AAA
```

Форматирование строк – четыре способа

```
print ('Привет, %s' % name)

print ('Привет, {}'.format(name))

# форматированные строковые литералы (Formatted String Literals)
print (f'Привет, {name}')

# шаблонные строки
from string import Template
print (Template('Привет, $name').substitute(name=name))
```

Строки: операции

```
s = 'one,one'

s.count('on') # подсчёт вхождения подстроки
2
s.find('on') # поиск подстроки (есть ещё index - с исключениями)
0
s.rfind('on') # поиск последней подстроки (последнее вхождение)
4
s.isalpha() # только буквы
False
s.islower() # только строчные / isupper / istitle / isspace
True
s.isdigit() # число
False
s.isalnum() # только буквы и цифры
False
s.replace('on','off') # замена подстрок
offe,offe
s.translate({ord('o'): 'a', ord('n'): 'b'}) #множеств.замена симв.
abe,abe
```

...

Строки: операции

```
'   12  '.strip() # del 1-ых и lst-их пробелов,ещё: lstrip, rstrip
12
s.upper() # в верхний регистр + lower
ONE,ONE
s.capitalize() # первую букву в верхний регистр, ост. - в нижний
One,One
'file.txt'.endswith('.exe') # startswith
False
s.rpartition(',') # расщепление по разделителю
('one', ',', 'one')


# вхождение подстроки (проверка, а не поиск)
s = "one,two,three"

'on' in s
True

'ab' not in s
True
```

Строки: операции

```
s = 'one,two,three'
s2 = s.split(',') # расщепление в список
print (s, s2)
one,two,three ['one', 'two', 'three']

print (";".join(s2)) # объединение через разделитель
one;two;three

# выравнивание в блоке фиксированной длины
s = 'my string'
print (s.ljust(13, ' '))
print (s.center(13, '-'))
print (s.rjust(13)) # пробел можно не указывать

my string
--my string--
    my string
```

индексация как в списках – будет дальше

Список (list)

изменяемый динамический массив

Простейший и удобный контейнер – для хранения перечня объектов (а в питоне всё – объект)

контейнер для разнородных элементов

```
s = [1, 'string', [1,2,3], True]

s[1]
'string'

s[2][0]
1
```

списки могут быть вложенные

```
a = [1, 2, 3]
b = [4, 5, 6]
lst = [1, [a, b]]

lst[1][0][2]
3
```

Список (list): задание

Для начала, простой вариант – перечень чисел

```
[1, 2, 3] # список

[x for x in range(3)] # потом узнаем о ...
[0, 1, 2]

# преобразование типов
list(range(3)) # из генератора

[0, 1, 2]

list(u'строка') # из строки
['с', 'т', 'р', 'о', 'к', 'а']

list({1, 3, 1, 2}) # из множества
[1, 2, 3]

list((1, 1, 2)) # из кортежа
# меньше скобок нельзя
[1, 1, 2]
```

Задаётся с помощью квадратных скобок

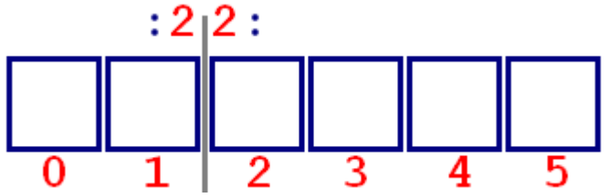
[] – пустой список

Можно преобразовывать из других объектов

Индексация, нарезка (slicing)

для списков, строк и т.д.

```
s = [0, 1, 2, 3, 4, 5]

s[2] # третий! элемент
2

s[:2] # первые два элемента
[0, 1]

s[2:] # после второго
[2, 3, 4, 5]

s[:-2] # без двух элементов
[0, 1, 2, 3]

s[-2:] # последние два
[4, 5]

s[0:4:2] # от : до : шаг
[0, 2]
```



Нумерация с нуля

Индексация слева и справа

Схема (от : до : шаг)

```
s[::3] # все через шаг
[0, 3]

s[::-1] # в обратном порядке
[5, 4, 3, 2, 1, 0]
```

Список (list): операции

```
s = [1, 2, 3] # это список

len(s) # длина списка!
3

max(s) # максимальный элемент
3

2 in s # принадлежность списку
True

s + s # конкатенация
# создаётся новый список!
[1, 2, 3, 1, 2, 3]

s * 2 # "удвоение"
[1, 2, 3, 1, 2, 3]

del s[1] # удаление элемента
s
[2, 3]
```

```
s[0] = 100  # присваивание значения
s
[100, 3]

# сравнение (лексикографический порядок)
print([1, 2] < [1, 3])
print([1, 2] < [1, 2, 1])
print([2] < [1, 3])
True
True
False

l = [1, 2, 3, 4, 5]
l[2:5] = 100 # не работает
l[2:5] = [100] # Работает!
```

Есть естественные функции: максимум, минимум, сумма

Список (list): операции

```
s = [4] * 3 # [4, 4, 4]
# удаление первого вхождения элемента
s.remove(4)
```
```
[4, 4]
```
```
# добавление элемента
s.append(2)
```
```
[4, 4, 2]
```
```
# добавление последовательности
s.extend([3, 3])
```
```
[4, 4, 2, 3, 3]
```
```
# сколько элементов
s.count(4)
```
```
2
```
```
# индекс элемента (первое вхождение), если не входит - исключение ValueError
s.index(2)
```
```
2
```
```
# инвертирование
s.reverse()# или s = s[::-1])
```
```
[3, 3, 2, 4, 4]
```
```
s.sort() # сортировка
```
```
[2, 3, 3, 4, 4]
```
```
# вырезает элемент (по индексу) pop() - последний
s.pop(1)
```
```
3, [2, 3, 4, 4]
```
```
# вставка элемента
s.insert(0, 1)
```
```
[1, 2, 3, 4, 4]
```
```
# вставка элемента
s.insert(-1, 5)
```
```
[1, 2, 3, 4, 5, 4]
```
```
# вставка элементов
s[-4:] = [0]*4
```
```
[1, 2, 0, 0, 0, 0]
```
```
# удаление элементов
del s[-3:]
```
```
[1, 2, 0]
```

Здесь не всегда показаны возвращаемые функциями значения, а просто текущий список!

Тонкости питона: копирование

```
x = [[0]]*2 # делаем список
```

```
[[0], [0]]
```

```
x[0][0] = 1 # меняем один элемент
# ... а поменялись оба
```

```
[[1], [1]]
```

```
id(x[0]), id(x[1])
```

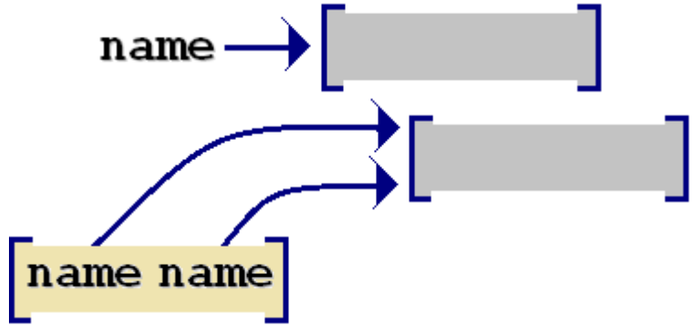
```
(72882632, 72882632)
```

```
x = x + x
x[0][0] = 2 # такой же эффект
```

```
[[2], [2], [2], [2]]
```

При операции * не происходит копирования списка!

id – идентификатор объекта (уникален)




```
[[0] for x in range(2)] # можно так!

# но так снова плохо...
b = [0]
a = [b for x in range(2)]
```

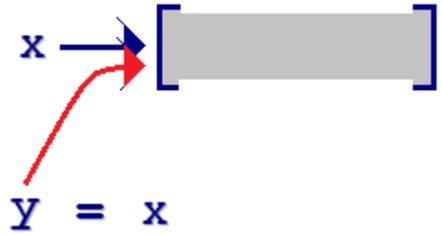
Тонкости питона: копирование

Как быть – копирование (поверхностное)

```
from copy import copy
x = [copy([0]) for i in range(2)]
x[0][0] = 1
```

```
[[1], [0]]
```

При присваивании просто создаётся ссылка на объект



Способы копирования списка:

```
new_list = old_list[:]

new_list = list(old_list) # аналогично dict, set

import copy
new_list = copy.copy(old_list)
```

Тонкости питона: копирование

Копирование не всегда помогает,
выручает глубокое копирование (частично ;)

```
from copy import copy

a = 1
b = [a, a]
c = [b, b]
c2 = copy(c)
c2[0][0] = 0

print (c)
print (c2)
```

```
[[0, 1], [0, 1]]
[[0, 1], [0, 1]]
```

```
from copy import deepcopy

a = 1
b = [a, a]
c = [b, b]
c2 = deepcopy(c)
c2[0][0] = 0

print (c)
print (c2)
```

```
[[1, 1], [1, 1]]
[[0, 1], [0, 1]]
```

Кортеж (tuple): присваивание через кортеж

Кортеж – «неизменяемый список», вместо [ ] будут () или ничего

```
a, b, c = 1, 2, 3 # a это кортеж!
print (a, b, c)
(1, 2, 3)

(x, y), (z, t) = [1, 2], [3, 4]
print (x, y, z, t)
(1, 2, 3, 4)

# сработает последнее присваивание
x, (x, x) = 1, (2, 3)
print (x)
3

a, b = 1, 2
a, b = b, a # присваивание кортежей
```

При инициализации переменных часто бывают такие конструкции

Переменные могут обменяться значениями без использования ещё одной переменной

Кстати, о присваивании… допустимы такие конструкции

```
i = j = k = 1
```

Кортеж (tuple): задание

```
# разные способы задания кортежа
a = 1, 2, 3
a = (1, 2, 3)
a = tuple((1, 2, 3))
```

```
# пустой кортеж
a = () # раньше (,)
a
()
```

```
x = (2,) # одноэлементный кортеж
print (x)
[y] = x # элемент этого кортежа
y
(2,)
2
```

Неизменяемый тип

Контейнер

Может содержать объекты разных типов

```
# итерация по кортежу и его вывод
for x in a:
    print (x)
1
2
3
```

Словарь (dict)

контейнер для хранения данных вида (key, value),
порядок не важен (в Python >= 3.6 он всё-таки автоматически выдерживается)

ключ – любой хешируемый тип
(есть hash-значение, которое не меняется)

Может содержать разные объекты (разных типов)

Словарь (dict)

```
dct = {'a': 1, 'b': 2} # словарь
{'b': 2, 'a': 1}

dct = dict(a=1, b=2) # другой способ
{'b': 2, 'a': 1}

# добавление к словарю
dct = dict(dct, a=3, d=2)
{'d': 2, 'a': 3, 'b': 2}

# преобразование из списка
dct = dict([('a', 1), ('b', 2)])
{'b': 2, 'a': 1}


dct['a'] # обращение по ключу (если нет - исключение KeyError)
1

# обращение по ключу со значением по умолчанию (когда ключ не найден)
dct.get('c', 0)
0
```

```
# проверка не-вхождения
'a' not in dct
False

del dct['a'] # удаление по ключу
dct
{'b': 2}

dct.keys() # ключи
dict_keys(['b', 'a'])

dct.values() # значения
dict_values([2, 1])

dct.items() # пары (ключ, значение)
dict_items([('b', 2), ('a', 1)])
```

обратите внимание на использование `dict`

Итерации по словарю

