Факультет
математики
и компьютерных
наук
СПбГУ

Машинное обучение

Семинар 1. Введение в библиотеки для машинного обучения numpy, pandas

2 сентября 2021

NumPy (https://docs.scipy.org/doc/numpy/user/index.html)

С 1995 numeric, с 2006 NumPy — «Numerical Python extensions» or «NumPy»

Возможности библиотеки NumPy:

- работать с многомерными массивами (таблицами), не отдельными числами
- быстро вычислять математические функций на многомерных массивах. Ядро пакета NumPy — объект ndarray (https://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.html)
- старайтесь минимизировать количество индексаций в своем коде (особенно индексаций целым числом)
- если вы делаете цикл, обращаясь по очереди к каждому элементу массива, почти наверняка ваш код неэффективен и может быть значительно улучшен

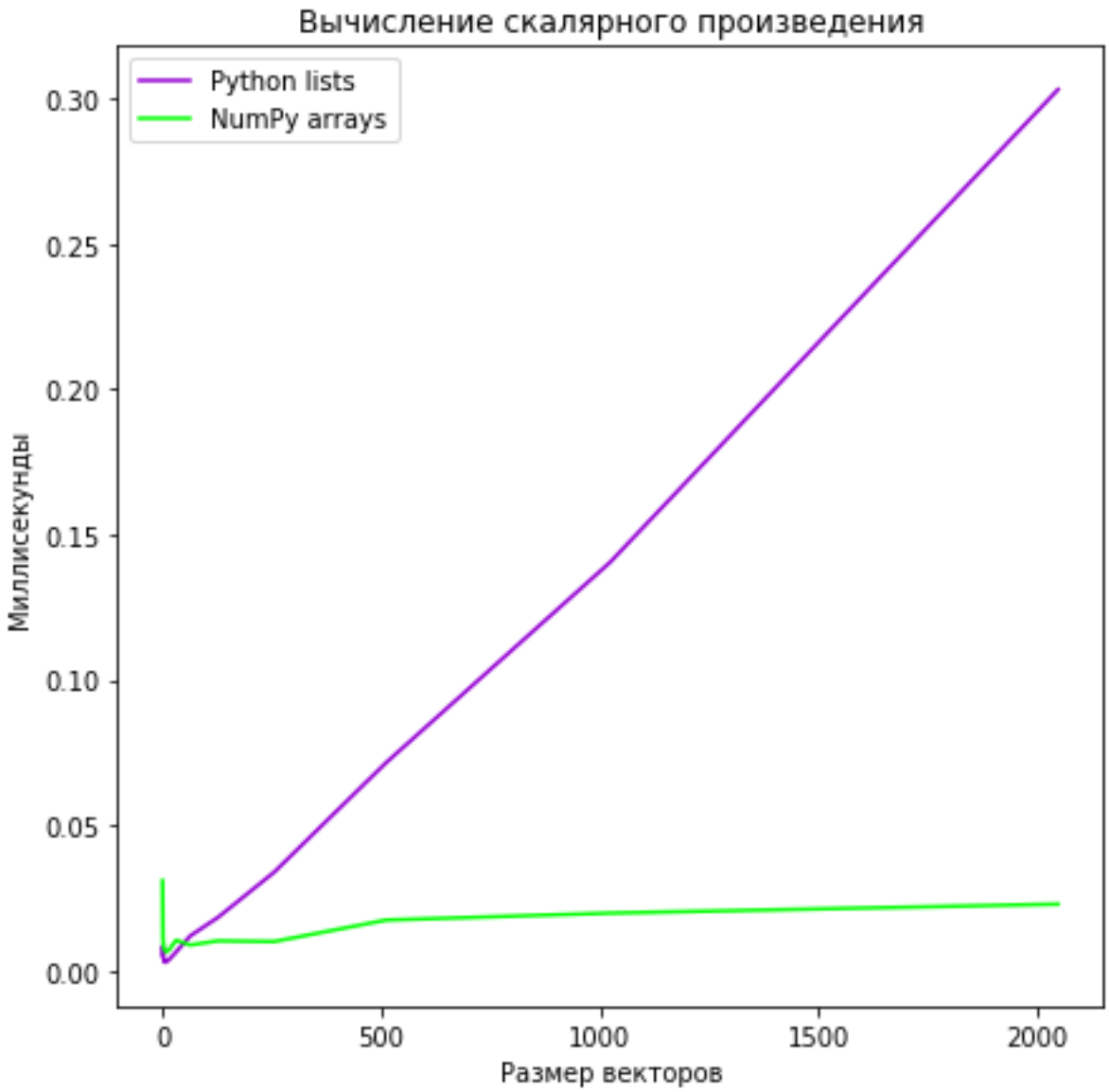
Важные отличия между NumPy arrays и Python sequences:

- NumPy array имеет фиксированную длину, которая определяется в момент его создания (в отличие от Python lists, которые могут расти динамически)
- Элементы в NumPy array должны быть одного типа
- Можно выполнять операции непосредственно над NumPy arrays

Скорость NumPy из-за:

- Реализации на C
- Vectorization and Broadcasting (например, произведение массивов совместимых форм)

Мотивирующий пример



Способы создания Numpy arrays

- Конвертация из Python structures
- Генерация с помощью встроенных функций
- Чтение с диска

Конвертация из Python structures

In [1]:

```
import numpy as np
```

In [2]:

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

Out[2]:

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

При конвертации можно задавать тип данных с помощью аргумента dtype (https://docs.scipy.org/doc/numpy/reference/generated/numpy.dtype.html):

