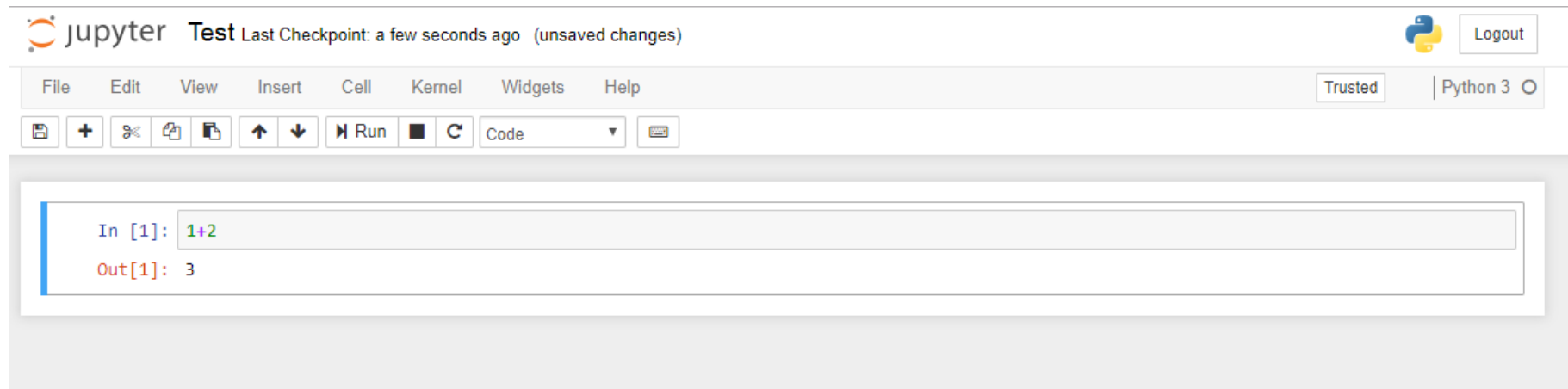


Возможности Jupyter

Jupyter Notebook

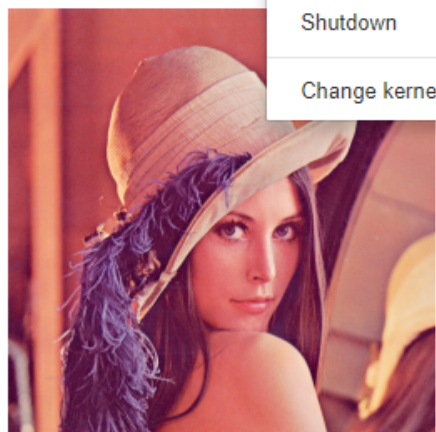


Типы ячеек:

- Code
- Markdown
- Raw NBConvert (сырой код)

Тип ячейки можно задать в командном режиме либо с помощью горячих клавиш (**y** to code, **m** to markdown, **r** to edit raw text), либо в меню *Cell* -> *Cell type*.

```
In [3]: from IPython.core.display
display(Image('lena.png'))
```

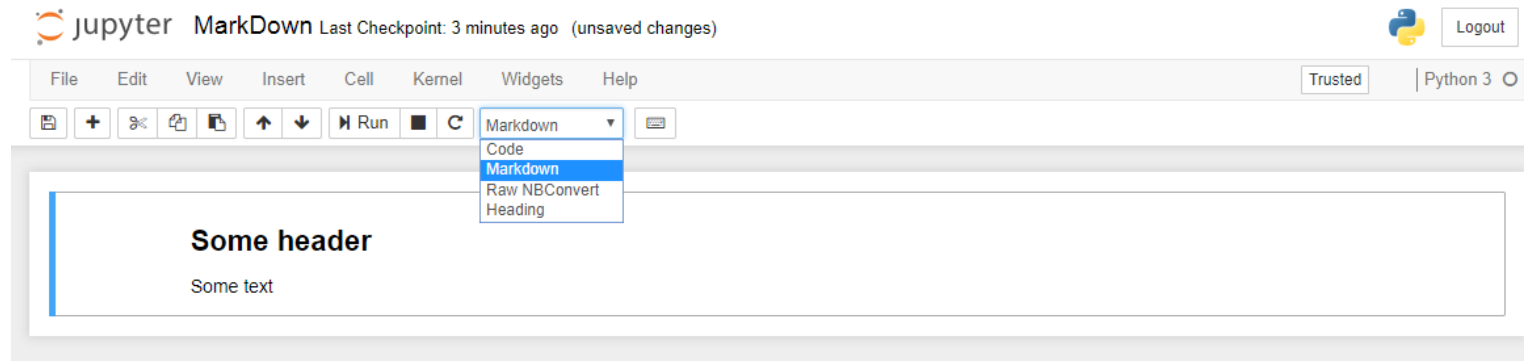


```
In [7]: import pandas as pd
df = pd.read_csv('titanic.csv')
df.head()
```

Out[7]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

Markdown



```
1      # Markdown
2
3      ## Краткое руководство
4
5      Абзацы создаются при помощи пустой строки. Если вокруг текста сверху и снизу
6      есть пустые строки, то текст превращается в абзац.
7
8      Чтобы сделать перенос строки вместо абзаца,
9      нужно поставить два пробела в конце предыдущей строки.
10
11     Заголовки отмечаются диэзом `#` в начале строки, от одного до шести. Например:
12
13     # Заголовок первого уровня #
14     ## Заголовок h2
15     ### Заголовок h3
16     #### Заголовок h4
17     ##### Заголовок h5
18     ##### Заголовок h6
19
20     В декоративных целях заголовки можно «закрывать» с обратной стороны.
```

Markdown

Краткое руководство

Абзацы создаются при помощи пустой строки. Если вокруг текста сверху и снизу есть пустые строки, то текст превращается в абзац.

Чтобы сделать перенос строки вместо абзаца, нужно поставить два пробела в конце предыдущей строки.

Заголовки отмечаются диэзом `#` в начале строки, от одного до шести. Например:

Заголовок первого уровня

Заголовок h2

Заголовок h3

Заголовок h4

Заголовок h5

ЗАГОЛОВОК h6

В декоративных целях заголовки можно «закрывать» с обратной стороны.

```

21     ### Списки
22
23     Для разметки неупорядоченных списков можно использовать или `*`, или `-`, или
24     `+`:
25
26     - элемент 1
27     - элемент 2
28     - элемент ...
29
30     Вложенные пункты создаются четырьмя пробелами перед маркером пункта:
31
32     * элемент 1
33     * элемент 2
34         * вложенный элемент 2.1
35         * вложенный элемент 2.2
36     * элемент ...
37
38     Упорядоченный список:
39
40     1. элемент 1
41     2. элемент 2
42         1. вложенный
43         2. вложенный
44     3. элемент 3
45     4. Donec sit amet nisl. Aliquam semper ipsum sit amet velit. Suspendisse id sem
46         consectetur libero luctus adipiscing.

```

Списки

Для разметки неупорядоченных списков можно использовать или `*`, или `-`, или `+`:

- элемент 1
- элемент 2
- элемент ...

Вложенные пункты создаются четырьмя пробелами перед маркером пункта:

- элемент 1
- элемент 2
 - вложенный элемент 2.1
 - вложенный элемент 2.2
- элемент ...

Упорядоченный список:

1. элемент 1
2. элемент 2
 1. вложенный
 2. вложенный
3. элемент 3
4. Donec sit amet nisl. Aliquam semper ipsum sit amet velit. Suspendisse id sem consectetur libero luctus adipiscing.

46 На самом деле не важно как в коде пронумерованы пункты, главное, чтобы перед
элементом списка стояла цифра (любая) с точкой. Можно сделать и так:

47

48 0. элемент 1

49 0. элемент 2

50 0. элемент 3

51 0. элемент 4

52

53 Список с абзацами:

54

55 * Раз абзац. Lorem ipsum dolor sit amet, consectetur adipisicing elit.

56

57 * Два абзац. Donec sit amet nisl. Aliquam semper ipsum sit amet velit.
Suspendisse id sem consectetur libero luctus adipiscing.