```
dct = {'a': 1, 'b': 2}
dct[0] = 5

# цикл по парам
for key, val in dct.items():
    print (key, val)
b 2
0 5
a 1

# цикл по ключам словаря
for key in dct: # dct.keys()
    print (key, dct[key])
b 2
0 5
a 1

# цикл по значениям словаря
for val in dct.values():
    print (val)
2
5
1
```

Ключ не обязательно строка,
м.б. число
(главное, что должен хэшироваться)

```
print ('длина словаря = %i' % len(dct)) #
количество пар в словаре

длина словаря = 3


d.clear() # удалить все значения
```

Ещё способы задания словаря

```
# преобразование типов
a = ['a', 'b', 'c']
b = [1, 2, 3]
dict(zip(a, b)) # см. потом про zip
```

```
{'a': 1, 'b': 2, 'c': 3}
```

Задача: объединить два словаря, не портя их

```
dct = {'a':1, 'b':2}
dct2 = {'b':3, 'c':4}

union = {**dct, **dct2} # Python3-способ
print(union, dct, dct2)
```

```
{'a': 1, 'b': 3, 'c': 4} {'a': 1, 'b': 2} {'c': 4, 'b': 3}
```

Одно из применений словарей – имитация switch

```
def first():
    print ('one')
def second():
    print ('two')
def third():
    print ('three')

x = 2

# плохой способ
if (x == 1):
    first()
elif (x == 2):
    second()
elif (x == 3):
    third()
```

```
# Python-style способ
dct = {1: first, 2: second, 3: third}
dct[x]()
```

Множество (set): операции

```
a = {1, 2, 3}
b = {2, 3, 4}

#  пересечение
a & b
a.intersection(b) #  2-й способ
{2, 3}

#  объединение
a | b
a.union(b) #  2-й способ
{1, 2, 3, 4}

#  разность
a - b
a.difference(b) #  2-й способ
{1}
```

```
# вложения
a <= b
False
a < b
False
a > b
False
```

Аргументов может быть много

```
x, y, z = {1, 2}, {3}, {1, 3, 4}

set.union(x, y, z)
{1, 2, 3, 4}

set.difference(x, y, z) # x - y - z
{2}
```

Задача: только ли из уникальных элементов состоит список

```
if len(x) == len(set(x)):
    print('List is unique!')
```

Файл (file)

```
f1 = open("file1.txt", "r")
f2 = open("file2.txt", "w",
          encoding="cp1251")
for line in f1.readlines():
    f2.write(line)
f2.close()
f1.close()
```

```
# чтобы не забывать закрывать файлы
with open('tmp.txt') as fin:
    for line in fin:
        # ...
```

Это называется

менеджер контекста

(не надо явно закрывать файл)

Функциональное программирование

в ФП вычисление – вычисление значений математических функций, а не последовательность процедур

императивный стиль

```
target = []
# для каждого элемента
# исходного списка
for item in source_list:
    # применить функцию G()
    trans1 = G(item)
    # применить функцию F()
    trans2 = F(trans1)
    # добавить ответ в список
    target.append(trans2)
```

функциональный стиль

```
# языки ФП часто имеют
# встроенную функцию compose()
compose2 = lambda A, B: lambda x: A(B(x))
          target = map(compose2(F, G),
                       source_list)

# list(...) в Python-3
```

«что нужно вычислить, а не как»

Функциональное программирование

- **Есть функции первого класса / высшего порядка**
  (принимают другие функции в качестве аргументов или возвращают другие функции, их можно присваивать и хранить)
- **Рекурсия – основная управляющая структура в программе**
  (нет цикла – он реализован через рекурсию)
- **Обработка списков (например, `print(len([1+1, 1/0]))`)**
- **Запрещение побочных эффектов у функций**
  (чистые функции – зависят только от своих параметров и возвращают только свой результат)
- **Описываем не шаги к цели, а математическую зависимость данные–цель**
  (в идеале, программа - одно выражение с сопутствующими определениями)

Питон – язык с элементами функционального стиля!

Функции первого класса

```
# функции первого класса
def create_adder(x):
    def adder(y): # определяем функцию внутри
        return x + y
    return adder # её же возвращаем

add_10 = create_adder(10)
print (add_10(3))
f = add_10 # та же функция
del add_10 # не удаляет саму функцию
print(f(0))
```

```
13
10
```

```
print(f.__name__)
```

```
adder
```

Интересно здесь также, что эта ф-ия – лексическое замыкание (lexical closure) – помнит значение из лексического контекста: что надо прибавлять именно 3!

Функции – полноправные объекты

могут быть

- **созданы во время выполнения**
- **присвоены переменной**
- **переданы функции в качестве аргументов**
- **возвращены функцией в качестве результата**

Всегда что-то возвращают

по умолчанию `return None`

Аргументы функций

именованные аргументы

```
def f(x=1, y=2):
    print ('x=%g, y=%g' % (x, y))

f(3, 4)
f(3)
f(y=10, x=20)
f(y=0)
```

```
x=3, y=4
x=3, y=2
x=20, y=10
x=1, y=0
```

сколько угодно аргументов – с помощью «упаковки»

```
def max_min(*args):
    # args - список аргументов
    # в порядке их указания при
    # вызове
    return max(args), min(args)

print (max_min(1, 2, 3, 4, 5))
print (max_min(*[4, 0, 3]))
print (max_min(*(1, 7, 3)))
print (max_min(*{6, 2, 4}))
```

```
(5, 1)
(4, 0)
(7, 1)
(6, 2)
```

возвратить можно только одно значение, но это м.б. кортеж!

Что такое распаковка (в Python 3)

```
first, *other = range(3)
print(first, other)
```
```
0 [1, 2]
```

```
*a, b, c = range(4)
a, b, c
```
```
([0, 1], 2, 3)
```

```
for a, *b in [range(3), range(2)]:
    print(a, b)
```
```
0 [1, 2]
0 [1]
```

```
[*range(5), 6]
```
```
[0, 1, 2, 3, 4, 6]
```

```
# инициализация контейнера
d = {'a':1, 'b':2}
d = {**d, 'a':3}
d
```
```
{'a': 3, 'b': 2}
```

```
def print_vec(x, y, z):
    print('<%s, %s, %s>' % (x, y, z))

lst = [1, 2, 3]
tpl = (4, 5, 6)
gen = (i*i for i in range(3))

print_vec(*lst)
print_vec(*tpl)
print_vec(*gen)

dct = {'y': 10, 'z': 20, 'x': 30}
print_vec(*dct) # нет гарантии порядка
print_vec(**dct)
```

```
<1, 2, 3>
<4, 5, 6>
<0, 1, 4>
<y, z, x>
<30, 10, 20>
```

Аргументы функций (с неизвестным числом аргументов)

```
# arg1 - фиксированный (здесь - 1 мы
#        обязательно должны передать)
# args - произвольные
# kwargs - любые

def swiss_knife(arg1, *args, **kwargs):
    print (arg1)
    print (args)
    print (kwargs)
    return None

swiss_knife(1, 2, [3, 4], b=-1, a=0)
```

```
1
(2, [3, 4])
{'b': -1, 'a': 0}
```

фактический синтаксис здесь – «*» и «»**

***args собирает аргументы в кортеж**

****kwargs – в словарь**

Лямбда-функции (анонимные)

```
# лямбда-функции (анонимные)
func = lambda x, y: x + y

print (func(1, 2))
```

```
3
```

```
(lambda x, y: x + y)(1, 2)
```

неявная инструкция return –
что вычисляется, то сразу возвращается

Пример использования

```
print (sorted(tuples, key=lambda x: x[0])) # так по умолчанию!
print (sorted(tuples, key=lambda x: x[0]*x[0]))
```

```
[(-2, 'MINUS TWO'), (-1, 'MINUS ONE'), (1, 'ONE'), (4, 'FOUR')]
[(1, 'ONE'), (-1, 'MINUS ONE'), (-2, 'MINUS TWO'), (4, 'FOUR')]
```

кстати, можно так:

```
import operator
sorted(tuples, key=operator.itemgetter(0))
```

Является ли объект вызываемым (функцией)

```
f = lambda x: x + 1
x = [1, 2, 3]
callable(f), callable(len), callable(x)
```

```
(True, True, False)
```

Любой объект может вести себя как функция, если определить `__call__`

Сохранение значений аргументов

```
# lst - хранится...
def mylist(val, lst=[]):
    lst.append(val)
    return lst

print(mylist(1))
print(mylist(2))
print(mylist(3))
```

```
[1]
[1, 2]
[1, 2, 3]
```

Значения по умолчанию вычисляются один раз – в момент определения функции. Python просто присваивает это значение (ссылку на него!) нужной переменной при каждом вызове функции.

```
# lst не сохраняется!
def mylist(val, lst=None):
    lst = lst or []
    # if lst is None:
    #     lst = []
    lst.append(val)
    return lst

print(mylist(1))
print(mylist(2))
print(mylist(3))
```

```
[1]
[2]
[3]
```

Часто очень полезно!

Тут не используем изменяемое значение как значение по умолчанию.

Глобальные и локальные переменные

```
b = 10
def f(a):
    # global b - если вставить - работает!
    print (a)
    print (b)
    b = 0 # ИЛИ если убрать - работает!
f(1)
```

```
1
UnboundLocalError:
local variable 'b'
referenced before assignment
```

```
3           0 LOAD_GLOBAL              0 (print)
            3 LOAD_FAST                0 (a)
            6 CALL_FUNCTION            1 (1 positional, 0 keyword pair)
            9 POP_TOP

4          10 LOAD_GLOBAL              0 (print)
           13 LOAD_FAST                1 (b)
           16 CALL_FUNCTION            1 (1 positional, 0 keyword pair)
           19 POP_TOP

5          20 LOAD_CONST               1 (0)
           23 STORE_FAST               1 (b)
           26 LOAD_CONST               0 (None)
           29 RETURN_VALUE
```

Сначала выводится 1!

Интерпретатор думает, что надо распечатать локальную переменную, а она ещё не объявлена!

Передача аргументов по ссылке

по ссылке

```
a = {'a' : 1, 'b': 2}
def f(a):
    a['b'] = 20
    a.update({'c' : 3})
    print('in', a)
print ('before', a)
f(a)
print('out', a)
('before', {'a': 1, 'b': 2})
('in', {'a': 1, 'c': 3, 'b': 20})
('out', {'a': 1, 'c': 3, 'b': 20})

a = [1, 2]
def f(a):
    a[1] = 3
    print('in', a)
print ('before', a)
f(a)
print('out', a)
('before', [1, 2])
('in', [1, 3])
('out', [1, 3])
```

по значению
(на самом деле поведение как«по значению»)

```
a = 1
def f(a):
    a = 2
    print('in', a)
print ('before', a)
f(a)
print('out', a)

('before', 1)
('in', 2)
('out', 1)
```

Зависит от изменяемости типа

Но не всё так просто…

Попробуйте `a = [3, 4]`

Описание функций

```
class MyClass:
    """
    __comment__
    """
    def __init__(self, var):
        self.var = var
    def __repr__(self):
        # для разработчиков
        return '__repr__ %g' % self.var
    def __str__(self):
        # удобочитаемо (если нет - repr)
        return '__str__ %g' % self.var

mc = MyClass(2)
print (mc)
print (MyClass)
help(mc)
str(mc)
mc
```

```
__str__ 2

<class '__main__.MyClass'>

Help on MyClass in module __main__ object:

class MyClass(builtins.object)
 |  __comment__
 |
 |  Methods defined here:
 |
 |  __init__(self, var)
 |      Initialize self. |
 |  __repr__(self)
 |      Return repr(self).
 |
 |  __str__(self)
 |      Return str(self).
 |
 |  ...