In [4]:

```
np.array([1, 2, 3, 4, 5], dtype=np.float32)
```

Out[4]:

```
array([1., 2., 3., 4., 5.], dtype=float32)
```

Аналогичное преобразование:

In [5]:

```
np.float32([1, 2, 3, 4, 5])
```

Out[5]:

```
array([1., 2., 3., 4., 5.], dtype=float32)
```

Генерация Numpy arrays

- arange (https://docs.scipy.org/doc/numpy/reference/generated/numpy.arange.html) — аналог range из Python, которому можно передать нецелочисленный шаг
- linspace (https://docs.scipy.org/doc/numpy/reference/generated/numpy.linspace.html) — способ равномерно разбить отрезок на `n-1` интервал
- logspace (https://docs.scipy.org/doc/numpy/reference/generated/numpy.logspace.html) — способ разбить отрезок по логарифмической шкале
- zeros (https://docs.scipy.org/doc/numpy/reference/generated/numpy.zeros.html) — создаёт массив, заполненный нулями заданной размерности
- ones (https://docs.scipy.org/doc/numpy/reference/generated/numpy.ones.html) — создаёт массив, заполненный единицами заданной размерности
- empty (https://docs.scipy.org/doc/numpy/reference/generated/numpy.empty.html) — создаёт массив неинициализированный никаким значением заданной размерности

In [6]:

```
np.arange(0, 5, 2)
```

Out[6]:

```
array([0, 2, 4])
```

In [7]:

```
np.linspace(0, 5, 11)
```

Out[7]:

```
array([0. , 0.5, 1. , 1.5, 2. , 2.5, 3. , 3.5, 4. , 4.5, 5. ])
```

In [8]:

```
np.logspace(0, 9, 10, base=2)
```

Out[8]:

```
array([  1.,   2.,   4.,   8.,  16.,  32.,  64., 128., 256., 512.])
```

In [3]:

```
np.zeros((2, 2, 3))
```

Out[3]:

```
array([[[0., 0., 0.],
        [0., 0., 0.]],

       [[0., 0., 0.],
        [0., 0., 0.]]])
```

In [10]:

```
np.ones((2, 2))
```

Out[10]:

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

In [11]:

```
np.empty((2, 2))
```

Out[11]:

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

In [12]:

```
np.diag([1,2,3])
```

Out[12]:

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

Размеры массива хранятся в поле **shape**, а количество размерностей — в **ndim**

In [13]:

```
arr = np.ones((2, 3))
print(f"Размерность массива — {arr.shape}, количество размерностей — {arr.ndim}")
```

```
Размерность массива — (2, 3), количество размерностей — 2
```

Метод reshape (https://docs.scipy.org/doc/numpy/reference/generated/numpy.reshape.html) позволяет преобразовать размеры массива без изменения данных, но возможно с копированием

In [7]:

```
array = np.arange(0, 6)
array = array.reshape((2, 3))
array
```

Out[7]:

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

In [74]:

```
a = np.zeros((10, 2))

# A transpose makes the array non-contiguous
b = a.T

# Taking a view makes it possible to modify the shape
# without modifying the initial object.
c = b.view()

# in latest version:
# Incompatible shape for in-place modification.
# Use `.reshape()` to make a copy with the desired shape.
c.shape = (4, 5) # but a.shape is changeable
```

```
---------------------------------------------------------------------------
-
AttributeError                            Traceback (most recent call las
t)
<ipython-input-74-2bedb0f9e36f> in <module>
     11 # Incompatible shape for in-place modification.
     12 # Use `.reshape()` to make a copy with the desired shape.
---> 13 c.shape = (4, 5) # but a.shape is changeable

AttributeError: incompatible shape for a non-contiguous array
```

In [23]:

```
a
```

Out[23]:

```
array([[0., 0.],
       [0., 0.],
       [0., 0.],
       [0., 0.],
       [0., 0.],
       [0., 0.],
       [0., 0.],
       [0., 0.],
       [0., 0.],
       [0., 0.]])
```

In [24]:

```
a.shape = (20)
a
```

Out[24]:

```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0.])
```

Для того чтобы развернуть многомерный массив в вектор, можно воспользоваться функцией ravel (https://numpy.org/doc/stable/reference/generated/numpy.ravel.html) (эквивалентна `reshape(-1, order=order)`)

In [28]:

```
array = np.ravel(array)
array
```

Out[28]:

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

Индексация

В NumPy работает привычная индексация Python, включая использование отрицательных индексов и срезов

In [29]:

```
print(array)
print(array[0])
print(array[-1])
print(array[1:-1])
print(array[1:-1:2])
print(array[::-1])
```

```
[0 1 2 3 4 5]
0
5
[1 2 3 4]
[1 3]
[5 4 3 2 1 0]
```

Замечание: Индексы и срезы в многомерных массивах не нужно разделять квадратными скобками, т.е. вместо **matrix[i][j]** нужно использовать **matrix[i, j]**

Замечание: Срезы в NumPy создают view, а не копии, как в случае срезов встроенных последовательностей Python (string, tuple and list).

В качестве индексов можно использовать списки

In [39]:

```
array = np.zeros((2,3))
print(array)
array[((0,0), (0,1))]
```

```
[[0. 0. 0.]
 [0. 0. 0.]]
```

Out[39]:

```
array([0., 0.])
```

In [44]:

```
print(np.array([[True, False, True], [True, False, True]]))
array[np.array([[True, False, True], [True, False, True]])]
```

```
[[ True False  True]
 [ True False  True]]
```

Out[44]:

```
array([0., 0., 0., 0.])
```

Замечание: Индексирование с помощью массива или маски создает новый массив (копию), а не view на старые данные.