58

59 * Три абзац. Ea, quis, alias nobis porro quos laborum minus sed fuga odio
dolore natus quas cum enim necessitatibus magni provident non saepe sequi?

60

61 Четыре абзац (Четыре пробела в начале или один tab).

62

На самом деле не важно как в коде пронумерованы пункты, главное, чтобы перед элементом списка стояла цифра (любая) с точкой. Можно сделать и так:

1. элемент 1
2. элемент 2
3. элемент 3
4. элемент 4

Список с абзацами:

- Раз абзац. Lorem ipsum dolor sit amet, consectetur adipisicing elit.
- Два абзац. Donec sit amet nisl. Aliquam semper ipsum sit amet velit. Suspendisse id sem consectetur libero luctus adipiscing.
- Три абзац. Ea, quis, alias nobis porro quos laborum minus sed fuga odio dolore natus quas cum enim necessitatibus magni provident non saepe sequi?

Четыре абзац (Четыре пробела в начале или один tab).

```

63     ### Цитаты
64
65     Цитаты оформляются как в емейлах, с помощью символа `>`.
66
67     > This is a blockquote with two paragraphs. Lorem ipsum dolor sit amet,
68     > consectetur adipiscing elit. Aliquam hendrerit mi posuere lectus.
69     > Vestibulum enim wisi, viverra nec, fringilla in, laoreet vitae, risus.
70     >
71     > Donec sit amet nisl. Aliquam semper ipsum sit amet velit. Suspendisse
72     > id sem consectetur libero luctus adipiscing.
73
74     Или более ленивым способом, когда знак `>` ставится перед каждым элементом
75     цитаты, будь то абзац, заголовок или пустая строка:
76
77     > This is a blockquote with two paragraphs. Lorem ipsum dolor sit amet,
78     > consectetur adipiscing elit. Aliquam hendrerit mi posuere lectus.
79     > Vestibulum enim wisi, viverra nec, fringilla in, laoreet vitae, risus.
80     >
81     > Donec sit amet nisl. Aliquam semper ipsum sit amet velit. Suspendisse
82     > id sem consectetur libero luctus adipiscing.
83
84     В цитаты можно помещать всё что угодно, в том числе вложенные цитаты:
85
86     > ## This is a header.
87     >
88     > 1. This is the first list item.
89     > 2. This is the second list item.
90     >
91     > > Вложенная цитата.
92     >
93     > Here's some example code:
94     >
95     >     return shell_exec("echo $input | $markdown_script");

```

Цитаты

Цитаты оформляются как в емейлах, с помощью символа `>`.

This is a blockquote with two paragraphs. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam hendrerit mi posuere lectus. Vestibulum enim wisi, viverra nec, fringilla in, laoreet vitae, risus.

Donec sit amet nisl. Aliquam semper ipsum sit amet velit. Suspendisse id sem consectetur libero luctus adipiscing.

Или более ленивым способом, когда знак `>` ставится перед каждым элементом цитаты, будь то абзац, заголовок или пустая строка:

This is a blockquote with two paragraphs. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam hendrerit mi posuere lectus. Vestibulum enim wisi, viverra nec, fringilla in, laoreet vitae, risus.

Donec sit amet nisl. Aliquam semper ipsum sit amet velit. Suspendisse id sem consectetur libero luctus adipiscing.

В цитаты можно помещать всё что угодно, в том числе вложенные цитаты:

This is a header.

1. This is the first list item.
2. This is the second list item.

Вложенная цитата.

Here's some example code:

```
return shell_exec("echo $input | $markdown_script");
```

```
96     ### Исходный код
97
98     В Маркдауне блоки кода отбиваются 4 пробелами в начале каждой строки.
99
100     <nav class="nav nav-primary">
101     <ul>
102         <li class="tab-conversation active">
103             <a href="#" data-role="post-count" class="publisher-nav-color" data-
104             nav="conversation">
105                 <span class="comment-count">0 комментариев</span>
106                 <span class="comment-count-placeholder">Комментарии</span>
107             </a>
108         </li>
109         <li class="dropdown user-menu" data-role="logout">
110             <a href="#" class="dropdown-toggle" data-toggle="dropdown">
111                 <span class="dropdown-toggle-wrapper">
112                     <span>
113                         Войти
114                     </span>
115                     <span class="caret"></span>
116                 </a>
117             </li>
118         </ul>
119     </nav>
120
```

Исходный код

В Маркдауне блоки кода отбиваются 4 пробелами в начале каждой строки.

```
<nav class="nav nav-primary">
<ul>
  <li class="tab-conversation active">
    <a href="#" data-role="post-count" class="publisher-nav-color" data-nav
    ="conversation">
      <span class="comment-count">0 комментариев</span>
      <span class="comment-count-placeholder">Комментарии</span>
    </a>
  </li>
  <li class="dropdown user-menu" data-role="logout">
    <a href="#" class="dropdown-toggle" data-toggle="dropdown">
      <span class="dropdown-toggle-wrapper">
        <span>
          Войти
        </span>
      </span>
      <span class="caret"></span>
    </a>
  </li>
</ul>
</nav>
```

```

130     ### Горизонтальная черта
131
132     `hr` создается тремя звездочками или тремя дефисами.
133
134     ***
135
136     ### Ссылки
137
138     Это встроенная [ссылка с title элементом](http://example.com/link "Я ссылка").
    Это – [без title](http://example.com/link).
139
140     А вот [пример][1] [нескольких][2] [ссылок][id] с разметкой как у сносок.
    Прокатит и [короткая запись][] без указания id.
141
142     [1]: http://example.com/ "Optional Title Here"
143     [2]: http://example.com/some
144     [id]: http://example.com/links (Optional Title Here)
145     [короткая запись]: http://example.com/short
146
147     Вынос длинных урлов из предложения способствует сохранению читабельности
    источника. Сноски можно располагать в любом месте документа.
148
149     ### Emphasis
150
151     Выделять слова можно при помощи `*` и `_`. Одним символ для наклонного текста,
    два символа для жирного текста, три – для наклонного и жирного одновременно.
152
153     Например, это _italic_ и это тоже *italic*. А вот так уже __strong__, и так
    тоже **strong**. А так ***жирный и наклонный*** одновременно.
154
155     ### Зачеркивание
156
157     В GFM добавлено зачеркивание текста: две тильды ~` до и после текста.
158
159     Зачеркнуто

```

Горизонтальная черта

`hr` создается тремя звездочками или тремя дефисами.

* * *

Ссылки

Это встроенная [ссылка с title элементом](#). Это — [без title](#).

А вот [пример нескольких ссылок](#) с разметкой как у сносок. Прокатит и [короткая запись](#) без указания `id`.

Вынос длинных урлов из предложения способствует сохранению читабельности исходника. Сноски можно располагать в любом месте документа.

Emphasis

Выделять слова можно при помощи `*` и `_`. Одним символ для наклонного текста, два символа для жирного текста, три — для наклонного и жирного одновременно.

Например, это *italic* и это тоже *italic*. А вот так уже **strong**, и так тоже **strong**. А так ***жирный и наклонный*** одновременно.

Зачеркивание

В GFM добавлено зачеркивание текста: две тильды `~` до и после текста.

~~Зачеркнуто~~


```
161     ## Картинки
162
163     Картинка без `alt` текста
164
165     
166
167     Картинка с альтом и тайтлом:
168
169     ![Alt text](//placeholder.it/150x100 "Можно задать title")
170
171     Запомнить просто: синтаксис как у ссылок, только перед открывающей квадратной
172     скобкой ставится восклицательный знак.
173
174     Картинки «сноски»:
175
176     ![Картинка][image1]
177     ![Картинка][image2]
178     ![Картинка][image3]
179
180     [image1]: //placeholder.it/250x100
181     [image2]: //placeholder.it/200x100
182     [image3]: //placeholder.it/150x100
183
184     Картинки-ссылки:
185
186     [![Alt text](//placeholder.it/150x100)](http://example.com/)
187
```

Картинки

Картинка без `alt` текста

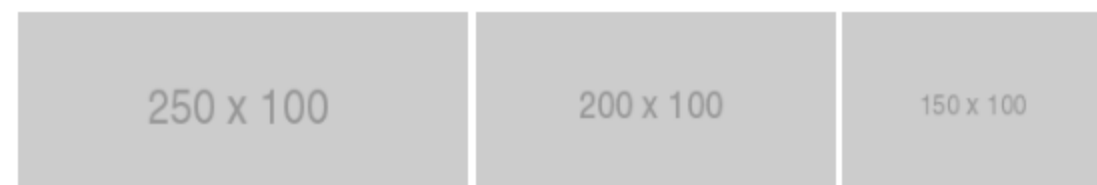


Картинка с альтом и тайтлом:

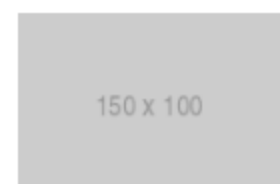


Запомнить просто: синтаксис как у ссылок, только перед открывающей квадратной скобкой ставится восклицательный знак.