__str__ 2

__repr__ 2
```

Списковые включения (List Comprehensions)
= генераторы списков != генераторы

```
[(i, j) for i in range(3) for j in range (5) if i > j]
[(1, 0), (2, 0), (2, 1)]

[x**2 for x in range(5)]
[0, 1, 4, 9, 16]

# + zip
['%s=%s' % (x, y) for y, x in zip(range(5), 'abcde')]
['a=0', 'b=1', 'c=2', 'd=3', 'e=4']
```

Set Comprehensions

```
lst = [10, 5, 100, 3, 20, 10, 3, 20]
{x for x in lst if 10 * round(x / 10) == x}
{10, 20, 100}
```

Dictionary Comprehensions

```
{x: y for y, x in zip(range(5), 'abcde') if y < 3}
{'a': 0, 'b': 1, 'c': 2}
```

Zip (это итератор)

```
x = range(5)
y = 'abcde'
z = [0, 1, 0, 1, 0]

list(zip(x, y, z)) # list - python3
```

```
[(0, 'a', 0), (1, 'b', 1), (2, 'c', 0), (3, 'd', 1), (4, 'e', 0)]
```

Задача: по строке сформировать перечень пар соседних букв

```
x = 'Привет!'
list(zip(x, x[1:])) # можно подавать разные по длине аргументы!
```

```
[('П', 'р'), ('р', 'и'), ('и', 'в'), ('в', 'е'), ('е', 'т'),
('т', '!')]
```

Пишем свой итератор

```
class Fibonacci:
    """
    Итератор последовательности
    Фибоначчи до N
    """

    def __init__(self, N):
        self.n, self.a, self.b, self.max = 0, 0, 1, N

    def __iter__(self):
        return self

    # должна быть такая функция
    def __next__(self): # Python 2: def next(self)
        if self.n < self.max:
            a, self.n, self.a, self.b = self.a, self.n+1, self.b, self.a+self.b
            return a
        else:
            raise StopIteration

list(Fibonacci(10))
```

```
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
```

Генератор

– упрощённый итератор

сравните…

```
class Repeater:
    def __init__(self, value):
        self.value = value
    def __iter__(self):
        return self
    def __next__(self):
        return self.value
```

```
def repeater(value):
    while True:
        yield value
```

Итератор – объект, который имеет метод `__next__` (`next` в Python2)

Генератор – функция, в которой есть yield-выражение

генератор ⇒ итератор

Простые генераторы

помогают делать ленивые вычисления (lazy computations)

```
def Fib(N):
    a, b = 0, 1
    for i in range(N):
        yield a # вместо return
        # для выдачи след. значения
        a, b = b, a + b

for i in Fib(10):
    print (i)
```

```
0
1
1
2
3
5
8
13
21
34
```

`yield` - похожа на `return`,
но работа функции приостанавливается и выдаётся значение

```
def f():
    yield 1
    yield 2
    yield 3
    # return 10 - нет эффекта

def g():
    # взять выход у f!
    x = yield from f()
    yield 4
    yield 5
    yield 6
    # return 100 - нет эффекта

list(g())
```

```
[1, 2, 3, 4, 5, 6]
```

Простые генераторы

```
def double_numbers(iterable):
    for i in iterable:
        yield i + i

list(double_numbers(range(5)))
```
```
[0, 2, 4, 6, 8]
```

генераторное выражение (Generator Expressions)

```
# это список
print ( [x * x for x in range(5)] )
```
```
[0, 1, 4, 9, 16]
```
```
# а это - генераторное выражение
print ( (x * x for x in range(5)) )
```
```
<generator object <genexpr> at
0x00000000046F6BA0>
```
```
# тоже генераторое выражение
print ( sum(x * x for x in range(5)) )
```
```
30
```

Генератор нельзя переиспользовать

```
gen = (x*x for x in range(5))
print ('использование генераторного выражения:')
for y in gen:
    print (y)

# ничего не будет!
print ('переиспользование:')
for y in gen:
    print (y)
```
```
использование генераторного выражения:
0
1
4
9
16
переиспользование:
```

Цепочки декораторов: декораторов может быть много!

```
def square(f): # на вход – функция
    # выход - функция,
    # которая будет реально выполняться
    return lambda x: f(x * x)

def add1(f): # на вход – функция
    # выход - функция,
    # которая будет реально выполняться
    return lambda x: f(x + 1)

# два декоратора у функции
@square
@add1
def time2(x):
    return (x * 2)

time2(3) # (3*3 + 1)*2
```

```
def makebold(fn):
    def wrapped():
        return "<b>" + fn() + "</b>"
    return wrapped

def makeitalic(fn):
    def wrapped():
        return "<i>" + fn() + "</i>"
    return wrapped

@makebold
@makeitalic
def hello():
    return "hello world"

print(hello())
```

```
<b><i>hello world</i></b>
```

Пройдитесь по декораторам «сверху вниз»!

Здесь – они готовят данные для функции

Модули

модуль – один файл с расширением *.py (сейчас уже и zip-архив), задаёт своё пространство имён

пакет – директория, в которой есть файл __init__.py (просто для организации кода). Может содержать поддиректории. Пользователю не так важно, с чем работать

```
import datetime # импортируем модуль

# появляется объект с этим названием
datetime.date(2004, 11, 20)
2004-11-20
print (datetime.__name__) # имя
datetime
print (datetime.__doc__) # описание
Fast implementation of the datetime type.
print (datetime.__file__) # файл
C:\Anaconda3\lib\datetime.py
```

```
pak1
|-- __init__.py
|--pak12
|   |-- __init__.py
|   |-- f.py
|--h.py

from pak1.pak12 import f
```

Модули

можно назначать синонимы модулей и функций

```
import datetime as dt # сокращение имени модуля
print (dt.date(2004, 11, 20))
```
```
2004-11-20
```

```
# импортирование конкретной функции
from datetime import date as dt
print (dt(2004, 11, 20))
```
```
2004-11-20
```

не рекомендуется такой импорт

```
from datetime import *
```

здесь питон ищет модули

```
import sys
sys.path
```
```
['',
 'C:\\Users\\Александр Дьяконов\\Anaconda3\\python35.zip',
 'C:\\Users\\Александр Дьяконов\\Anaconda3\\DLLs',
 'C:\\Users\\Александр Дьяконов\\Anaconda3\\lib',
 'C:\\Users\\Александр Дьяконов\\Anaconda3',
```

Модули: перезагрузка

```
reload(module) # Python 2.x

importlib.reload # >=Python 3.4
imp.reload # Python 3.0 - 3.3
```

Несколько раз использовать import бесполезно!

Вычисления eval

```
a = 1
b = 2
c = eval('a + b') # вычисление выражений
# eval('c = a + b') # нельзя так
```

Объектно-ориентированное программирование (ООП)

```
# первый аргумент всех методов - экземпляр класса
class MyGraph:
    def __init__(self, V, E): # конструктор (деструктор - __del__)
        self.vertices = set(V)
        self.edges = set(E)
    def add_vertex(self, v): # метод - функция, объявленная в теле класса
        self.vertices.add(v)
    def add_edge(self, e):
        self.vertices.add(e[0])
        self.vertices.add(e[1])
        self.edges.add(e)
    def __str__(self): # представление в виде строки
        return ("%s; %s" % (self.vertices, self.edges))

g = MyGraph([1, 2, 3], [(1, 2), (1, 3)])
g.add_edge((3, 4))
print (g)
print (g.vertices)
print (g.__getattribute__('vertices'))
g.__setattr__('vertices',
              set([1, 2, 3, 4, 5]))
print (g)
```

```
{1, 2, 3, 4}; {(1, 2), (1, 3), (3, 4)}
{1, 2, 3, 4}
{1, 2, 3, 4}
{1, 2, 3, 4, 5}; {(1, 2), (1, 3), (3, 4)}
```

Что есть ещё…

сопрограмма

декоратор

исключения

менеджер контекста

особенности хранения объектов

объектно-ориентированное программирование (ООП)

дескриптор

свойство

Хорошие материалы

- **Bruce Eckel** Python 3 Patterns, Recipes and Idioms
- **Никита Лесников** Беглый обзор внутренностей Python // slideshare
- **Сергей Лебедев** Лекции по языку Питон // youtube, канал "Computer Science Center" очень хороший курс
- **Learn X in Y minutes** https://learnxinyminutes.com/docs/python/
- **Роман Сузи** Язык программирования Python // НОУ Интуит
- http://stackoverflow.com
- **Jake VanderPlas** A Whirlwind Tour of Python, 2016 O'Reilly Media Inc. 98 p. для новичков
- **Лучано Рамальо** «Python. К вершинам мастерства» для профи
- **Дэн Бейдер** Чистый Python. Тонкости программирования для профи тонкости программирования

Pandas: начало работы

```
import pandas as pd

pd.set_option("display.width", 40) # будет полезно при демострации дата-фреймов
```

Загрузка данных

```
# Excel
data2 = pd.read_excel('D:\\filename.xlsx', sheetname='1')
titanic.to_excel("titanic.xlsx", sheet_name="passengers", index=False)

# csv-файл
data = pd.read_csv('D:\\filename.csv', sep=';', decimal=',')
data.to_csv('foo.csv') # сохранение

# HDF5
pd.read_hdf('foo.h5', 'df')
df.to_hdf('foo.h5', 'df') # сохранение
```

После загрузки – 1. Смотрим на данные

```
datatrain = pd.read_csv('D:\\Competitions\\Rossman\\train.csv')
datatrain.head(3) # [:3]
```

	Store	DayOfWeek	Date	Sales	Customers	Open	Promo	StateHoliday	SchoolHoliday
0	1	5	2015-07-31	5263	555	1	1	0	1
1	2	5	2015-07-31	6064	625	1	1	0	1
2	3	5	2015-07-31	8314	821	1	1	0	1

В ноутбуке `print (datatrain[:3])` **смотрится хуже**

начало – `head`

конец – `tail`

случайная подвыборка – `sample`

конкретные индексы – `take`

2. Приводим данные к нужным типам

```
datatrain.Date = pd.to_datetime(datatrain.Date)
```

Основные объекты в Pandas

1. Серия (1D)

```
s = pd.Series(np.random.randn(5), index=['a', 'b', 'c', 'd', 'e'])
```

```
s
a    1.321250
b    0.365307
c    0.709577
d    0.542710
e   -0.212721
dtype: float64
```

Похоже на словарь:

```
print s['b']
0.365307109524

print s.get('z', 'error')
error
```

Автоматическое выравнивание по индексу

```
print s + s[1:]
a         NaN
b    0.730614
c    1.419154
d    1.085419
e   -0.425441
```

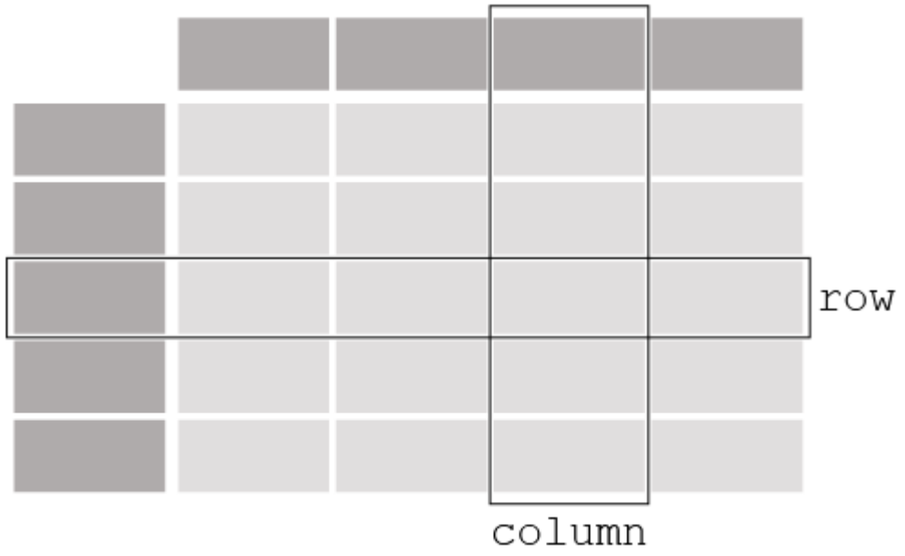
Основные объекты в Pandas

2. ДатаФрейм (2D) – набор серий с одним индексом

```
df = pd.DataFrame(np.random.randn(8, 3),
                  index=pd.date_range('1/1/2000', periods=8),
                  columns=['A', 'B', 'C'])
```

```
df
```

	A	B	C
2000-01-01	0.684918	0.240427	-0.030283
2000-01-02	0.533952	-0.573713	-1.602537
2000-01-03	-1.291314	-0.650594	1.771561
2000-01-04	2.813297	-1.093390	-0.209462
2000-01-05	0.894795	-0.574468	0.765031
2000-01-06	1.513772	0.618505	-1.402341
2000-01-07	-0.435267	-1.199286	0.990490
2000-01-08	-0.541890	0.590653	-0.530153



Есть ещё панели и многомерные объекты…

Создание ДатаФрейма

```
# первый способ
data = pd.DataFrame({ 'A' : [1., 4., 2., 1.],
'B' : pd.Timestamp('20130102'),
'C' : pd.Series(1,index=list(range(4)),dtype='float32'),
'D' : np.array([3] * 4,dtype='int32'),
'E' : pd.Categorical(["test","train","test","train"]),
'F' : 'foo' }, index=pd.period_range('Jan-2000', periods=4, freq='M'))

print (data) # но так будет некрасиво
```

	A	B	C	D	E	F
2000-01	1.0	2013-01-02	NaN	3	test	foo
2000-02	4.0	2013-01-02	NaN	3	train	foo
2000-03	2.0	2013-01-02	NaN	3	test	foo
2000-04	1.0	2013-01-02	NaN	3	train	foo

Статистика по признакам

```
# типы
print (data.dtypes)
A            float64
B    datetime64[ns]
C            float32
D              int32
E             object
F             object
dtype: object
```

Изменение типа: .astype()

```
# статистика + транспонирование
print (data.describe().T) # транспонирование часто удобно!
   count  mean       std  min  25%  50%  75%  max
A      4     2  1.414214    1    1  1.5  2.5    4
C      0   NaN       NaN  NaN  NaN  NaN  NaN  NaN
D      4     3  0.000000    3    3  3.0  3.0    3
```

Совет: смотрите на число уникальных элементов .nunique()

```
# число уникальных элементов (можно через describe)
for i in data.columns: # можно просто data
    print str(i) + ':' + str(data[i].nunique())
```

```
A:3
B:1
C:0
D:1
E:2
F:1
```

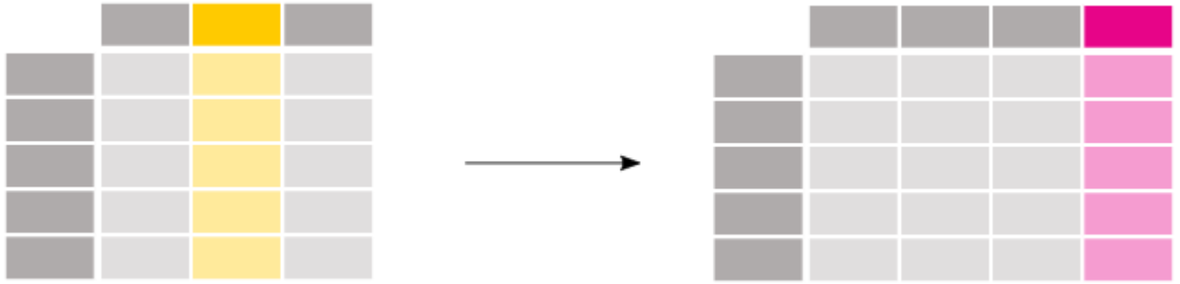
Индексация

Operation	Syntax	Result
Select column	`df[col]`	Series
Select row by label	`df.loc[label]`	Series
Select row by integer location	`df.iloc[loc]`	Series
Slice rows	`df[5:10]`	DataFrame
Select rows by boolean vector	`df[bool_vec]`	DataFrame

Индексация

```
data.at['2000-01','A'] = 10. # по названию
data.iat[0, 1] = pd.Timestamp('19990101') # по номеру
# просто = '1999/01/01' не работает

data.loc['2000-01':'2000-02',['D','B','A']] # по названию
data.iloc[0:2, 1:3] # по номеру


# выбор с проверкой на вхождение
data[data['E'].isin(['test','valid'])] # полезно: isin
```

Создание столбца



```
air_quality["london_mg_per_cubic"] =
```

удаление

```
del air_quality["london_mg_per_cubic"]
```

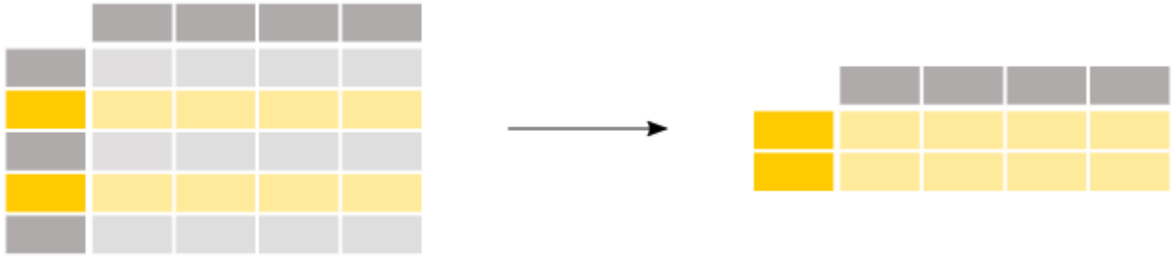
универмально, можно удалять несколько + строки

```
df.drop(["a", "d"], axis=0)
```

удаление с присвоением

```
y = air_quality.pop("london_mg_per_cubic")
```

Логические условия

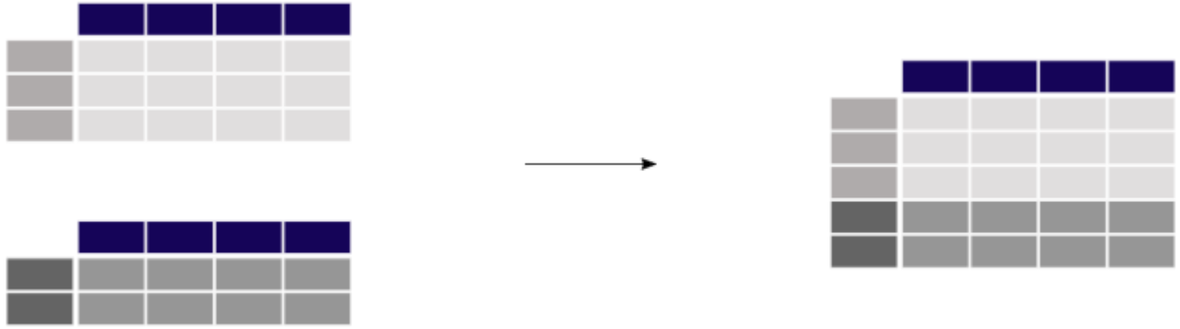


```
above_35 = titanic[titanic["Age"] > 35]
class_23 = titanic[titanic["Pclass"].isin([2, 3])] # эквивалентно -
class_23 = titanic[(titanic["Pclass"] == 2) | (titanic["Pclass"] == 3)]
age_no_na = titanic[titanic["Age"].notna()] # выбрать известные значения
```

Объединение ДатаФреймов

```
# объединение дата-фреймов
left = pd.DataFrame({'key': [1,2,1], 'l': [1, 2, 3]})
right = pd.DataFrame({'key': [1,2,3], 'r': [4, 5, 6]})
print left
print right
pd.merge(left, right, on='key')
```

Вертикальная конкатенация ДатаФреймов



```
a = pd.DataFrame(dict([('A',[1., 3., 2., 1.]), ('B',[2.2, 1.1, 3.3, 0.0]), ('C', 1)]))
b = pd.DataFrame(dict([('A',[0., 2.]), ('B',4)]))
```

Первый способ

```
a.append(b)
```

Второй способ

```
pd.concat([a, b])
```

Можно и горизонтально

```
pd.concat([a, b], keys=['a', 'b'], axis=1)
```

Портятся индексы

	A	B	C
0	1.0	2.2	1.0
1	3.0	1.1	1.0
2	2.0	3.3	1.0
3	1.0	0.0	1.0
0	0.0	4.0	NaN
1	2.0	4.0	NaN

Группировка

Функция `.groupby()` :

1. **Разделение данных на группы (по некоторому критерию)**
2. **Применение к каждой группе функции**
3. **Получение результата**

Функция

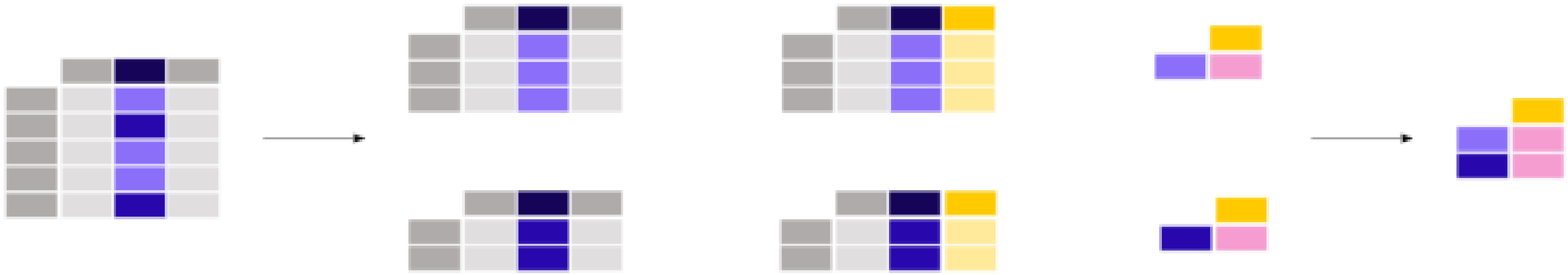
2.1. Агрегация (статистика по группе)

2.2. Трансформация (изменение/формирование значений по группе)

2.3. Фильтрация (удаление некоторых групп)

Группировка