Сравнение многомерных массивов — сначала форма, потом содержимое

In [76]:

```
x = np.array([[1, 2, 3]])
y = np.array([1, 2, 3])

print(x.shape, y.shape)
print(np.array_equal(x, y))
print(np.array_equal(x, y[np.newaxis, :]))
```

```
(1, 3) (3,)
False
True
```

In [77]:

```
x = np.arange(10)
x
```

Out[77]:

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [78]:

```
x % 2
```

Out[78]:

```
array([0, 1, 0, 1, 0, 1, 0, 1, 0, 1], dtype=int32)
```

In [79]:

```
x[(x % 2 == 0) & (x > 5)]
```

Out[79]:

```
array([6, 8])
```

In [58]:

```
print(x)
x[x > 3] *= 2
print(x)
```

```
[0 1 2 3 4 5 6 7 8 9]
[ 0  1  2  3  8 10 12 14 16 18]
```

Для копирования в numpy есть метод copy

In [33]:

```
x.copy()
```

Out[33]:

```
array([ 0,  1,  2,  3,  8, 10, 12, 14, 16, 18])
```

Операции в NumPy можно производить непосредственно над векторами одинаковой размерности без использования циклов

Например, вычисление поэлементной разности между векторами выглядит следующим образом:

In [80]:

```
_l = np.ones(5)
x = np.linspace(1, 5, 5)
print(x - _l)
print(x - 2 * _l)
```

```
[0. 1. 2. 3. 4.]
[-1.  0.  1.  2.  3.]
```

Аналогчино для многомерных массивов.

Замечание: Все арифметические операции над массивами одинаковой размерности производятся поэлементно

Неочевидная проблема с типами

In [81]:

```
help(round)
```

```
Help on built-in function round in module builtins:

round(number, ndigits=None)
    Round a number to a given precision in decimal digits.

    The return value is an integer if ndigits is omitted or None.  Otherwi
se
    the return value has the same type as the number.  ndigits may be nega
tive.
```

In [82]:

```
round(1.2)
```

Out[82]:

```
1
```

In [18]:

```
np.array([1.2])[0]
```

Out[18]:

```
1.2
```

In [19]:

```
round(np.array([1.2])[0])
```

Out[19]:

```
1.0
```

> **Вопрос 1:** Что происходит, почему результаты разные?

In [20]:

```
type(1.2), type(np.array([1.2])[0])
```

Out[20]:

```
(float, numpy.float64)
```

Конкретно этот пример исправлен в numpy==1.19. Но т.к. это ломающее изменение, большая часть библиотек до сих пор зависит от numpy < 1.19

In [21]:

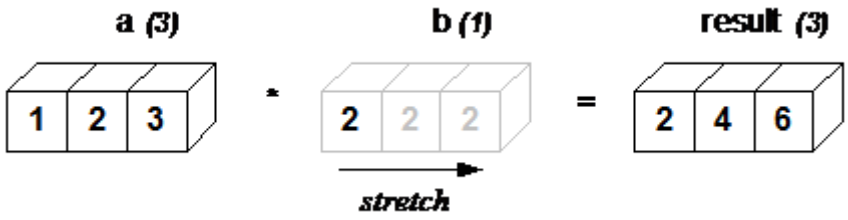
```
np.__version__
```

Out[21]:

```
'1.18.1'
```

Broadcasting (https://docs.scipy.org/doc/numpy/user/basics.broadcasting.html)

Broadcasting снимает правило одной размерности и позволяет производить арифметические операции над массивами разных, но всё-таки согласованных размерностей. Простейшим примером является умножение вектора на число



In [36]:

```
2 * np.arange(1, 4)
```

Out[36]:

```
array([2, 4, 6])
```

Правило согласования размерностей выражается в одном предложении:

`Для броадкастинга размерности по осям в двух массивах должен быть либо одинаковым, либо один из них должен быть равен единице.`

Если количество размерностей не совпадают, то к массиву меньшей размерности добавляются фиктивные размерности «слева», например

In [37]:

```
a  = np.ones((2, 3, 4))
b = np.arange(1, 5) # b.shape = (4,)
c = a * b # here a.shape = (2, 3, 4) and b.shape is considered to be (1, 1, 4)
c
```

Out[37]:

```
array([[[1., 2., 3., 4.],
        [1., 2., 3., 4.],
        [1., 2., 3., 4.]],

       [[1., 2., 3., 4.],
        [1., 2., 3., 4.],
        [1., 2., 3., 4.]]])
```

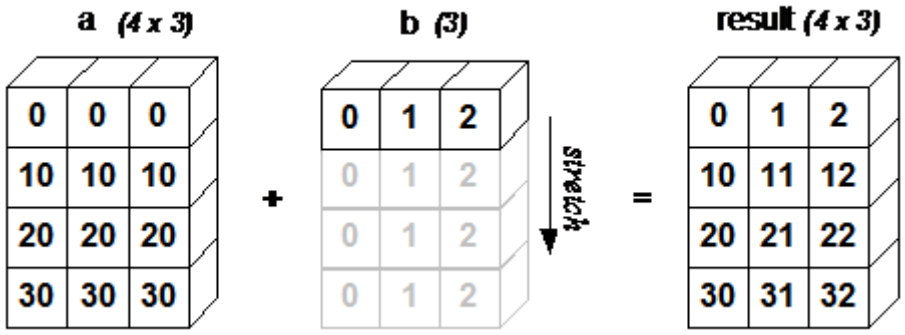
Прибавим к каждой строчке матрицы один и тот же вектор