Картинки «сноски»:



Картинки-ссылки:



H3 title for the text

TODO List:

1. Create note about IPython notebook.
2. Add part about Markdown.
3. Do not forget about MathJax.
4. Add table in the end.

<h4>An Identity of Ramanujan</h4>

$$\frac{1}{\sqrt{\phi\sqrt{5}-\phi}}e^{\frac{1}{3}\pi} = 1 + \frac{e^{-2\pi}}{1 + \frac{e^{-4\pi}}{1 + \frac{e^{-6\pi}}{1 + \dots}}}$$

Simple table

Tables	True	False
-----	:-----:	-----:
True	78	14
False	3	45

H3 title for the text

TODO List:

1. Create note about IPython notebook.
2. Add part about Markdown.
3. Don't forget about MathJax.
4. Add table in the end.

An Identity of Ramanujan

$$\frac{1}{(\sqrt{\phi\sqrt{5}-\phi})e^{\frac{1}{3}\pi}} = 1 + \frac{e^{-2\pi}}{1 + \frac{e^{-4\pi}}{1 + \frac{e^{-6\pi}}{1 + \dots}}}$$

Simple table

Tables	True	False
True	78	14
False	3	45

iagsav / sometest

Unwatch 1 Star 0 Fork 0

Code Issues 0 Pull requests 0 Projects 0 Wiki Settings Insights

Branch: master sometest / sometest / TestNotebook.ipynb

Find file Copy path

Your Name x ed1e469 2 minutes ago

1 contributor

206 lines (205 sloc) 639 KB

Raw Blame History

```
In [3]: from IPython.core.display import Image, display
display(Image('lena.png', width=256, unconfined=True))
```



```
In [7]: import pandas as pd
df = pd.read_csv('titanic.csv')
df.head()
```

Out[7]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

NumPy

NumPy — библиотека языка Python, позволяющая [удобно] работать с многомерными массивами и матрицами, содержащая математические функции. Кроме того, NumPy позволяет векторизовать многие вычисления, имеющие место в машинном обучении.

```
In [4]: import numpy as np
```

Основным типом данных NumPy является многомерный массив элементов одного типа — [numpy.ndarray](#). Каждый подобный массив имеет несколько *измерений* или *осей* — в частности, вектор (в классическом понимании) является одномерным массивом и имеет 1 ось, матрица является двумерным массивом и имеет 2 оси и т.д.

```
In [5]: vec = np.array([1, 2, 3])
vec.ndim # количество осей
```

```
Out[5]: 1
```

```
In [6]: mat = np.array([[1, 2, 3], [4, 5, 6]])
mat.ndim
```

```
Out[6]: 2
```

Чтобы узнать длину массива по каждой из осей, можно воспользоваться атрибутом `shape`:

```
In [8]: vec.shape
```

```
Out[8]: (3,)
```

Чтобы узнать тип элементов и их размер в байтах:

```
In [9]: mat.dtype.name
```

```
Out[9]: 'int64'
```

```
In [10]: mat.itemsize
```

```
Out[10]: 8
```

Создание массивов

- Передать итерируемый объект в качестве параметра функции `array` (можно также явно указать тип элементов):

```
In [11]: A = np.array([1, 2, 3])  
A
```

```
Out[11]: array([1, 2, 3])
```

```
In [12]: A = np.array([1, 2, 3], dtype = float)  
A
```

```
Out[12]: array([ 1.,  2.,  3.])
```

```
In [13]: B = np.array([(1, 2, 3), (4, 5, 6)])  
B
```

```
Out[13]: array([[1, 2, 3],  
               [4, 5, 6]])
```

- Создание массивов специального вида при помощи функций `zeros`, `ones`, `empty`, `identity`:

```
In [15]: np.zeros((3,))
```

```
Out[15]: array([ 0.,  0.,  0.])
```

```
In [14]: np.ones((3, 4))
```

```
Out[14]: array([[ 1.,  1.,  1.,  1.],  
               [ 1.,  1.,  1.,  1.],  
               [ 1.,  1.,  1.,  1.]])
```

```
In [15]: np.identity(3)
```

```
Out[15]: array([[ 1.,  0.,  0.],  
               [ 0.,  1.,  0.],  
               [ 0.,  0.,  1.]])
```

```
In [16]: np.empty((2, 5))
```

```
Out[16]: array([[ 1.72723371e-077,  2.68679363e+154,  2.22586790e-314,
                  2.22594686e-314,  2.22594008e-314],
                [ 2.22598140e-314,  2.22236417e-314,  0.00000000e+000,
                  2.22024022e-314,  6.95337803e-309]])
```

Обратите внимание, что содержимое массива, созданного при помощи функции `empty`, **не инициализируется**, то есть в качестве значений он **может содержать "мусор"**.

- Создание последовательностей при помощи функций `arange` (в качестве параметров принимает левую и правую границы последовательности и `step`) и `linspace` (принимает левую и правую границы и **количество элементов**):

```
In [17]: np.arange(2, 20, 3) # аналогично стандартной функции range python, правая граница не включается
```

```
Out[17]: array([ 2,  5,  8, 11, 14, 17])
```

```
In [18]: np.arange(2.5, 8.7, 0.9) # но может работать и с вещественными числами
```

```
Out[18]: array([ 2.5,  3.4,  4.3,  5.2,  6.1,  7. ,  7.9])
```

```
In [19]: np.linspace(2, 18, 14) # правая граница включается (по умолчанию)
```

```
Out[19]: array([ 2.          ,  3.23076923,  4.46153846,  5.69230769,
                  6.92307692,  8.15384615,  9.38461538, 10.61538462,
                  11.84615385, 13.07692308, 14.30769231, 15.53846154,
                  16.76923077, 18.          ])
```

- Для изменения размеров существующего массива можно воспользоваться функцией `reshape` (при этом количество элементов должно оставаться неизменным):

```
In [20]: np.arange(9).reshape(3, 3)
```

```
Out[20]: array([[0, 1, 2],
                 [3, 4, 5],
                 [6, 7, 8]])
```


Вместо значения длины массива по одному из измерений можно указать -1 — в этом случае значение будет рассчитано автоматически:

```
In [18]: np.arange(8).reshape(2, -1)
```

```
Out[18]: array([[0, 1, 2, 3],  
               [4, 5, 6, 7]])
```

- Транспонирование существующего массива:

```
In [19]: C = np.arange(6).reshape(2, -1)  
C
```

```
Out[19]: array([[0, 1, 2],  
               [3, 4, 5]])
```

```
In [20]: C.T
```

```
Out[20]: array([[0, 3],  
               [1, 4],  
               [2, 5]])
```

- Объединение существующих массивов по заданной оси:

```
In [21]: A = np.arange(6).reshape(2, -1)  
np.hstack((A, A**2))
```

```
Out[21]: array([[ 0,  1,  2,  0,  1,  4],  
               [ 3,  4,  5,  9, 16, 25]])
```

```
In [25]: np.vstack((A, A**2))
```

```
Out[25]: array([[ 0,  1,  2],  
               [ 3,  4,  5],  
               [ 0,  1,  4],  
               [ 9, 16, 25]])
```

```
In [26]: np.concatenate((A, A**2), axis = 1)
```

```
Out[26]: array([[ 0,  1,  2,  0,  1,  4],  
               [ 3,  4,  5,  9, 16, 25]])
```