```
titanic      .groupby("Sex")      ["Age"]      .mean()
```



Группировка

Для каждого уникального значения A найти минимальный B

```
d = pd.DataFrame({'A': [1,2,2,1,3,3], 'B': [1,2,3,3,2,1]})
print (d)

# первый способ
print (d.loc[d.groupby('A')['B'].idxmin()])

# второй способ - чтобы A не был индексом
print (d.sort_values(by='B').groupby('A', as_index=False).first())
```

```
   A  B
0  1  1
1  2  2
2  2  3
3  1  3
4  3  2
5  3  1
```

```
   A  B
0  1  1
1  2  2
5  3  1
```

```
   A  B
0  1  1
1  2  2
5  3  1
```

Агрегация

	A	B	C
0	1	3	5
1	2	4	5
2	2	3	5
3	1	4	6
4	1	3	6
5	2	3	6
6	2	4	6

```
# агрегация по разным столбцам
print (a.groupby('A').agg({'B':np.sum, 'C':np.mean}))
```

```
          C   B
A
1  5.666667  10
2  5.500000  14
```

Трансформация

	A	B	C
0	1	3	5
1	2	4	5
2	2	3	5
3	1	4	6
4	1	3	6
5	2	3	6
6	2	4	6

```
mmean = lambda x: (x-np.mean(x))
print (a.groupby('A').transform(mmean))
```

```
          B         C
0 -0.333333 -0.666667
1  0.500000 -0.500000
2 -0.500000 -0.500000
3  0.666667  0.333333
4 -0.333333  0.333333
5 -0.500000  0.500000
6  0.500000  0.500000
```

Apply – пример нормировки

```
a.apply(lambda x: x/sum(x))
# по столбцам
```

	A	B	C
0	0.090909	0.125000	0.128205
1	0.181818	0.166667	0.128205
2	0.181818	0.125000	0.128205
3	0.090909	0.166667	0.153846
4	0.090909	0.125000	0.153846
5	0.181818	0.125000	0.153846
6	0.181818	0.166667	0.153846

```
a.apply(lambda x: x/sum(x), axis=1)
# по строкам
```

	A	B	C
0	0.111111	0.333333	0.555556
1	0.181818	0.363636	0.454545
2	0.200000	0.300000	0.500000
3	0.090909	0.363636	0.545455
4	0.100000	0.300000	0.600000
5	0.181818	0.272727	0.545455
6	0.166667	0.333333	0.500000

`pipe()` – к ДатаФреймам

`apply()` – к строкам/столбцам

`applymap()` – поэлементно

Map – основное применение

```
df = pd.DataFrame({'CITY': [u'London', u'Moscow', u'Paris'], 'Stats': [0,2,1]})

d = {u'London':u'GB', u'Moscow':u'RUS', u'Paris':u'FR'}
df['country'] = df['CITY'].map(d)
df.columns = map(str.lower, df.columns)
df
```

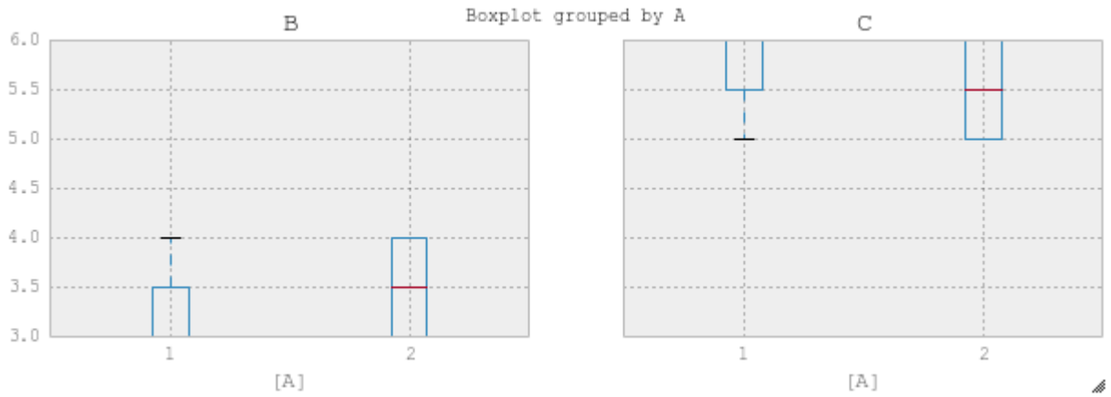
	city	stats	country
0	London	0	GB
1	Moscow	2	RUS
2	Paris	1	FR

Иногда есть другие средства – замена значений

```
df.replace(u'Moscow', u'Ufa') # замена значения
```

Рисование

```
a.boxplot(by='A') # a.groupby('A').boxplot()
```



Иерархическая (многоуровневая) индексация

```
tuples = list(zip(*[['bar', 'bar', 'baz', 'baz', 'foo', 'foo', 'qux', 'qux'],
 ['one', 'two', 'one', 'two','one', 'two', 'one', 'two']]))
print (tuples)
[('bar', 'one'), ('bar', 'two'), ('baz', 'one'), ('baz', 'two'), ('foo', 'one'),
('foo', 'two'), ('qux', 'one'), ('qux', 'two')]

index = pd.MultiIndex.from_tuples(tuples, names=['first', 'second'])
df = pd.DataFrame(np.random.randn(8, 2), index=index, columns=['A', 'B'])
print (df)
                     A         B
first second
bar   one     -0.240469 -0.533312
      two     -0.847305  0.845316
baz   one      0.274592  0.473476
      two      1.433575 -0.977992
foo   one      0.957252 -1.246396
      two     -2.821039 -0.625924
qux   one      0.086683 -0.450850
      two     -1.236494  0.706156
```

Иерархическая (многоуровневая) индексация

```
print (df.stack()) # обратная операция unstack()
```

```
first  second
bar    one     A   -0.240469
               B   -0.533312
       two     A   -0.847305
               B    0.845316
baz    one     A    0.274592
               B    0.473476
       two     A    1.433575
               B   -0.977992
foo    one     A    0.957252
               B   -1.246396
       two     A   -2.821039
               B   -0.625924
qux    one     A    0.086683
               B   -0.450850
       two     A   -1.236494
               B    0.706156
dtype: float64
```

stack

df2

		A	B
first	second		
bar	one	1	2
	two	3	4
baz	one	5	6
	two	7	8

MultiIndex

stacked = df2.stack()

first	second		
bar	one	A	1
		B	2
	two	A	3
		B	4
baz	one	A	5
		B	6
	two	A	7
		B	8

MultiIndex

unstack

stacked

first	second		
bar	one	A	1
		B	2
	two	A	3
		B	4
baz	one	A	5
		B	6
	two	A	7
		B	8

MultiIndex

stacked.unstack()

		A	B
first	second		
bar	one	1	2
	two	3	4
baz	one	5	6
	two	7	8

MultiIndex

Pivot

df

	foo	bar	baz	zoo
0	one	A	1	x
1	one	B	2	y
2	one	C	3	z
3	two	A	4	q
4	two	B	5	w
5	two	C	6	t

```
df.pivot(index='foo',
         columns='bar',
         values='baz')
```

bar	A	B	C
foo			
one	1	2	3
two	4	5	6

Melt

df3

	first	last	height	weight
0	John	Doe	5.5	130
1	Mary	Bo	6.0	150

```
df3.melt(id_vars=['first', 'last'])
```

	first	last	variable	value
0	John	Doe	height	5.5
1	Mary	Bo	height	6.0
2	John	Doe	weight	130
3	Mary	Bo	weight	150

Pivot tables

```
df = pd.DataFrame({'ind1':[1,1,1,2,2,2,2], 'ind2':[1,1,2,2,3,3,2],
'x':[1,2,3,4,5,6,7], 'y':[1,1,1,1,1,1,2]})

print (df)
print (df.pivot(index='x', columns='ind2', values='y'))
```

```
   ind1  ind2  x  y
0     1     1  1  1
1     1     1  2  1
2     1     2  3  1
3     2     2  4  1
4     2     3  5  1
5     2     3  6  1
6     2     2  7  2
```

```
ind2   1   2   3
x
1      1 NaN NaN
2      1 NaN NaN
3    NaN   1 NaN
4    NaN   1 NaN
5    NaN NaN   1
6    NaN NaN   1
7    NaN   2 NaN
```

Pandas: ещё преимущества

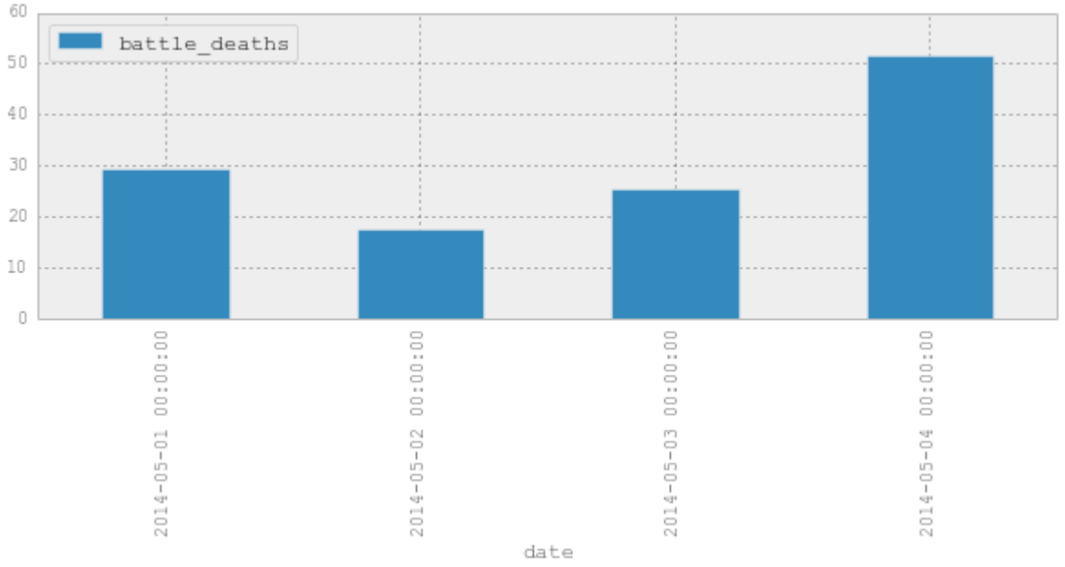
поддержка категориальных признаков

поддержка даты / времени

поддержка строк

Временные ряды

```
# переход к дням и визуализация
print (df.resample('D', how='mean').plot(kind='bar'))

# пересортировка df
df.sort_index(by = 'battle_deaths', inplace=True)
df
```



	battle_deaths
date	
2014-05-02 18:47:05.280592	14
2014-05-02 18:47:05.230071	15
2014-05-02 18:47:05.230071	15
2014-05-01 18:47:05.119994	25
2014-05-03 18:47:05.385109	25
2014-05-02 18:47:05.178768	26
2014-05-03 18:47:05.332662	26
2014-05-01 18:47:05.069722	34
2014-05-04 18:47:05.486877	41
2014-05-04 18:47:05.436523	62

n

Строки

Пример возможного извлечения признаков

```
lst = ['mark 10 12-10-2015', 'also 7 10-10-2014', 'take 2 01-05-2015']
df = pd.DataFrame({'x':lst})
df['num'] = df.x.str.extract('(\d+)')
df['date'] = df.x.str.extract('(..-..-....)')
df['word'] = df.x.str.extract('([a-z]\w{0,})')
df
```

	x	num	date	word
0	mark 10 12-10-2015	10	12-10-2015	mark
1	also 7 10-10-2014	7	10-10-2014	also
2	take 2 01-05-2015	2	01-05-2015	take

Как часто встречаются пары значений

очень полезная штука!