In [64]:

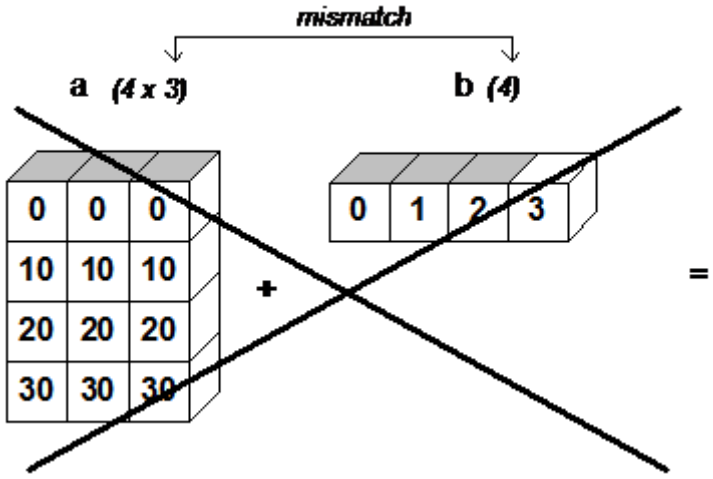
```
np.array([[0, 0, 0], [10, 10, 10], [20, 20, 20], [30, 30, 30]]) + np.arange(3)
```

Out[64]:

```
array([[ 0,  1,  2],
       [10, 11, 12],
       [20, 21, 22],
       [30, 31, 32]])
```



Теперь если мы хотим, проделать тот же трюк но со столбцами, то мы не можем просто добавить вектор состоящий из 4 элементов т.к. в данном случае размеры будут не согласованы



Сначала нужно преобразовать вектор к виду

In [65]:

```
np.arange(4)[:, np.newaxis]
```

Out[65]:

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

А затем к нему добавить матрицу:

In [66]:

```
np.arange(4)[:, np.newaxis] + np.array([[0, 0, 0], [10, 10, 10], [20, 20, 20], [30, 30, 30]])
```

Out[66]:

```
array([[ 0,  0,  0],
       [11, 11, 11],
       [22, 22, 22],
       [33, 33, 33]])
```

Если нужно перемножить многомерные массивы не поэлеметно, а по правилу перемножения матриц, то следует воспользоваться np.dot (https://docs.scipy.org/doc/numpy/reference/generated/numpy.dot.html), или оператором `@`

Транспонирование производится с помощью array**.T** или np.transpose

Так же в NumPy реализованно много полезных операций для работы с массивами: np.min (https://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.min.html), np.max (https://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.max.html), np.sum (https://docs.scipy.org/doc/numpy/reference/generated/numpy.sum.html), np.mean (https://docs.scipy.org/doc/numpy/reference/generated/numpy.mean.html) и т.д.

Замечание: В каждой из перечисленных функций есть параметр **axis**, который указывает по какому измерению производить данную операцию. По умолчанию операция производится по всем значениям массива.

Операции

In [7]:

```
x = np.arange(40).reshape(5, 2, 4)
print(x)
```

```
[[[ 0  1  2  3]
  [ 4  5  6  7]]

 [[ 8  9 10 11]
  [12 13 14 15]]

 [[16 17 18 19]
  [20 21 22 23]]

 [[24 25 26 27]
  [28 29 30 31]]

 [[32 33 34 35]
  [36 37 38 39]]]
```

In [9]:

```
print(x.T.shape)
x.T
```

```
(4, 2, 5)
```

Out[9]:

```
array([[[ 0,  8, 16, 24, 32],
        [ 4, 12, 20, 28, 36]],

       [[ 1,  9, 17, 25, 33],
        [ 5, 13, 21, 29, 37]],

       [[ 2, 10, 18, 26, 34],
        [ 6, 14, 22, 30, 38]],

       [[ 3, 11, 19, 27, 35],
        [ 7, 15, 23, 31, 39]]])
```

In [35]:

```
print(x.mean())
print(np.mean(x))
```

```
19.5
19.5
```

In [37]:

```
x_mean_0 = x.mean(axis=0)
print(x_mean_0.shape)
print(x_mean_0)
```

```
(2, 4)
[[16. 17. 18. 19.]
 [20. 21. 22. 23.]]
```

In [38]:

```
x_mean_1 = x.mean(axis=1)
print(x_mean_1.shape)
print(x_mean_1)
```

```
(5, 4)
[[ 2.  3.  4.  5.]
 [10. 11. 12. 13.]
 [18. 19. 20. 21.]
 [26. 27. 28. 29.]
 [34. 35. 36. 37.]]
```

In [39]:

```
x_mean_02 = x.mean(axis=(0, 2))
print(x_mean_02.shape)
print(x_mean_02)
```

```
(2,)
[17.5 21.5]
```

In [40]:

```
x.mean(axis=(0,1,2)), x.mean()
```

Out[40]:

```
(19.5, 19.5)
```

Конкатенация многомерных массивов

Конкатенировать несколько массивов можно с помощью функций **np.concatenate, np.vstack, np.hstack, np.dstack**

In [67]:

```
x = np.arange(10).reshape(5, 2)
y = np.arange(100, 120).reshape(5, 4)
```

In [68]:

```
x
```

Out[68]:

```
array([[0, 1],
       [2, 3],
       [4, 5],
       [6, 7],
       [8, 9]])
```