- Повторение существующего массива

```
In [22]: a = np.arange(3)  
np.tile(a, (2, 2))
```

```
Out[22]: array([[0, 1, 2, 0, 1, 2],  
               [0, 1, 2, 0, 1, 2]])
```

```
In [23]: np.tile(a, (4, 1))
```

```
Out[23]: array([[0, 1, 2],  
               [0, 1, 2],  
               [0, 1, 2],  
               [0, 1, 2]])
```

Базовые операции

- Базовые арифметические операции над массивами выполняются поэлементно:

```
In [24]: A = np.arange(9).reshape(3, 3)
        B = np.arange(1, 10).reshape(3, 3)
```

```
In [29]: print A
        print B
```

```
[[0 1 2]
 [3 4 5]
 [6 7 8]]
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

```
In [30]: A + B
```

```
Out[30]: array([[ 1,  3,  5],
                [ 7,  9, 11],
                [13, 15, 17]])
```

```
In [32]: A * 1.0 / B
```

```
Out[32]: array([[ 0.         ,  0.5         ,  0.66666667],
                [ 0.75        ,  0.8         ,  0.83333333],
                [ 0.85714286,  0.875        ,  0.88888889]])
```

```
In [33]: A + 1
```

```
Out[33]: array([[1, 2, 3],
                [4, 5, 6],
                [7, 8, 9]])
```

```
In [34]: 3 * A
```

```
Out[34]: array([[ 0,  3,  6],
                [ 9, 12, 15],
                [18, 21, 24]])
```

```
In [35]: A ** 2
```

```
Out[35]: array([[ 0,  1,  4],
                [ 9, 16, 25],
                [36, 49, 64]])
```

Отдельно обратим внимание на то, что умножение массивов также является **поэлементным**, а не матричным:


```
In [36]: A * B
```

```
Out[36]: array([[ 0,  2,  6],
               [12, 20, 30],
               [42, 56, 72]])
```

Для выполнения матричного умножения необходимо использовать функцию `dot`:

```
In [40]: A.dot(B)
```

```
Out[40]: array([[ 18,  21,  24],
               [ 54,  66,  78],
               [ 90, 111, 132]])
```

Поскольку операции выполняются поэлементно, операнды бинарных операций должны иметь одинаковый размер. Тем не менее, операция может быть корректно выполнена, если размеры операндов таковы, что они могут быть расширены до одинаковых размеров. Данная возможность называется **broadcasting**:  http://www.scipy-lectures.org/_images/numpy_broadcasting.png

```
In [41]: np.tile(np.arange(0, 40, 10), (3, 1)).T + np.array([0, 1, 2])
```

```
Out[41]: array([[ 0,  1,  2],
               [10, 11, 12],
               [20, 21, 22],
               [30, 31, 32]])
```

- Универсальные функции (`sin`, `cos`, `exp` и т.д.) также применяются поэлементно:

```
In [42]: np.exp(A)
```

```
Out[42]: array([[ 1.00000000e+00,  2.71828183e+00,  7.38905610e+00],
               [ 2.00855369e+01,  5.45981500e+01,  1.48413159e+02],
               [ 4.03428793e+02,  1.09663316e+03,  2.98095799e+03]])
```

- Некоторые операции над массивами (например, вычисления минимума, максимума, суммы элементов) выполняются над всеми элементами вне зависимости от формы массива, однако при указании оси выполняются вдоль нее (например, для нахождения максимума каждой строки или каждого столбца):

In [43]:

```
A
```

Out[43]: array([[0, 1, 2],
 [3, 4, 5],
 [6, 7, 8]])

In [44]:

```
A.min()
```

Out[44]: 0

In [42]:

```
A.max(axis = 0)
```

Out[42]: array([6, 7, 8])

In [45]:

```
A.sum(axis = 1)
```

Out[45]: array([3, 12, 21])

Индексация

Для доступа к элементам может использоваться [много различных способов](#), рассмотрим основные.

- Для индексации могут использоваться конкретные значения индексов и срезы (slice), как и в стандартных типах Python. Для многомерных массивов индексы для различных осей разделяются запятой. Если для многомерного массива указаны индексы не для всех измерений, недостающие заполняются полным срезом (:).

```
In [46]: a = np.arange(10)
a
```

```
Out[46]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
In [47]: a[2:5]
```

```
Out[47]: array([2, 3, 4])
```

```
In [48]: a[3:8:2]
```

```
Out[48]: array([3, 5, 7])
```

```
In [49]: A = np.arange(81).reshape(9, -1)
A
```

```
Out[49]: array([[ 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, 40, 41, 42, 43, 44],
 [45, 46, 47, 48, 49, 50, 51, 52, 53],
 [54, 55, 56, 57, 58, 59, 60, 61, 62],
 [63, 64, 65, 66, 67, 68, 69, 70, 71],
 [72, 73, 74, 75, 76, 77, 78, 79, 80]])
```

```
In [50]: A[2:4]
```

```
Out[50]: array([[18, 19, 20, 21, 22, 23, 24, 25, 26],
 [27, 28, 29, 30, 31, 32, 33, 34, 35]])
```

Индексация начинается с 0

item[START:STOP:STEP] – берёт срез от номера START, до STOP (не включая его), с шагом STEP. По умолчанию START = 0, STOP = длине объекта, STEP = 1.

```
In [51]: A[:, 2:4]
```

```
Out[51]: array([[ 2,  3],
                [11, 12],
                [20, 21],
                [29, 30],
                [38, 39],
                [47, 48],
                [56, 57],
                [65, 66],
                [74, 75]])
```

```
In [52]: A[2:4, 2:4]
```

```
Out[52]: array([[20, 21],
                [29, 30]])
```

```
In [53]: A[-1]
```

```
Out[53]: array([72, 73, 74, 75, 76, 77, 78, 79, 80])
```

- Также может использоваться индексация при помощи списков индексов (по каждой из осей):

```
In [54]: A = np.arange(81).reshape(9, -1)
A
```

```
Out[54]: array([[ 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, 40, 41, 42, 43, 44],
                [45, 46, 47, 48, 49, 50, 51, 52, 53],
                [54, 55, 56, 57, 58, 59, 60, 61, 62],
                [63, 64, 65, 66, 67, 68, 69, 70, 71],
                [72, 73, 74, 75, 76, 77, 78, 79, 80]])
```

```
In [53]: A[[2, 4, 5], [0, 1, 3]]
```

```
Out[53]: array([18, 37, 48])
```

- Может также применяться логическая индексация (при помощи логических массивов):

```
In [55]: A = np.arange(11)  
A
```

```
Out[55]: array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

```
In [56]: A[A % 5 != 3]
```

```
Out[56]: array([ 0,  1,  2,  4,  5,  6,  7,  9, 10])
```

```
In [57]: A[np.logical_and(A != 7, A % 5 != 3)] # также можно использовать логические операции
```

```
Out[57]: array([ 0,  1,  2,  4,  5,  6,  9, 10])
```


Зачем?

Зачем необходимо использовать NumPy, если существуют стандартные списки/кортежи и циклы?

Причина заключается в скорости работы. Попробуем посчитать сумму поэлементных произведений 2 больших векторов:

```
In [60]: import time

A_quick_arr = np.random.normal(size = (1000000,))
B_quick_arr = np.random.normal(size = (1000000,))

A_slow_list, B_slow_list = list(A_quick_arr), list(B_quick_arr)
```

```
In [61]: start = time.clock()
ans = 0
for i in xrange(len(A_slow_list)):
    ans += A_slow_list[i] * B_slow_list[i]
print(time.clock() - start) # время выполнения в секундах

0.310231
```

```
In [62]: start = time.clock()
ans = sum([A_slow_list[i] * B_slow_list[i] for i in range(1000000)])
print(time.clock() - start)

0.322319
```

```
In [65]: start = time.clock()
ans = np.sum(A_quick_arr * B_quick_arr)
print(time.clock() - start)

0.005845
```

```
In [69]: start = time.clock()
ans = A_quick_arr.dot(B_quick_arr)
print(time.clock() - start)

0.003035
```

Pandas

DataFrame и Series ¶

Чтобы эффективно работать с pandas, необходимо освоить самые главные структуры данных библиотеки: DataFrame и Series. Без понимания что они из себя представляют, невозможно в дальнейшем проводить качественный анализ.