```
d = pd.DataFrame({'A': [1,2,2,1,2,3,2,1,3],
                  'B': [1,2,3,4,1,2,3,3,4]})

pd.crosstab(d['A'], d['B'])
```

B	1	2	3	4
A				
1	1	0	1	1
2	1	1	2	0
3	0	1	0	1

Другие возможности

Удаление дубликатов

```
df = pd.DataFrame({'name': ['Al', 'Max',  'Al'],
                   'surname':[u'Run', u'Crone', u'Run']})
print (df.duplicated())

df.drop_duplicates(['name'], keep='last')
# df.drop_duplicates()
```

```
0    False
1    False
2     True
```

	name	surname
1	Max	Crone
2	Al	Run

dummy-кодирование для категориальных признаков

```
pd.get_dummies([1,2,1,2,3])
```

	1	2	3
0	1	0	0
1	0	1	0
2	1	0	0
3	0	1	0
4	0	0	1

Кодирование категориальных признаков по порядку

```
pd.factorize([20,10,np.nan,10,np.nan,30,20])
```

```
(array([ 0,  1, -1,  1, -1,  2,  0]), array([ 20.,  10.,  30.]))
```

Ссылки

Несколько иллюстраций взято отсюда:

https://jalammar.github.io/visualizing-pandas-pivoting-and-reshaping/

Scikit-Learn – установка

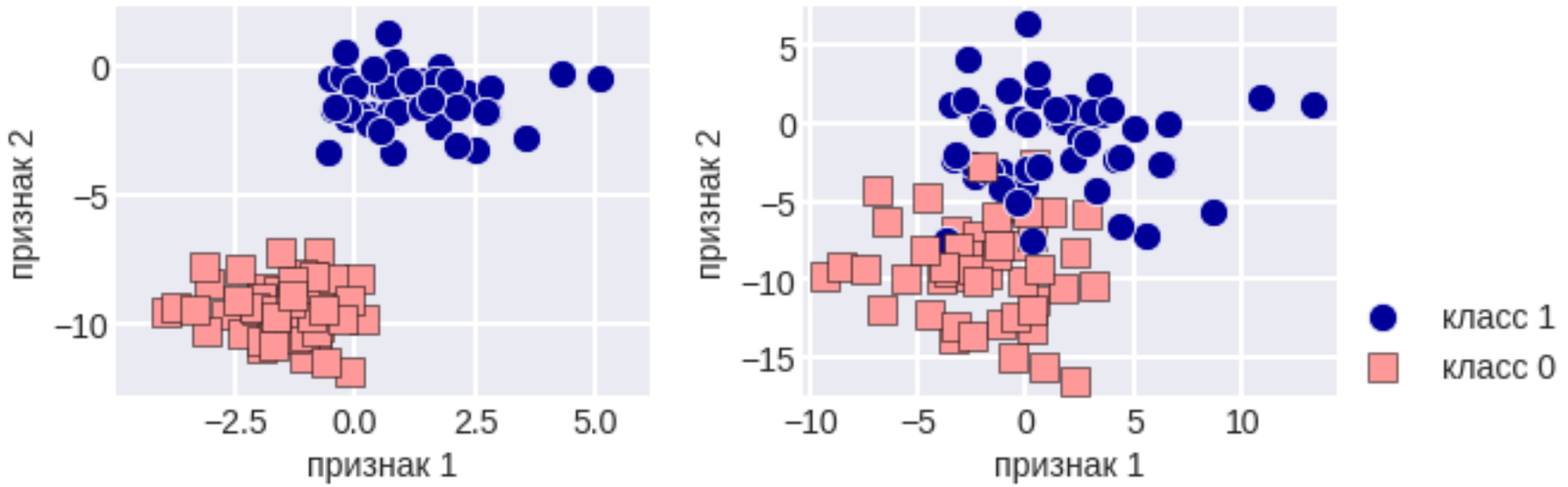
http://scikit-learn.org/stable/install.html

входит во многие дистрибутивы

Подробнее – будут примеры по курсу

Встроенные датасеты (генераторы)

«Кучки»




```
from sklearn.datasets import make_blobs
X, y = make_blobs(centers=2, random_state=2)
plt.scatter(X[:, 0], X[:, 1], c=y, s=75)
```

n_samples, n_features – **размеры**

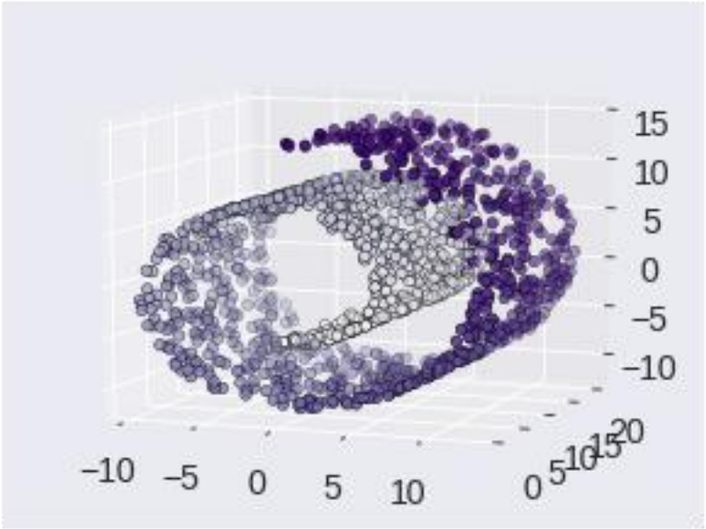
centers – **сколько кучек**

cluster_std – **дисперсия**

random_state – **инициализация генератора**

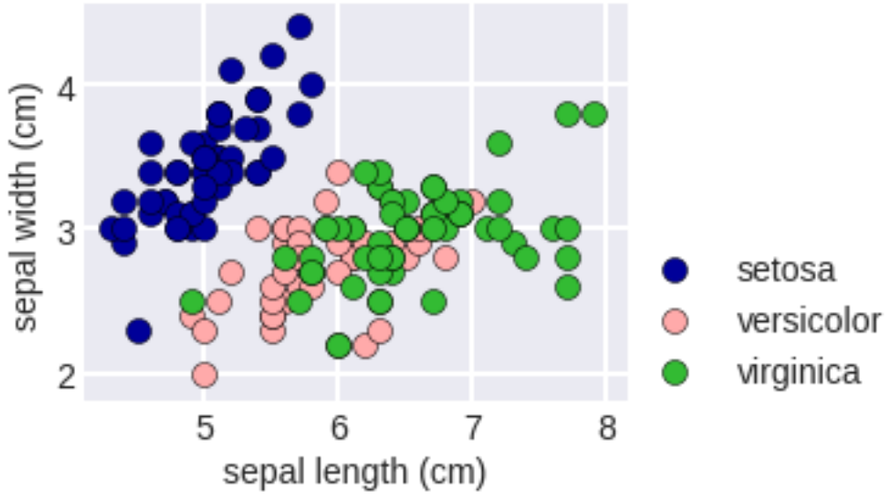
Встроенные датасеты (генераторы)

```
from sklearn.datasets import make_swiss_roll
n_samples = 1500
noise = 0.05
X, y = make_swiss_roll(n_samples, noise)
import mpl_toolkits.mplot3d.axes3d as p3

fig = plt.figure(figsize=(4, 3))
ax = p3.Axes3D(fig)
ax.view_init(10, -70)
ax.scatter(X[:, 0], X[:, 1], X[:, 2],
           s=20, c=y, cmap='Purples',
           edgecolor='k', linewidth=0.5)
```



Встроенные датасеты (загрузчики)

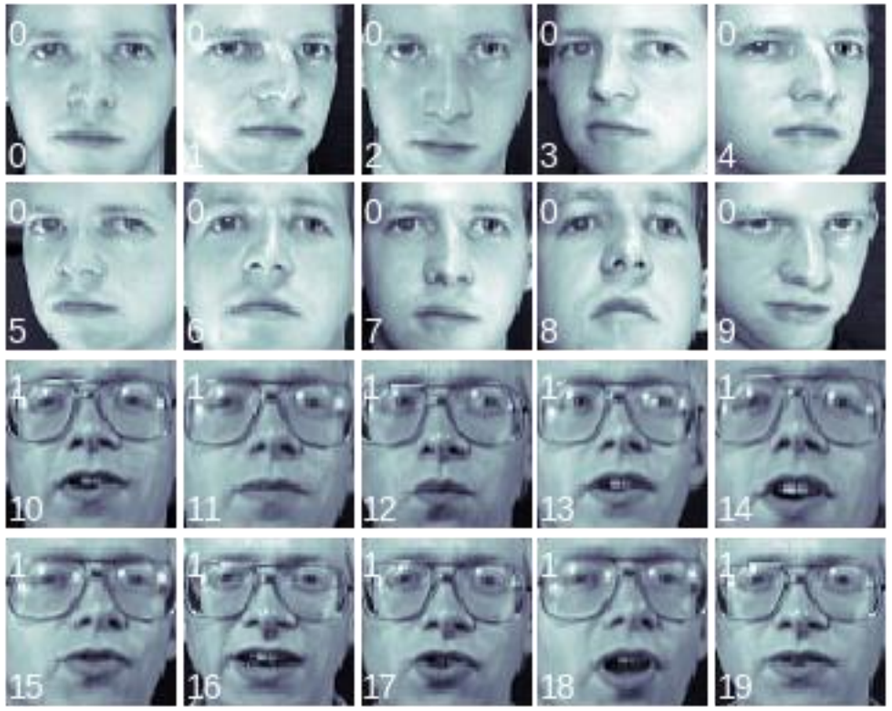
```
from sklearn import datasets
iris = datasets.load_iris()
X_iris, y_iris = iris.data, iris.target
```



	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

Встроенные датасеты (загрузчики)