In [69]:

```
y
```

Out[69]:

```
array([[100, 101, 102, 103],
       [104, 105, 106, 107],
       [108, 109, 110, 111],
       [112, 113, 114, 115],
       [116, 117, 118, 119]])
```

In [70]:

```
np.hstack((x, y))
```

Out[70]:

```
array([[  0,   1, 100, 101, 102, 103],
       [  2,   3, 104, 105, 106, 107],
       [  4,   5, 108, 109, 110, 111],
       [  6,   7, 112, 113, 114, 115],
       [  8,   9, 116, 117, 118, 119]])
```

In [71]:

```
p = np.arange(1).reshape([1, 1, 1, 1])
p
```

Out[71]:

```
array([[[[0]]]])
```

In [42]:

```
print("vstack: ", np.vstack((p, p)).shape)
print("hstack: ", np.hstack((p, p)).shape)
print("dstack: ", np.dstack((p, p)).shape)
```

```
vstack:  (2, 1, 1, 1)
hstack:  (1, 2, 1, 1)
dstack:  (1, 1, 2, 1)
```

In [72]:

```
np.concatenate((p, p), axis=3).shape
```

Out[72]:

```
(1, 1, 1, 2)
```

Типы в NumPy

In [44]:

```
x = [1, 2, 70000]
```

In [46]:

```
np.array(x, dtype=np.float64)
```

Out[46]:

```
array([1.e+00, 2.e+00, 7.e+04])
```

In [47]:

```
np.array(x, dtype=np.uint16)
```

Out[47]:

```
array([    1,     2, 4464], dtype=uint16)
```

In [48]:

```
np.array(x, dtype=np.unicode_)
```

Out[48]:

```
array(['1', '2', '70000'], dtype='<U5')
```

Векторизация функции

In [49]:

```
def f(value):
    return np.sqrt(value)

vf = np.vectorize(f)
#  like the python map function, except it uses the broadcasting rules of numpy
```

In [50]:

```
vf(np.arange(10))
```

Out[50]:

```
array([0.        , 1.        , 1.41421356, 1.73205081, 2.        ,
       2.23606798, 2.44948974, 2.64575131, 2.82842712, 3.        ])
```

Линейная алгебра

Документацию numpy.linalg можно посмотреть тут:
https://numpy.org/doc/stable/reference/routines.linalg.html
(https://numpy.org/doc/stable/reference/routines.linalg.html)

In [23]:

```
v1 = np.array([1.0, 1.0, 2.0])
v2 = np.array([0.0, 1.0, -1.0])
np.inner(v1, v2)
```

Out[23]:

```
-1.0
```

In [26]:

```
M = np.array([[2.0, 0.0, 0.0],
              [0.0, 0.0, 1.0],
              [0.0, 1.0, 0.0]])
print(M.shape, v1.shape)
print(np.dot(M, v1))
```

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

Pandas (https://pythonspot.com/category/pandas/)

Подключаем библиотеку Pandas (от panel data), предназначенную для считывания, предобработки и быстрой визуализации структурированных данных, а также для простой аналитики.

Даже когда только два массива есть (например, группировка по одному, а вычисления по второму), уже лучше Pandas. В названиях столбцов правильно отражать физический смысл, индекс (обозначения строк) тоже не обязательно числовой.

In [83]:

```
import pandas as pd
df = pd.read_csv("titanic.csv", sep="\t")
# and also pd.read_csv, pd.read_excel, pd.read_hdf5
```

In [84]:

```
df.head()
```

Out[84]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500

In [54]:

```
df.describe()
```

Out[54]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	156.000000	156.000000	156.000000	126.000000	156.000000	156.000000	156.000000
mean	78.500000	0.346154	2.423077	28.141508	0.615385	0.397436	28.109587
std	45.177428	0.477275	0.795459	14.613880	1.056235	0.870146	39.401047
min	1.000000	0.000000	1.000000	0.830000	0.000000	0.000000	6.750000
25%	39.750000	0.000000	2.000000	19.000000	0.000000	0.000000	8.003150
50%	78.500000	0.000000	3.000000	26.000000	0.000000	0.000000	14.454200
75%	117.250000	1.000000	3.000000	35.000000	1.000000	0.000000	30.371850
max	156.000000	1.000000	3.000000	71.000000	5.000000	5.000000	263.000000

In [55]:

```
df[["Sex", "Cabin"]].describe()
```

Out[55]:

	Sex	Cabin
count	156	31
unique	2	28
top	male	C123
freq	100	2

Срезы в DataFrame

Индексация

In [57]:

```
df.sort_values("Age", inplace=True)
```

In [58]:

```
df.iloc[78] # integer-location – просто строчка по порядку от 0 до length-1
```

Out[58]:

```
PassengerId                                                    54
Survived                                                        1
Pclass                                                          2
Name           Faunthorpe, Mrs. Lizzie (Elizabeth Anne Wilkin...
Sex                                                       female
Age                                                           29
SibSp                                                          1
Parch                                                          0
Ticket                                                      2926
Fare                                                          26
Cabin                                                        NaN
Embarked                                                       S
Name: 53, dtype: object
```

In [59]:

```
df.loc[78] # index 78
```

Out[59]:

```
PassengerId                                79
Survived                                    1
Pclass                                      2
Name           Caldwell, Master. Alden Gates
Sex                                      male
Age                                      0.83
SibSp                                       0
Parch                                       2
Ticket                                 248738
Fare                                       29
Cabin                                     NaN
Embarked                                    S
Name: 78, dtype: object
```

In [61]:

```
df.loc[[78, 79, 100], ["Age", "Name"]]
```

Out[61]:

	Age	Name
78	0.83	Caldwell, Master. Alden Gates
79	30.00	Dowdell, Miss. Elizabeth
100	28.00	Petranec, Miss. Matilda

Замечание: Если хотите модифицировать данные среза, не меняя основной таблицы, нужно сделать копию.