Series

Структура/объект Series представляет из себя объект, похожий на одномерный массив (питоновский список, например), но отличительной его чертой является наличие ассоциированных меток, т.н. индексов, вдоль каждого элемента из списка. Такая особенность превращает его в ассоциативный массив или словарь в Python.

```
In [1]: import pandas as pd
my_series = pd.Series([5, 6, 7, 8, 9, 10])
my_series
```

```
Out[1]: 0      5
        1      6
        2      7
        3      8
        4      9
        5     10
        dtype: int64
```

Словари в Python -
неупорядоченные коллекции
произвольных объектов с
доступом по ключу. Их иногда ещё
называют ассоциативными
массивами или хеш-таблицами.

В строковом представлении объекта Series, индекс находится слева, а сам элемент справа. Если индекс явно не задан, то pandas автоматически создаёт RangeIndex от 0 до N-1, где N общее количество элементов. Также стоит обратить, что у Series есть тип хранимых элементов, в нашем случае это int64, т.к. мы передали целочисленные значения.

У объекта Series есть атрибуты через которые можно получить список элементов и индексы, это values и index соответственно.

```
In [2]: my_series.index
```

```
Out[2]: RangeIndex(start=0, stop=6, step=1)
```

```
In [3]: my_series.values
```

```
Out[3]: array([ 5,  6,  7,  8,  9, 10], dtype=int64)
```

```
In [4]: my_series[4]
```

```
Out[4]: 9
```

Индексы можно задавать явно:

```
In [8]: my_series2 = pd.Series([5, 6, 7, 8, 9, 10], index=['a', 'b', 'c', 'd', 'e', 'f'])  
my_series2['f']
```

```
Out[8]: 10
```

Делать выборку по нескольким индексам и осуществлять групповое присваивание:

```
In [9]: my_series2[['a', 'b', 'f']]
```

```
Out[9]: a      5  
b      6  
f     10  
dtype: int64
```

```
In [12]: my_series2[['a', 'b', 'f']] = 0  
my_series2
```

```
Out[12]: a      0  
b      0  
c      7  
d      8  
e      9  
f      0  
dtype: int64
```

Фильтровать Series как душе заблагорассудится, а также применять математические операции и многое другое:

```
In [13]: my_series2[my_series2 > 0]
```

```
Out[13]: c    7
         d    8
         e    9
         dtype: int64
```

```
In [14]: my_series2[my_series2 > 0] * 2
```

```
Out[14]: c    14
         d    16
         e    18
         dtype: int64
```

Если Series напоминает нам словарь, где ключом является индекс, а значением сам элемент, то можно сделать так:

```
In [17]: my_series3 = pd.Series({'a': 5, 'b': 6, 'c': 7, 'd': 8})
         my_series3
```

```
Out[17]: a    5
         b    6
         c    7
         d    8
         dtype: int64
```

У объекта Series и его индекса есть атрибут name, задающий имя объекту и индексу соответственно.

```
In [18]: my_series3.name = 'numbers'
         my_series3.index.name = 'letters'
         my_series3
```

```
Out[18]: letters
         a    5
         b    6
         c    7
         d    8
         Name: numbers, dtype: int64
```

Индекс можно поменять "на лету", присвоив список атрибуту index объекта Series

Индекс можно поменять "на лету", присвоив список атрибуту index объекта Series

```
In [19]: my_series3.index = ['A', 'B', 'C', 'D']  
my_series3
```

```
Out[19]: A    5  
        B    6  
        C    7  
        D    8  
        Name: numbers, dtype: int64
```

DataFrame

Объект DataFrame лучше всего представлять себе в виде обычной таблицы и это правильно, ведь DataFrame является табличной структурой данных. В любой таблице всегда присутствуют строки и столбцы. Столбцами в объекте DataFrame выступают объекты Series, строки которых являются их непосредственными элементами.

DataFrame проще всего сконструировать на примере питоновского словаря:

```
In [20]: df = pd.DataFrame({  
...     'country': ['Kazakhstan', 'Russia', 'Belarus', 'Ukraine'],  
...     'population': [17.04, 143.5, 9.5, 45.5],  
...     'square': [2724902, 17125191, 207600, 603628]  
... })  
df
```

```
Out[20]:
```

	country	population	square
0	Kazakhstan	17.04	2724902
1	Russia	143.50	17125191
2	Belarus	9.50	207600
3	Ukraine	45.50	603628

Чтобы убедиться, что столбец в DataFrame это Series, извлекаем любой:

```
In [21]: df['country']
```

```
Out[21]: 0    Kazakhstan
         1      Russia
         2     Belarus
         3     Ukraine
         Name: country, dtype: object
```

Объект DataFrame имеет 2 индекса: по строкам и по столбцам. Если индекс по строкам явно не задан (например, колонка по которой нужно их строить), то pandas задаёт целочисленный индекс RangeIndex от 0 до N-1, где N это количество строк в таблице.

```
In [22]: df.columns
```

```
Out[22]: Index(['country', 'population', 'square'], dtype='object')
```

```
In [23]: df.index
```

```
Out[23]: RangeIndex(start=0, stop=4, step=1)
```

Доступ по индексу в DataFrame Индекс по строкам можно задать разными способами, например, при формировании самого объекта DataFrame или "на лету":

```
In [24]: df = pd.DataFrame({
...     'country': ['Kazakhstan', 'Russia', 'Belarus', 'Ukraine'],
...     'population': [17.04, 143.5, 9.5, 45.5],
...     'square': [2724902, 17125191, 207600, 603628]
... }, index=['KZ', 'RU', 'BY', 'UA'])
df
```

```
Out[24]:
```

	country	population	square
KZ	Kazakhstan	17.04	2724902
RU	Russia	143.50	17125191
BY	Belarus	9.50	207600
UA	Ukraine	45.50	603628

```
In [25]: df.index = ['KZ', 'RU', 'BY', 'UA']
df.index.name = 'Country Code'
df
```

Out[25]:

	country	population	square
Country Code			
KZ	Kazakhstan	17.04	2724902
RU	Russia	143.50	17125191
BY	Belarus	9.50	207600
UA	Ukraine	45.50	603628

Как видно, индексу было задано имя - Country Code. Отмечу, что объекты Series из DataFrame будут иметь те же индексы, что и объект DataFrame:

```
In [26]: df['country']
```

Out[26]: Country Code
KZ Kazakhstan
RU Russia
BY Belarus
UA Ukraine
Name: country, dtype: object

Доступ к строкам по индексу возможен несколькими способами:

.loc - используется для доступа по строковой метке

.iloc - используется для доступа по числовому значению (начиная от 0)

```
In [27]: df.loc['KZ']
```

Out[27]: country Kazakhstan
population 17.04
square 2724902
Name: KZ, dtype: object


```
In [28]: df.iloc[0]
```

```
Out[28]: country      Kazakhstan  
population      17.04  
square      2724902  
Name: KZ, dtype: object
```

Можно делать выборку по индексу и интересующим колонкам:

```
In [29]: df.loc[['KZ', 'RU'], 'population']
```

```
Out[29]: Country Code  
KZ      17.04  
RU     143.50  
Name: population, dtype: float64
```

Как можно заметить, .loc в квадратных скобках принимает 2 аргумента: интересующий индекс, в том числе поддерживается слайсинг и колонки.

```
In [30]: df.loc['KZ':'BY', :]
```

```
Out[30]:
```

	country	population	square
Country Code			
KZ	Kazakhstan	17.04	2724902
RU	Russia	143.50	17125191
BY	Belarus	9.50	207600

Фильтровать DataFrame с помощью т.н. булевых массивов:

```
In [31]: df[df.population > 10][['country', 'square']]
```

```
Out[31]:
```

	country	square
Country Code		
KZ	Kazakhstan	2724902
RU	Russia	17125191
UA	Ukraine	603628

```
In [32]: df.reset_index()
```

Out[32]:

	Country Code	country	population	square
0	KZ	Kazakhstan	17.04	2724902
1	RU	Russia	143.50	17125191
2	BY	Belarus	9.50	207600
3	UA	Ukraine	45.50	603628

pandas при операциях над DataFrame, возвращает новый объект DataFrame.