```
from sklearn.datasets import fetch_olivetti_faces
faces = fetch_olivetti_faces()

print (faces.keys())
print (faces.images.shape)
print (faces.data.shape)
dict_keys(['data', 'images',
           'target', 'DESCR'])
(400, 64, 64)
(400, 4096)

print_faces(faces.images,
            faces.target, 20)

def print_faces(images, target, top_n):
    fig = plt.figure(figsize=(5, 5))
    fig.subplots_adjust(bottom=0, top=1, hspace=0.05, wspace=0.05)
    for i in range(top_n):
        p = fig.add_subplot(5, 5, i + 1, xticks=[], yticks=[])
        p.imshow(images[i], cmap=plt.cm.bone)
        p.text(0, 14, str(target[i]), color='white')
        p.text(0, 60, str(i), color='white')
```



Интерфейсы

У Scikit-learn единый способ использования всех методов.
Для всех моделей (estimator object) доступны следующие методы.

`model.fit()` – **настройка на данные (обучение)**
`model.fit(X, y)` – **для обучения с учителем (supervised learning)**
`model.fit(X)` – **для обучение без учителя (unsupervised learning)**

`model.predict`	`model.transform`
Classification	**Preprocessing**
Regression	**Dimensionality Reduction**
Clustering	**Feature Extraction**
	Feature selection

Для обучения с учителем

`model.predict(X_test)` – **предсказать значения целевой переменной**

`model.predict_proba()` – **выдать «степень уверенности» в ответе (вероятность) – для некоторых моделей**

`model.decision_function()` – **решающая функция – для некоторых моделей**

`model.score()` – **в большинстве моделей встроены методы оценки их качества работы**

`model.transform()` – **для отбора признаков (feature selection) «сжимает» обучающую матрицу. Для регрессионных моделей и классификаторов (linear, RF и т.п.) выделяет наиболее информативные признаки**

Для обучения без учителя

`model.transform()` – **преобразует данные**

`model.fit_transform()` – **не во всех моделях – эффективная настройка и трансформация обучения**

`model.predict()` – **для кластеризации (не во всех моделях) – получить метки кластеров**

`model.predict_proba()` – **Gaussian mixture models (GMMs) получают вероятности принадлежности к компонентам для каждой точки**

`model.score()` – **некоторые модели (KDE, GMMs) получают правдободобие (насколько данные соответствуют модели)**

Работа с моделями (1)

```
# данные
from sklearn.datasets import make_blobs
X, y = make_blobs(centers=2, random_state=0)

# разбивка: обучение - контроль
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    random_state=0)

# обучение модели и предсказание
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression()
classifier.fit(X_train, y_train)
prediction = classifier.predict(X_test)

# качество
print (np.mean(prediction == y_test))                    0.8
print (classifier.score(X_test, y_test)) # более удобная  0.8
print (classifier.score(X_train, y_train))               0.93
```

Работа с моделями (2)

```
# визуализация
plot_2d_separator(classifier, X, fill=True)
plt.scatter(X[:, 0], X[:, 1], c=y, s=70)
```

```
# матрица несоответствий
from sklearn.metrics import confusion_matrix
print (confusion_matrix(y_test, prediction))
```

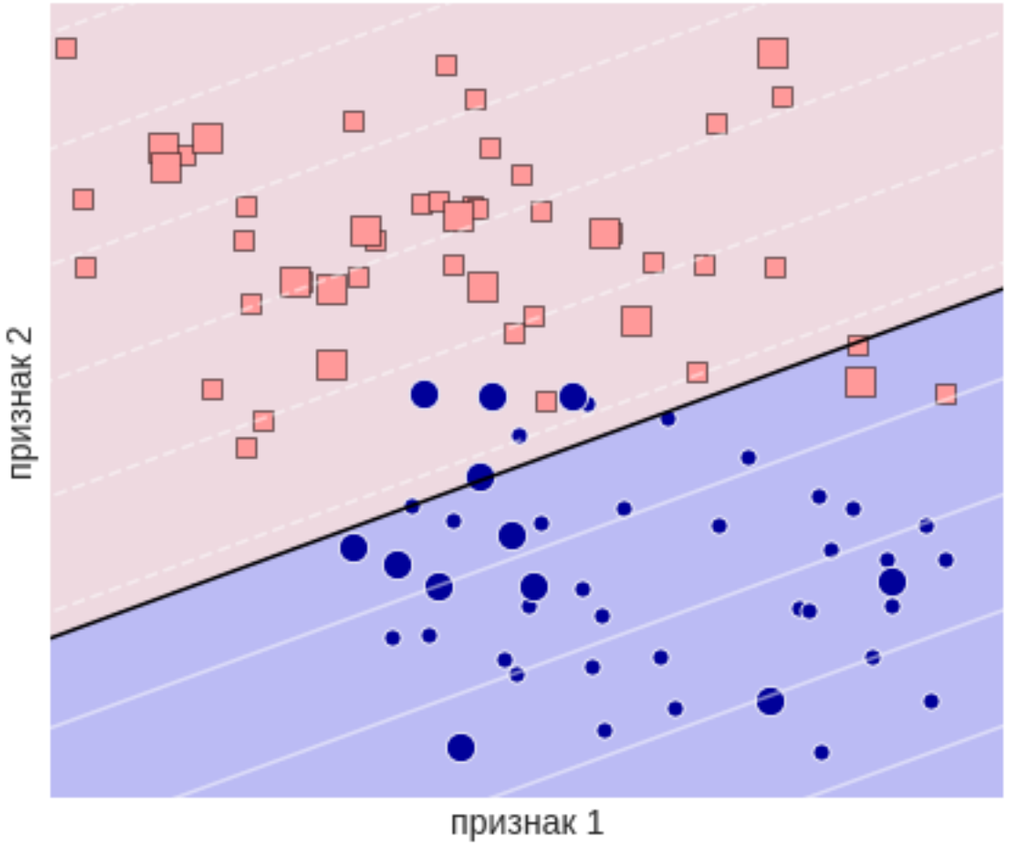
```
[[12  1]
 [ 4  8]]
```

```
# отчёт о точности
from sklearn.metrics import classification_report
print(classification_report(y_test, prediction))
```

```
              precision    recall  f1-score   support

           0       0.75      0.92      0.83        13
           1       0.89      0.67      0.76        12

   micro avg       0.80      0.80      0.80        25
   macro avg       0.82      0.79      0.79        25
weighted avg       0.82      0.80      0.80        25
```

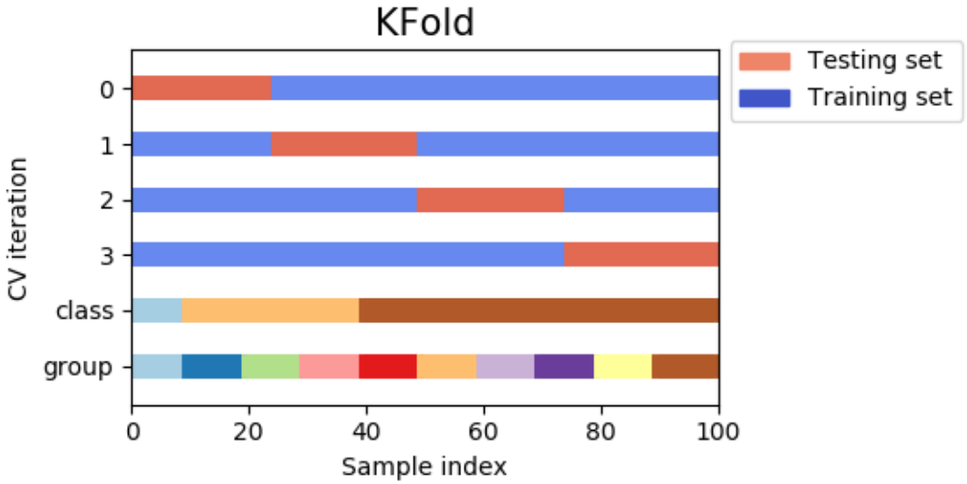
Работа с моделями

Что получилось (логистическая регрессия)



Разбиение выборки при валидации (подробнее – дальше)

```
sklearn.model_selection.KFold(n_splits='warn',
                              shuffle=False,
                              random_state=None)
```




```
from sklearn.model_selection import KFold
kf = KFold(n_splits=2)
kf.get_n_splits(X)
print(kf)
for train_index, test_index in kf.split(X):
    print("TRAIN:", train_index, "TEST:", test_index)
    X_train, X_test = X[train_index], X[test_index]
                      y_train, y_test = y[train_index], y[test_index]
```

Оценка модели (cross_val_score)

```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import ShuffleSplit
from sklearn.linear_model import LogisticRegression

logreg = LogisticRegression()
cv = ShuffleSplit(n_splits=3, test_size=0.1,
                  train_size=None, random_state=None)
cross_val_score(logreg, X, y, cv=cv)
```

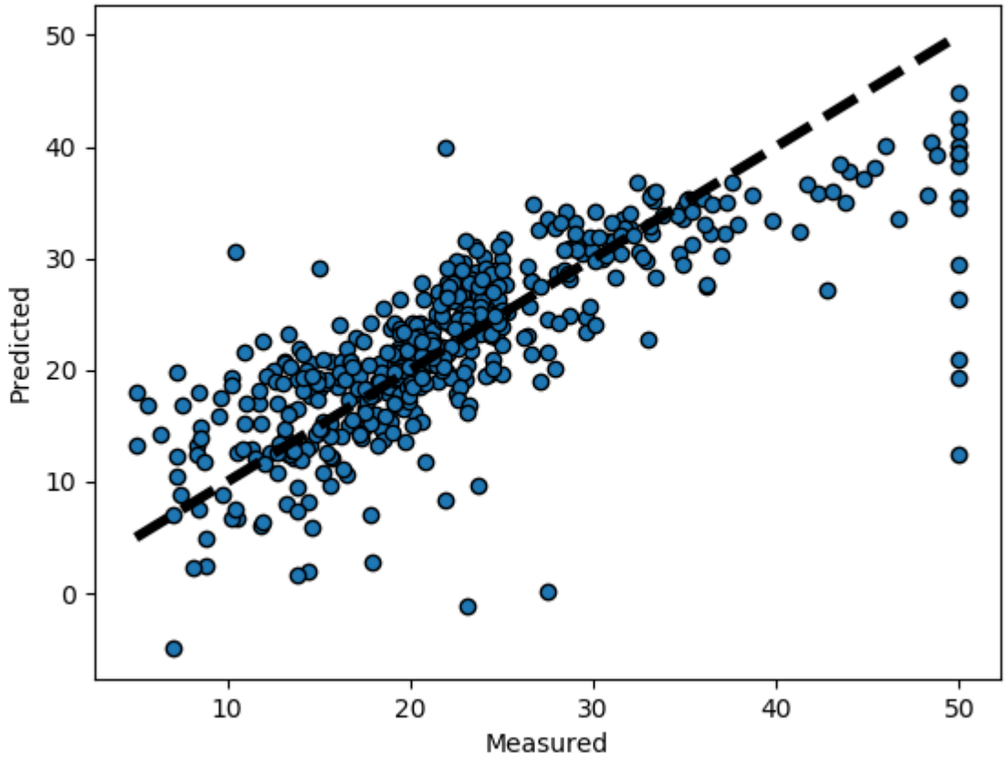
```
array([0.8, 0.8 , 1. ])
```

У этих функций много параметров…

Они (функции) «понимают» друг друга

Пока не указываем скорер – используется встроенный (в модель)

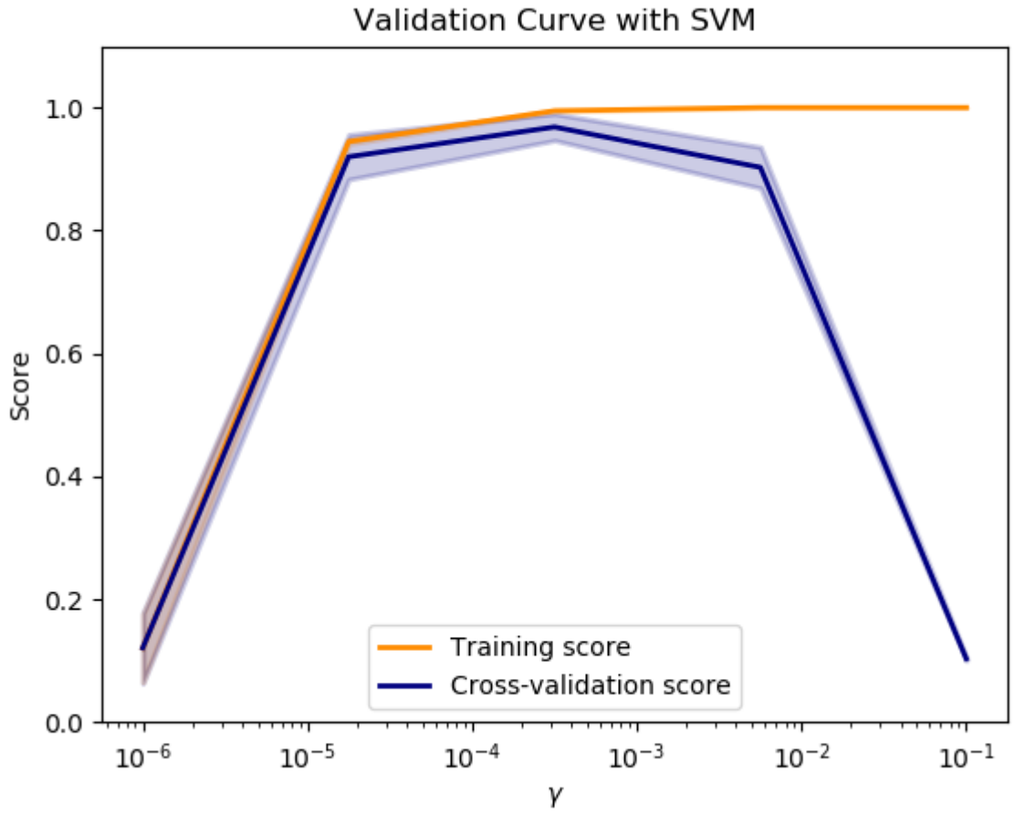
Ответы алгоритма: model_selection.cross_val_predict



Для визуальной оценки работы алгоритма