In [81]:

```
df_slice_copy = df.loc[[78, 79, 100], ["Age", "Name"]].copy()
```

In [82]:

```
df_slice_copy["Age"] = 3
```

In [83]:

```
df_slice_copy
```

Out[83]:

	Age	Name
78	3	Caldwell, Master. Alden Gates
79	3	Dowdell, Miss. Elizabeth
100	3	Petranec, Miss. Matilda

> **Замечание:** Если хотите менять основную таблицу, то используйте **loc/iloc**

In [86]:

```
some_slice = df["Age"].isin([20, 25, 30])
df.loc[some_slice, "Fare"] = df.loc[some_slice, "Fare"] * 1000
```

Так лучше не делать:

In [87]:

```
slice_df = df[some_slice]
slice_df["Fare"] = slice_df["Fare"] * 10
```

```
C:\Users\avalur\Anaconda3\lib\site-packages\ipykernel_launcher.py:2: Setti
ngWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

Получить значения только нужных столбцов можно передав в `[]` название столбца (или список названий столбцов).

Замечание: Если передаём название одного столбца, то получаем объект класса pandas.Series (http://pandas.pydata.org/pandas-docs/stable/generated/pandas.Series.html), а если список названий столбцов, то получаем pandas.DataFrame (http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.html), чтобы получить numpy.array (https://docs.scipy.org/doc/numpy/reference/generated/numpy.array.html) обратитесь к полю **values**.

У Series и DataFrame много общих методов

In [88]:

```
# pandas.Series
df["Age"].head(5)
```

Out[88]:

```
78     0.83
119    2.00
7      2.00
16     2.00
43     3.00
Name: Age, dtype: float64
```

In [89]:

```
# pandas.DataFrame
df[["Age"]].head(5)
```

Out[89]:

	Age
78	0.83
119	2.00
7	2.00
16	2.00
43	3.00

pd.Series

Одномерные срезы датафреймов имеют тип pd.Series.

Можно получить np.array из pd.Series (но вы не хотите этого делать :).

In [90]:

```
df["Age"].head(5).values
```

Out[90]:

```
array([0.83, 2.  , 2.  , 2.  , 3.  ])
```

Можно достать и индекс

In [91]:

```
df["Age"].head(5).index
```

Out[91]:

```
Int64Index([78, 119, 7, 16, 43], dtype='int64')
```

Создаются они примерно так же, как np.array. Опционально указывается индекс

In [92]:

```
pd.Series([1, 2, 3], index=["Red", "Green", "Blue"])
```

Out[92]:

```
Red      1
Green    2
Blue     3
dtype: int64
```

In [93]:

```
pd.Series(1, index=["Red", "Green", "Blue"])
```

Out[93]:

```
Red      1
Green    1
Blue     1
dtype: int64
```

Series можно перевести в DataFrame

In [98]:

```
s = pd.Series([1, 2, 3], index=["Red", "Green", "Blue"])
s.to_frame("Values")
```

Out[98]:

	Values
Red	1
Green	2
Blue	3

In [99]:

```
s.loc["Red"]
```

Out[99]:

```
1
```

In [100]:

```
s.iloc[0]
```

Out[100]:

```
1
```

Объединение таблиц (http://pandas.pydata.org/pandas-docs/stable/merging.html)

In [101]:

```
df1 = df[["Age", "Parch"]].copy()
df2 = df[["Ticket", "Fare"]].copy()
```

In [102]:

```
# by index
df1.join(df2).head(5)
```

Out[102]:

	Age	Parch	Ticket	Fare
78	0.83	2	248738	29.0000
119	2.00	2	347082	31.2750
7	2.00	1	349909	21.0750
16	2.00	1	382652	29.1250
43	3.00	2	SC/Paris 2123	41.5792

In [103]:

```
df1 = df[["Age", "Parch", "PassengerId"]].copy()
df2 = df[["Ticket", "Fare", "PassengerId"]].copy()
```

In [105]:

```
# by columns
pd.merge(df1, df2, on=["PassengerId"]).head(5)
```

Out[105]:

	Age	Parch	PassengerId	Ticket	Fare
0	0.83	2	79	248738	29.0000
1	2.00	2	120	347082	31.2750
2	2.00	1	8	349909	21.0750
3	2.00	1	17	382652	29.1250
4	3.00	2	44	SC/Paris 2123	41.5792

In [106]:

```
pd.merge(df1, df2, on=["PassengerId"], how="inner").head(5)
```

Out[106]:

	Age	Parch	PassengerId	Ticket	Fare
0	0.83	2	79	248738	29.0000
1	2.00	2	120	347082	31.2750
2	2.00	1	8	349909	21.0750
3	2.00	1	17	382652	29.1250
4	3.00	2	44	SC/Paris 2123	41.5792

Группировка

In [107]:

```
print("Pclass 1: ", df[df["Pclass"] == 1]["Age"].mean())
print("Pclass 2: ", df[df["Pclass"] == 2]["Age"].mean())
print("Pclass 3: ", df[df["Pclass"] == 3]["Age"].mean())
```