Добавим новый столбец, в котором население (в миллионах) поделим на площадь страны, получив тем самым плотность:

```
In [34]: df['density'] = df['population'] / df['square'] * 1000000  
df
```

Out[34]:

	country	population	square	density	
Country Code					
	KZ	Kazakhstan	17.04	2724902	6.253436
	RU	Russia	143.50	17125191	8.379469
	BY	Belarus	9.50	207600	45.761079
	UA	Ukraine	45.50	603628	75.377550

Не нравится новый столбец? Не проблема, удалим его:

```
In [35]: df.drop(['density'], axis='columns')
```

Out[35]:

	country	population	square
Country Code			
KZ	Kazakhstan	17.04	2724902
RU	Russia	143.50	17125191
BY	Belarus	9.50	207600
UA	Ukraine	45.50	603628

Особо ленивые могут просто написать `del df['density']`.

Переименовывать столбцы нужно через метод `rename`:

```
In [36]: df = df.rename(columns={'Country Code': 'country_code'})
df
```

Out[36]:

	country	population	square	density
Country Code				
KZ	Kazakhstan	17.04	2724902	6.253436
RU	Russia	143.50	17125191	8.379469
BY	Belarus	9.50	207600	45.761079
UA	Ukraine	45.50	603628	75.377550

Группировка и агрегирование в pandas

Группировка данных один из самых часто используемых методов при анализе данных. В pandas за группировку отвечает метод `.groupby`.

```
In [37]: titanic_df = pd.read_csv('titanic.csv')
print(titanic_df.head())
```

	PassengerId	Survived	Pclass	\
0	1	0	3	
1	2	1	1	
2	3	1	3	
3	4	1	1	
4	5	0	3	

	Name	Sex	Age	SibSp	\
0	Braund, Mr. Owen Harris	male	22.0	1	
1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	
2	Heikkinen, Miss. Laina	female	26.0	0	
3	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	
4	Allen, Mr. William Henry	male	35.0	0	

	Parch	Ticket	Fare	Cabin	Embarked
0	0	A/5 21171	7.2500	NaN	S
1	0	PC 17599	71.2833	C85	C
2	0	STON/O2. 3101282	7.9250	NaN	S
3	0	113803	53.1000	C123	S
4	0	373450	8.0500	NaN	S

Необходимо подсчитать, сколько женщин и мужчин выжило, а сколько нет. В этом нам поможет метод `.groupby`.

```
In [39]: print(titanic_df.groupby(['Sex', 'Survived'])['PassengerId'].count())
```

Sex	Survived	
female	0	81
	1	233
male	0	468
	1	109

Name: PassengerId, dtype: int64

А теперь проанализируем в разрезе класса каюты:

```
In [41]: print(titanic_df.groupby(['Pclass', 'Survived'])['PassengerId'].count())
```

Pclass	Survived	
1	0	80
	1	136
2	0	97
	1	87
3	0	372
	1	119

Name: PassengerId, dtype: int64

```
In [71]: import pandas as pd
%matplotlib inline
```

Pandas (Python Data Analysis Library) — библиотека языка Python для [удобных] обработки и анализа данных.

Рассмотрим данную библиотеку на примере [данных соревнования](#) о предсказании судьбы пассажиров лайнера "Титаник".
Имеется некоторая информация о пассажирах, по которой необходимо предсказать выживаемость каждого из них.

- Какого типа данная задача?
- Что является объектами?
- Что является ответами?
- Какие могут быть признаки? Какие у них типы?

Команда `%matplotlib inline` указывает, что график необходимо построить все в той же оболочке Jupyter, но теперь он выводится как обычная картинка.

Загрузка данных в **pandas** происходит в специальный объект типа **DataFrame**:

```
In [72]: pass_data = pd.read_csv('titanic.csv')
pass_data
```

Out[72]:

	pclass	survived	name	sex	age	sibsp	parch	ticket	fare	cabin	embarked	boat	body	home
0	1	1	Allen, Miss. Elisabeth Walton	female	29.00	0	0	24160	211.3375	B5	S	2	NaN	St Lot MO
1	1	1	Allison, Master. Hudson Trevor	male	0.92	1	2	113781	151.5500	C22 C26	S	11	NaN	Montr PQ / Chest ON
2	1	0	Allison, Miss. Helen Loraine	female	2.00	1	2	113781	151.5500	C22 C26	S	NaN	NaN	Montr PQ / Chest ON
3	1	0	Allison, Mr. Hudson Joshua Creighton	male	30.00	1	2	113781	151.5500	C22 C26	S	NaN	135.0	Montr PQ / Chest ON

1309 rows × 14 columns

Как видно, данные представляют из себя таблицу, где строка — объект, столбец — признак. Для экономии места можно выводить заданное количество первых строк объекта при помощи метода `head()`:

In [73]: `pass_data.head(3)`

Out[73]:

	pclass	survived	name	sex	age	sibsp	parch	ticket	fare	cabin	embarked	boat	body	home.dest
0	1	1	Allen, Miss. Elisabeth Walton	female	29.00	0	0	24160	211.3375	B5	S	2	NaN	St Louis, MO
1	1	1	Allison, Master. Hudson Trevor	male	0.92	1	2	113781	151.5500	C22 C26	S	11	NaN	Montreal, PQ / Chesterville, ON
2	1	0	Allison, Miss. Helen Loraine	female	2.00	1	2	113781	151.5500	C22 C26	S	NaN	NaN	Montreal, PQ / Chesterville, ON

Посмотрим на признаки:

```
In [17]: pass_data.columns
```

```
Out[17]: Index([u'pclass', u'survived', u'name', u'sex', u'age', u'sibsp', u'parch',  
              u'ticket', u'fare', u'cabin', u'embarked', u'boat', u'body',  
              u'home.dest'],  
              dtype='object')
```

По данным можно индексироваться при помощи номеров строк/столбцов или названий признаков:

```
In [74]: pass_data[2:5]
```

```
Out[74]:
```

	pclass	survived	name	sex	age	sibsp	parch	ticket	fare	cabin	embarked	boat	body	home.dest
2	1	0	Allison, Miss. Helen Loraine	female	2.0	1	2	113781	151.55	C22 C26	S	NaN	NaN	Montreal, PQ / Chesterville, ON
3	1	0	Allison, Mr. Hudson Joshua Creighton	male	30.0	1	2	113781	151.55	C22 C26	S	NaN	135.0	Montreal, PQ / Chesterville, ON
4	1	0	Allison, Mrs. Hudson J C (Bessie Waldo Daniels)	female	25.0	1	2	113781	151.55	C22 C26	S	NaN	NaN	Montreal, PQ / Chesterville, ON

```
In [75]: pass_data.iloc[1:5, 1:3]
```

```
Out[75]:
```

	survived	name
1	1	Allison, Master. Hudson Trevor
2	0	Allison, Miss. Helen Loraine
3	0	Allison, Mr. Hudson Joshua Creighton
4	0	Allison, Mrs. Hudson J C (Bessie Waldo Daniels)

```
In [20]: pass_data['name'].head()
```

```
Out[20]: 0      Allen, Miss. Elisabeth Walton  
1      Allison, Master. Hudson Trevor  
2      Allison, Miss. Helen Loraine  
3      Allison, Mr. Hudson Joshua Creighton  
4      Allison, Mrs. Hudson J C (Bessie Waldo Daniels)  
Name: name, dtype: object
```



```
In [76]: pass_data[['name', 'sex', 'parch']].head()
```

Out[76]:

	name	sex	parch
0	Allen, Miss. Elisabeth Walton	female	0
1	Allison, Master. Hudson Trevor	male	2
2	Allison, Miss. Helen Loraine	female	2
3	Allison, Mr. Hudson Joshua Creighton	male	2
4	Allison, Mrs. Hudson J C (Bessie Waldo Daniels)	female	2

Также действует и логическая индексация, которая позволяет рассматривать определенные группы объектов:

```
In [77]: pass_data[pass_data['sex'] == 'female'].head() # женщины на борту
```