```
from sklearn.model_selection import cross_val_predict
predicted = cross_val_predict(model, X, y, cv=10)
plt.scatter(y, predicted)
```

Качество при варьировании параметра: `model_selection.validation_curve`



Перебор параметров: model_selection.GridSearchCV

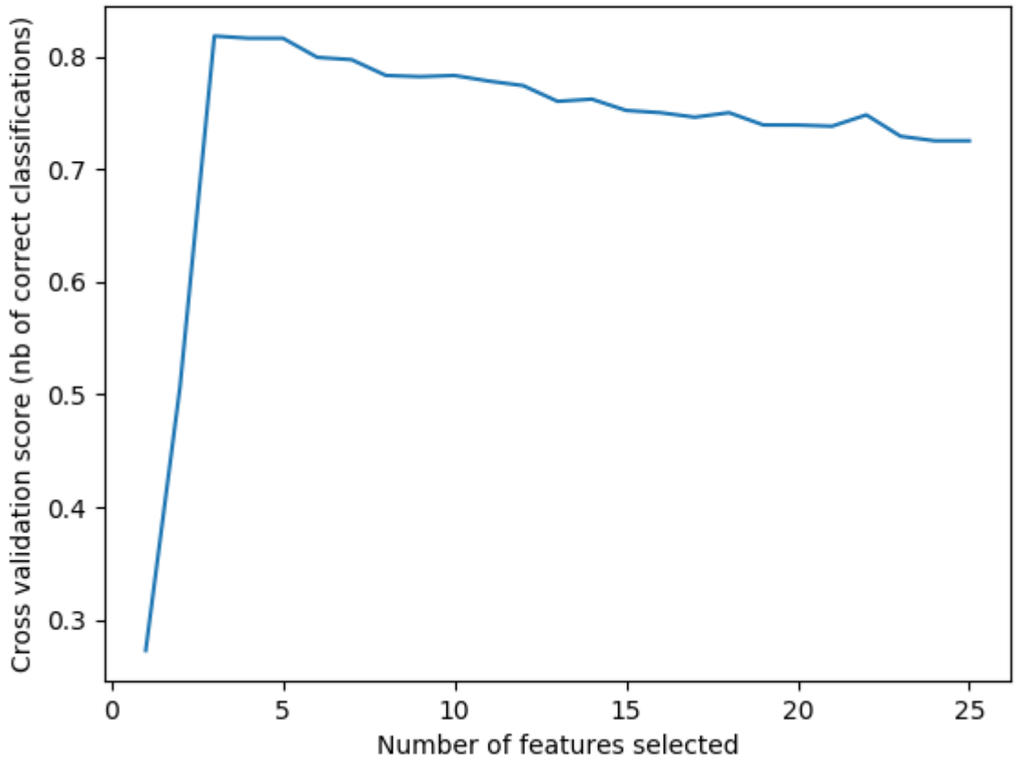
```
from sklearn.model_selection import GridSearchCV
parameters = {'kernel':('linear', 'rbf'), 'C':[1, 10]}
svc = svm.SVC(gamma="scale")
clf = GridSearchCV(svc, parameters, cv=5)
clf.fit(iris.data, iris.target)
```

Результаты перебора в

```
clf.cv_results_
```

Есть также случайный поиск model_selection.RandomizedSearchCV
(тут есть «число итераций»)

Отбор признаков: sklearn.feature_selection



https://scikit-learn.org/stable/auto_examples/feature_selection/plot_rfe_with_cross_validation.html#sphx-glr-auto-examples-feature-selection-plot-rfe-with-cross-validation-py

Предобработка данных: preprocessing

Нормировка данных

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X)
X_scaled = scaler.transform(X)
```

Перенумерация

```
f = ['a', 'bb', 20, 'bb' , 'a', 'a']
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()
encoder.fit(f)
encoder.transform(f)
```

```
array([1, 2, 0, 2, 1, 1], dtype=int64)
```

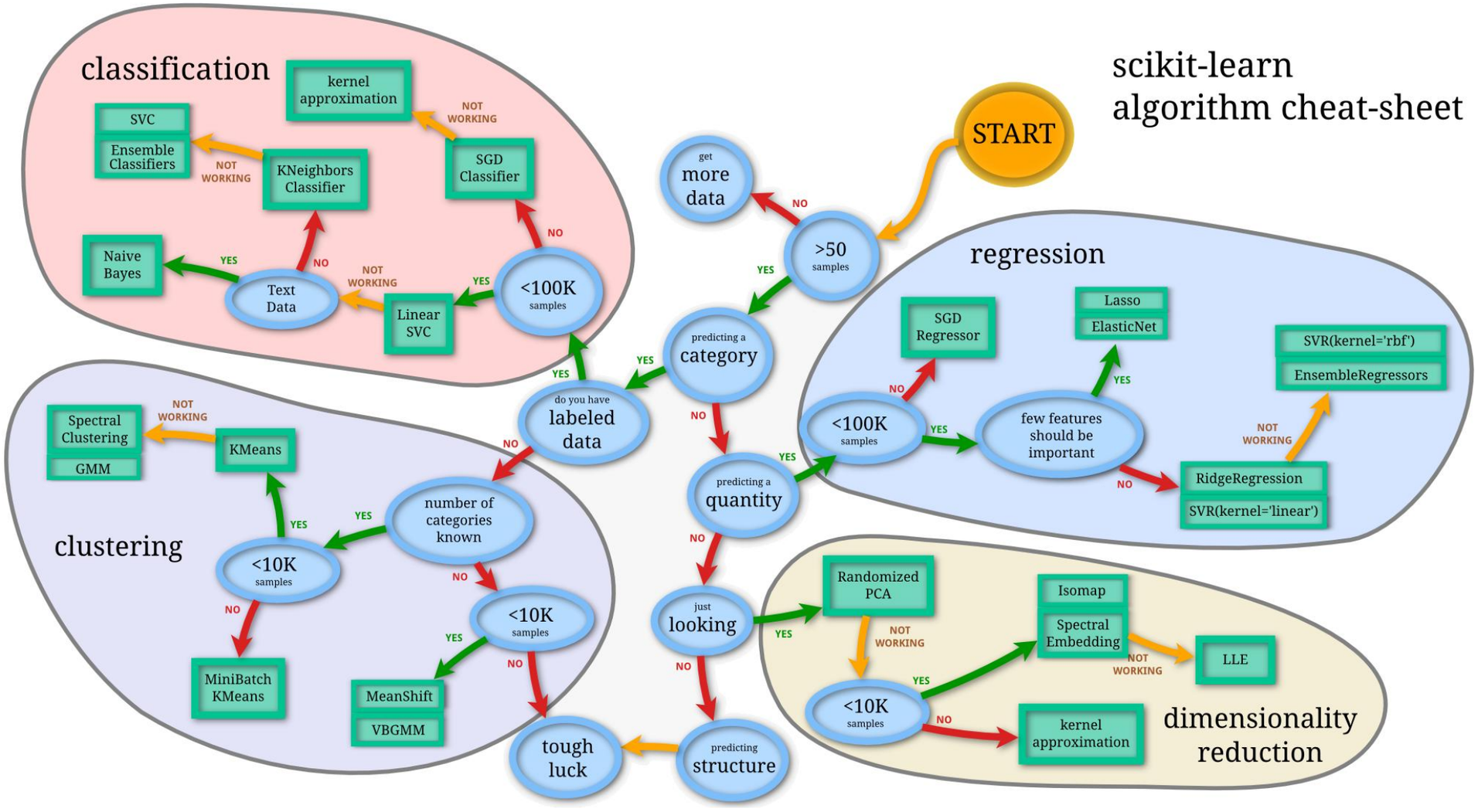
Последовательность операторов: pipeline

```
from sklearn.pipeline import make_pipeline
pipeline = make_pipeline(TfidfVectorizer(), LogisticRegression())
pipeline.fit(text_train, y_train)
pipeline.score(text_test, y_test)
0.5
```

Оптимизация параметров

```
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import make_pipeline
from sklearn.feature_extraction.text import TfidfVectorizer

pipeline = make_pipeline(TfidfVectorizer(), LogisticRegression())

params = {'logisticregression__C': [.1, 1, 10, 100],
          "tfidfvectorizer__ngram_range": [(1, 1), (1, 2), (2, 2)]}
grid = GridSearchCV(pipeline, param_grid=params, cv=5)
grid.fit(X, y)
print(grid.best_params_)
grid.score(X, y)
```

Выбор алгоритма (модели)



Сохранение моделей

```
import pickle
s = pickle.dumps(clf)
clf2 = pickle.loads(s)
```

Ссылки

В данной презентации много примеров взято из ноутбука

https://github.com/amueller/scipy_2015_sklearn_tutorial/tree/master/notebooks

Спасибо Андреасу Мюллеру!

См. API Reference

http://scikit-learn.org/stable/modules/classes.html