```
Pclass 1:  38.111111111111114
Pclass 2:  28.114827586206893
Pclass 3:  24.307142857142857
```

In [110]:

```
df.groupby(["Pclass"])[["Age"]].mean()
```

Out[110]:

	Age
Pclass	
1	38.111111
2	28.114828
3	24.307143

In [111]:

```
df.groupby(["Survived", "Pclass"])
```

Out[111]:

```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x000001DBCEB6B7C8
>
```

In [112]:

```
df.groupby(["Survived", "Pclass"])["PassengerId"].count()
```

Out[112]:

```
Survived  Pclass
0         1         18
          2         16
          3         68
1         1         12
          2         14
          3         28
Name: PassengerId, dtype: int64
```

In [113]:

```
df.groupby(["Survived", "Pclass"])[["PassengerId", "Cabin"]].count()
```

Out[113]:

		PassengerId	Cabin
Survived	**Pclass**		
0	**1**	18	12
	2	16	1
	3	68	1
1	**1**	12	12
	2	14	3
	3	28	2

In [114]:

```
df.groupby(["Survived", "Pclass"])[["PassengerId"]].describe()
```

Out[114]:

		PassengerId							
		count	**mean**	**std**	**min**	**25%**	**50%**	**75%**	**max**
Survived	**Pclass**								
0	**1**	18.0	82.555556	44.501450	7.0	40.75	88.5	117.00	156.0
	2	16.0	107.187500	44.842270	21.0	72.50	122.0	145.25	151.0
	3	68.0	80.161765	43.375352	1.0	46.75	84.0	114.25	155.0
1	**1**	12.0	60.083333	50.118874	2.0	21.00	54.5	91.25	152.0
	2	14.0	62.000000	39.240874	10.0	27.50	58.0	83.50	134.0
	3	28.0	71.607143	45.525198	3.0	32.00	72.0	108.50	147.0

Работа с timestamp'ами

In [115]:

```
tdf = df.copy()
tdf["ts"] = range(1560000000, 1560000000 + tdf.shape[0])
```

In [116]:

```
tdf.head(2)
```

Out[116]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare
78	79	1	2	Caldwell, Master. Alden Gates	male	0.83	0	2	248738	29.000
119	120	0	3	Andersson, Miss. Ellis Anna Maria	female	2.00	4	2	347082	31.275

In [117]:

```
tdf["ts"] = pd.to_datetime(tdf["ts"], unit="s")
```

In [118]:

```
tdf.head(2)
```

Out[118]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare
78	79	1	2	Caldwell, Master. Alden Gates	male	0.83	0	2	248738	29.000
119	120	0	3	Andersson, Miss. Ellis Anna Maria	female	2.00	4	2	347082	31.275

In [119]:

```
tdf.set_index("ts", inplace=True)
```

In [120]:

```
# для сглаживания
tdf.resample("15s").sum()
```

Out[120]:

ts	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
2019-06-08 13:20:00	837	6	41	97.83	33	19	415.6084
2019-06-08 13:20:15	1117	5	41	253.50	18	9	483.4376
2019-06-08 13:20:30	1445	3	37	305.00	4	4	328127.9958
2019-06-08 13:20:45	1569	6	34	346.00	6	4	77305.0458
2019-06-08 13:21:00	1116	5	36	407.50	7	4	130305.3374
2019-06-08 13:21:15	1324	8	38	468.00	7	1	125008.5166
2019-06-08 13:21:30	829	5	31	554.50	9	12	472.2124
2019-06-08 13:21:45	1531	2	26	724.00	5	4	601.4584
2019-06-08 13:22:00	642	7	35	389.50	1	1	361.8667
2019-06-08 13:22:15	1052	5	41	0.00	4	1	194.8457
2019-06-08 13:22:30	784	2	18	0.00	2	3	84.8666

In [121]:

```
tdf.resample("1T").sum()
```

Out[121]:

ts	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
2019-06-08 13:20:00	4968	20	153	1002.33	61	36	406332.0876
2019-06-08 13:21:00	4800	20	131	2154.00	28	21	256387.5248
2019-06-08 13:22:00	2478	14	94	389.50	7	5	641.5790

Rolling функции

In [122]:

```
tdf.sort_index(inplace=True)
```

In [129]:

```
tdf[["Fare"]].rolling(window=5).mean().head(10)
```

Out[129]:

	Fare
ts	
2019-06-08 13:20:00	NaN
2019-06-08 13:20:01	NaN
2019-06-08 13:20:02	NaN
2019-06-08 13:20:03	NaN
2019-06-08 13:20:04	30.41084
2019-06-08 13:20:05	30.19084
2019-06-08 13:20:06	27.27584
2019-06-08 13:20:07	28.61084
2019-06-08 13:20:08	30.72334
2019-06-08 13:20:09	26.62250

Удобно делать вместе с groupby

In [130]:

```
rol = tdf[["Sex", "Fare"]].groupby(["Sex"]).rolling(window=5).mean()
```

In [132]:

```
rol.head(6)
```

Out[132]:

		Fare
Sex	**ts**	
female	**2019-06-08 13:20:01**	NaN
	2019-06-08 13:20:04	NaN
	2019-06-08 13:20:06	NaN
	2019-06-08 13:20:07	NaN
	2019-06-08 13:20:09	27.67584
	2019-06-08 13:20:10	28.29584

In [149]:

```
rol.loc[("male")].head(10)
```

Out[149]:

	Fare
ts	
2019-06-08 13:20:00	NaN
2019-06-08 13:20:02	NaN
2019-06-08 13:20:03	NaN
2019-06-08 13:20:05	NaN
2019-06-08 13:20:08	29.35750
2019-06-08 13:20:11	32.93750
2019-06-08 13:20:12	30.97084
2019-06-08 13:20:18	26.98918
2019-06-08 13:20:19	28.28418
2019-06-08 13:20:25	22.64668

Работа со строками

In [150]:

```
df["Name"].str.lower()\
          .str.replace(",", " ")\
          .str.split(".").str[1]\ # expand=True
          .head(10)
```

Out[150]:

```
78                    alden gates
119              ellis anna maria
7                   gosta leonard
16                          eugene
43      simonne marie anne andree
63                          harald
10                 marguerite rut
58               constance mirium
50                     juha niilo
24                 torborg danira
Name: Name, dtype: object
```

Работа с NaN'ами

In [151]:

```
df["Cabin"].head(15)
```

Out[151]:

```
78     NaN
119    NaN
7      NaN
16     NaN
43     NaN
63     NaN
10      G6
58     NaN
50     NaN
24     NaN
147    NaN
59     NaN
125    NaN
9      NaN
14     NaN
Name: Cabin, dtype: object
```

In [152]:

```
df["Cabin"].dropna().head(15)
```

Out[152]:

```
10              G6
27     C23 C25 C27
136            D47
102            D26
151             C2
88     C23 C25 C27
97         D10 D12
118        B58 B60
139            B86
75           F G73
23              A6
66             F33
123           E101
21             D56
3             C123
Name: Cabin, dtype: object
```

In [153]:

```
df["Cabin"].fillna(3).head(5)
```

Out[153]:

```
78     3
119    3
7      3
16     3
43     3
Name: Cabin, dtype: object
```

In [154]:

```
df["Cabin"].fillna(method="backfill").head(15)
```

Out[154]:

```
78              G6
119             G6
7               G6
16              G6
43              G6
63              G6
10              G6
58     C23 C25 C27
50     C23 C25 C27
24     C23 C25 C27
147    C23 C25 C27
59     C23 C25 C27
125    C23 C25 C27
9      C23 C25 C27
14     C23 C25 C27
Name: Cabin, dtype: object
```

In [155]:

```
pd.isna(df["Cabin"]).head(10)
```

Out[155]:

```
78      True
119     True
7       True
16      True
43      True
63      True
10     False
58      True
50      True
24      True
Name: Cabin, dtype: bool
```

Быстрая визуализация

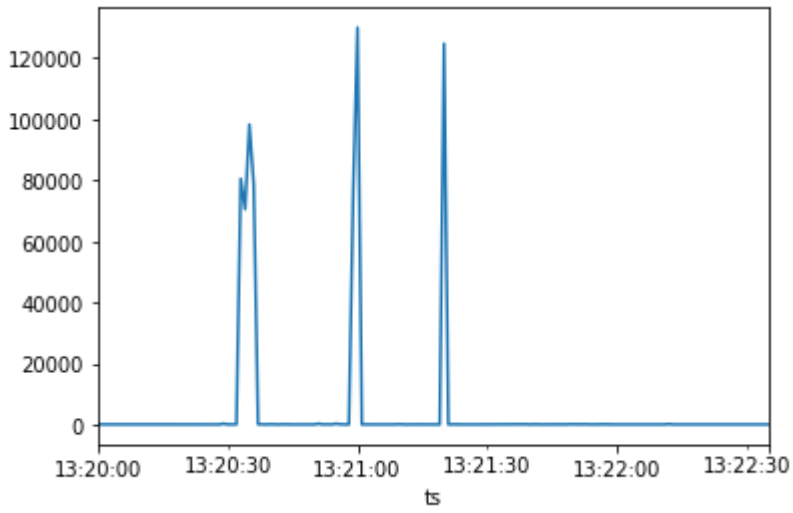
In [156]:

```
%matplotlib inline

tdf["Fare"].plot()
```

Out[156]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1dbcf1b2e48>
```

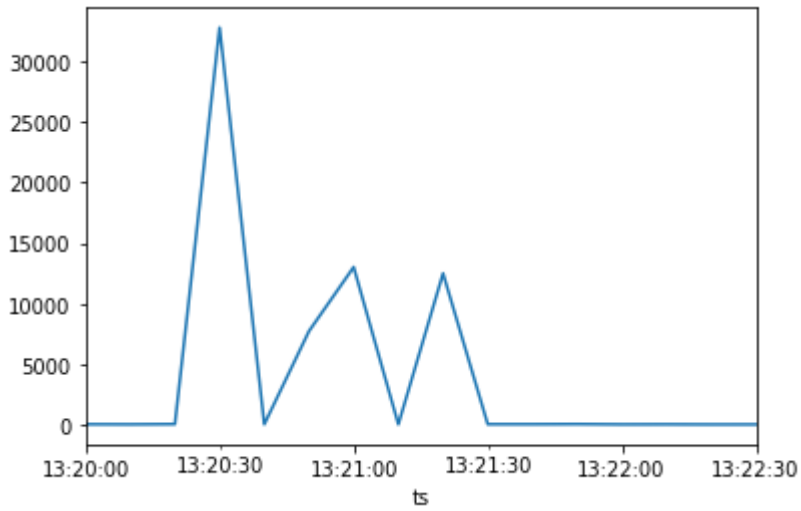


In [157]:

```
tdf["Fare"].resample("10s").mean().plot()
```

Out[157]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1dbd02bb248>
```



In [158]:

```
tdf["Sex"].hist()
```

Out[158]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1dbd030c1c8>
```