Out[77]:

	pclass	survived	name	sex	age	sibsp	parch	ticket	fare	cabin	embarked	boat	body	home.dest
0	1	1	Allen, Miss. Elisabeth Walton	female	29.0	0	0	24160	211.3375	B5	S	2	NaN	St Louis, MO
2	1	0	Allison, Miss. Helen Loraine	female	2.0	1	2	113781	151.5500	C22 C26	S	NaN	NaN	Montreal, PQ / Chesterville, ON
4	1	0	Allison, Mrs. Hudson J C (Bessie Waldo Daniels)	female	25.0	1	2	113781	151.5500	C22 C26	S	NaN	NaN	Montreal, PQ / Chesterville, ON
6	1	1	Andrews, Miss. Kornelia Theodosia	female	63.0	1	0	13502	77.9583	D7	S	10	NaN	Hudson, NY
8	1	1	Appleton, Mrs. Edward Dale (Charlotte Lamson)	female	53.0	2	0	11769	51.4792	C101	S	D	NaN	Bayside, Queens, NY

```
In [78]: # женщины старше 60 и мужчины на борту
pass_data[(pass_data['sex'] == 'female') & (pass_data['age'] >= 60) | (pass_data['sex'] == 'male')].head()
```

Out[78]:

	pclass	survived	name	sex	age	sibsp	parch	ticket	fare	cabin	embarked	boat	body	home.dest
1	1	1	Allison, Master. Hudson Trevor	male	0.92	1	2	113781	151.5500	C22 C26	S	11	NaN	Montreal, PQ / Chesterville, ON
3	1	0	Allison, Mr. Hudson Joshua Creighton	male	30.00	1	2	113781	151.5500	C22 C26	S	NaN	135.0	Montreal, PQ / Chesterville, ON
5	1	1	Anderson, Mr. Harry	male	48.00	0	0	19952	26.5500	E12	S	3	NaN	New York, NY
6	1	1	Andrews, Miss. Kornelia Theodosia	female	63.00	1	0	13502	77.9583	D7	S	10	NaN	Hudson, NY
7	1	0	Andrews, Mr. Thomas Jr	male	39.00	0	0	112050	0.0000	A36	S	NaN	NaN	Belfast, NI

Пример

Посмотрим, сколько на борту было относительно молодых женщин, путешествующих в одиночку. Скорее всего, довольно мало, потому что в такое длительное путешествие молодых девушек одних не отпустили бы опекающие родственники.

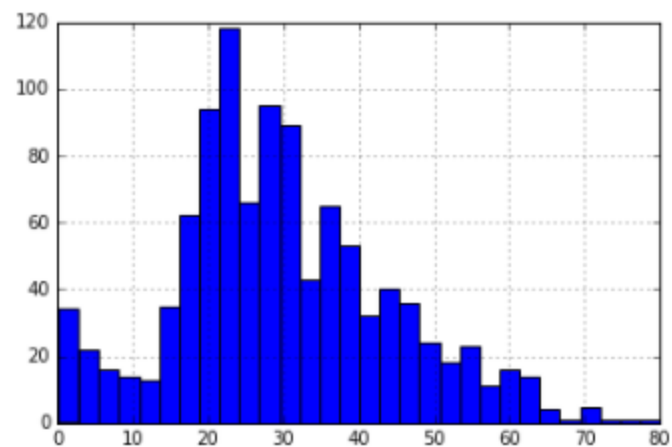
```
In [80]: pass_data[(pass_data.sex == 'female') &  
                (pass_data.age > 18) &  
                (pass_data.age < 25) &  
                (pass_data.sibsp == 0) &  
                (pass_data.parch == 0)].shape
```

Out[80]: (41, 14)

Кроме того, для заданного признака можно построить гистограмму:

```
In [82]: pass_data.age.hist(bins = 30)
```

Out[82]: <matplotlib.axes._subplots.AxesSubplot at 0x10f7b47d0>



Редактирование DataFrame

- Переименование признаков

```
In [83]: pass_data.rename(columns={'sex': 'Sex'}, inplace=True)
pass_data.head()
```

Out[83]:

	pclass	survived	name	Sex	age	sibsp	parch	ticket	fare	cabin	embarked	boat	body	home.dest
0	1	1	Allen, Miss. Elisabeth Walton	female	29.00	0	0	24160	211.3375	B5	S	2	NaN	St Louis, MO
1	1	1	Allison, Master. Hudson Trevor	male	0.92	1	2	113781	151.5500	C22 C26	S	11	NaN	Montreal, PQ / Chesterville, ON
2	1	0	Allison, Miss. Helen Loraine	female	2.00	1	2	113781	151.5500	C22 C26	S	NaN	NaN	Montreal, PQ / Chesterville, ON
3	1	0	Allison, Mr. Hudson Joshua Creighton	male	30.00	1	2	113781	151.5500	C22 C26	S	NaN	135.0	Montreal, PQ / Chesterville, ON
4	1	0	Allison, Mrs. Hudson J C (Bessie Waldo Daniels)	female	25.00	1	2	113781	151.5500	C22 C26	S	NaN	NaN	Montreal, PQ / Chesterville, ON

- Применение преобразования к существующему признаку. Например, выделим фамилию:

```
In [84]: def get_last_name(name):
return name.split(',')[0].strip()

last_names = pass_data['name'].apply(get_last_name)
last_names.head()
```

Out[84]:

```
0    Allen
1    Allison
2    Allison
3    Allison
4    Allison
Name: name, dtype: object
```

- Добавление признака

```
In [85]: pass_data['Last_name'] = last_names
pass_data.head()
```

Out[85]:

ass	survived	name	Sex	age	sibsp	parch	ticket	fare	cabin	embarked	boat	body	home.dest	Last_name
	1	Allen, Miss. Elisabeth Walton	female	29.00	0	0	24160	211.3375	B5	S	2	NaN	St Louis, MO	Allen
	1	Allison, Master. Hudson Trevor	male	0.92	1	2	113781	151.5500	C22 C26	S	11	NaN	Montreal, PQ / Chesterville, ON	Allison
	0	Allison, Miss. Helen Lorraine	female	2.00	1	2	113781	151.5500	C22 C26	S	NaN	NaN	Montreal, PQ / Chesterville, ON	Allison
	0	Allison, Mr. Hudson Joshua Creighton	male	30.00	1	2	113781	151.5500	C22 C26	S	NaN	135.0	Montreal, PQ / Chesterville, ON	Allison
	0	Allison, Mrs. Hudson J C (Bessie Waldo Daniels)	female	25.00	1	2	113781	151.5500	C22 C26	S	NaN	NaN	Montreal, PQ / Chesterville, ON	Allison

- Удаление признака

```
In [86]: pass_data.drop('Last_name', axis=1, inplace=True)
pass_data.head()
```

Out[86]:

	pclass	survived	name	Sex	age	sibsp	parch	ticket	fare	cabin	embarked	boat	body	home.dest
0	1	1	Allen, Miss. Elisabeth Walton	female	29.00	0	0	24160	211.3375	B5	S	2	NaN	St Louis, MO
1	1	1	Allison, Master. Hudson Trevor	male	0.92	1	2	113781	151.5500	C22 C26	S	11	NaN	Montreal, PQ / Chesterville, ON
2	1	0	Allison, Miss. Helen Lorraine	female	2.00	1	2	113781	151.5500	C22 C26	S	NaN	NaN	Montreal, PQ / Chesterville, ON
3	1	0	Allison, Mr. Hudson Joshua Creighton	male	30.00	1	2	113781	151.5500	C22 C26	S	NaN	135.0	Montreal, PQ / Chesterville, ON
4	1	0	Allison, Mrs. Hudson J C (Bessie Waldo Daniels)	female	25.00	1	2	113781	151.5500	C22 C26	S	NaN	NaN	Montreal, PQ / Chesterville, ON

- Работа с пропущенными данными

Методы `isnull()` и `notnull()` позволяют получить бинарный массив, отражающий отсутствие или наличие данных для каждого из объектов соответственно:

```
In [87]: pass_data['boat'].isnull().head()
```

```
Out[87]: 0    False
1    False
2     True
3     True
4     True
Name: boat, dtype: bool
```

In [88]: `pass_data[pass_data['boat'].notnull()].head() # пассажиры с известным номером шлюпки эвакуации`

Out[88]:

	pclass	survived	name	Sex	age	sibsp	parch	ticket	fare	cabin	embarked	boat	body	home.dest
0	1	1	Allen, Miss. Elisabeth Walton	female	29.00	0	0	24160	211.3375	B5	S	2	NaN	St Louis, MO
1	1	1	Allison, Master. Hudson Trevor	male	0.92	1	2	113781	151.5500	C22 C26	S	11	NaN	Montreal, PQ / Chesterville, ON
5	1	1	Anderson, Mr. Harry	male	48.00	0	0	19952	26.5500	E12	S	3	NaN	New York, NY
6	1	1	Andrews, Miss. Kornelia Theodosia	female	63.00	1	0	13502	77.9583	D7	S	10	NaN	Hudson, NY
8	1	1	Appleton, Mrs. Edward Dale (Charlotte Lamson)	female	53.00	2	0	11769	51.4792	C101	S	D	NaN	Bayside, Queens, NY

- Сортировка объектов/признаков

In [89]: `pass_data.sort_values(by=['pclass', 'fare'], ascending=True).head()`

Out[89]:

	pclass	survived	name	Sex	age	sibsp	parch	ticket	fare	cabin	embarked	boat	body	home.dest
7	1	0	Andrews, Mr. Thomas Jr	male	39.0	0	0	112050	0.0	A36	S	NaN	NaN	Belfast, NI
70	1	0	Chisholm, Mr. Roderick Robert Crispin	male	NaN	0	0	112051	0.0	NaN	S	NaN	NaN	Liverpool, England / Belfast
125	1	0	Fry, Mr. Richard	male	NaN	0	0	112058	0.0	B102	S	NaN	NaN	NaN
150	1	0	Harrison, Mr. William	male	40.0	0	0	112059	0.0	B94	S	NaN	110.0	NaN
170	1	1	Ismay, Mr. Joseph Bruce	male	49.0	0	0	112058	0.0	B52 B54 B56	S	C	NaN	Liverpool

```
In [86]: pass_data.sort_values(by=['pclass', 'fare'], ascending=[True, False]).head()
```

Out[86]:

	pclass	survived	name	Sex	age	sibsp	parch	ticket	fare	cabin	embarked	boat	body	home.dest
49	1	1	Cardeza, Mr. Thomas Drake Martinez	male	36.0	0	1	PC 17755	512.3292	B51 B53 B55	C	3	NaN	Austria-Hungary / Germantown, Philadelphia, PA
50	1	1	Cardeza, Mrs. James Warburton Martinez (Charlo...	female	58.0	0	1	PC 17755	512.3292	B51 B53 B55	C	3	NaN	Germantown, Philadelphia, PA
183	1	1	Lesurer, Mr. Gustave J	male	35.0	0	0	PC 17755	512.3292	B101	C	3	NaN	NaN
302	1	1	Ward, Miss. Anna	female	35.0	0	0	PC 17755	512.3292	NaN	C	3	NaN	NaN
111	1	1	Fortune, Miss. Alice Elizabeth	female	24.0	3	2	19950	263.0000	C23 C25 C27	S	10	NaN	Winnipeg, MB

Группировка данных

Группировка при помощи метода `groupby` позволяет объединять данные в группы по одному или нескольким признакам и считать по ним общую статистику.

```
In [90]: pass_data.groupby('Sex') # разбиение всех объектов на 2 группы по полу
```

```
Out[90]: <pandas.core.groupby.DataFrameGroupBy object at 0x1102bf4d0>
```

```
In [91]: pass_data.groupby('Sex')['pclass'].value_counts()
```

```
Out[91]: Sex      pclass
female  3          216
         1          144
         2          106
male     3          493
         1          179
         2          171
Name: pclass, dtype: int64
```

```
In [92]: pass_data.groupby('pclass')['fare'].describe()
```

```
/usr/local/lib/python2.7/site-packages/numpy/lib/function_base.py:3823: RuntimeWarning: Invalid value encountered in percentile
RuntimeWarning)
```

```
Out[92]: pclass
1      count    323.000000
      mean      87.508992
      std       80.447178
      min        0.000000
      25%       30.695800
      50%       60.000000
      75%      107.662500
      max      512.329200
2      count    277.000000
      mean      21.179196
      std       13.607122
      min        0.000000
      25%       13.000000
      50%       15.045800
      75%       26.000000
      max       73.500000
3      count    708.000000
      mean      13.302889
      std       11.494358
      min        0.000000
      25%        NaN
      50%        NaN
      75%        NaN
      max       69.550000
Name: fare, dtype: float64
```

```
In [93]: pass_data.groupby('Sex')['age'].mean() # средний возраст для пассажиров каждого из полов
```

```
Out[93]: Sex
female    28.687088
male      30.585228
Name: age, dtype: float64
```

Как известно, в первую очередь спасали женщин и детей в порядке повышения номера класса, в связи с этим доля выживших женщин, вероятно, будет выше, чем доля выживших мужчин. Проверим:

```
In [94]: pass_data.groupby('Sex')['survived'].mean()
```

```
Out[94]: Sex
female    0.727468
male      0.190985
Name: survived, dtype: float64
```

Аналогично для пассажиров различных классов:

```
In [95]: pass_data.groupby('pclass')['survived'].mean()
```

```
Out[95]: pclass
1      0.619195
2      0.429603
3      0.255289
Name: survived, dtype: float64
```

Рассмотренные выше статистические данные могут быть рассмотрены и в виде стандартного DataFrame:

```
In [96]: pass_data.groupby('Sex', as_index=False)['survived'].mean()
```

```
Out[96]:
```

	Sex	survived
0	female	0.727468
1	male	0.190985

apply

Returns:
applied : Series or DataFrame

pandas.DataFrame.apply

`DataFrame.apply(func, axis=0, broadcast=None, raw=False, reduce=None, result_type=None, args=(), **kwargs)`

Apply a function along an axis of the DataFrame.

[\[source\]](#)

Objects passed to the function are Series objects whose index is either the DataFrame's index (`axis=0`) or the DataFrame's columns (`axis=1`). By default (`result_type=None`), the final return type is inferred from the return type of the applied function. Otherwise, it depends on the `result_type` argument.

```
>>> df = pd.DataFrame([[4, 9],] * 3, columns=['A', 'B'])
>>> df
   A  B
0  4  9
1  4  9
2  4  9
```

Using a numpy universal function (in this case the same as `np.sqrt(df)`):

```
>>> df.apply(np.sqrt)
   A  B
0  2.0  3.0
1  2.0  3.0
2  2.0  3.0
```

Using a reducing function on either axis

```
>>> df.apply(np.sum, axis=0)
A    12
B    27
dtype: int64
```

```
>>> df.apply(np.sum, axis=1)
0     13
1     13
2     13
dtype: int64
```

Another frequent operation is applying a function on 1D arrays to each column or row. DataFrame's apply method does exactly this:

```
In [116]: frame = DataFrame(np.random.randn(4, 3), columns=list('bde'), index=['Utah', 'Ohio',  
In [117]: frame  
Out[117]:
```

	b	d	e
Utah	-0.029638	1.081563	1.280300
Ohio	0.647747	0.831136	-1.549481
Texas	0.513416	-0.884417	0.195343
Oregon	-0.485454	-0.477388	-0.309548

```
  
In [118]: f = lambda x: x.max() - x.min()  
  
In [119]: frame.apply(f)  
Out[119]:  
b    1.133201  
d    1.965980  
e    2.829781  
dtype: float64
```

Выполняется поиск максимума и минимума по каждому столбцу (Series)

applymap

Применяется к
каждому элементу
DataFrame.

Интересен
возвращаемый
результат!!!

pandas.DataFrame.applymap

`DataFrame.applymap(func)`

[\[source\]](#)

Apply a function to a DataFrame that is intended to operate elementwise, i.e. like doing `map(func, series)` for each series in the DataFrame

Parameters:

func : function

Python function, returns a single value from a single value

Returns:

applied : DataFrame

See also:

[DataFrame.apply](#)

For operations on rows/columns

Examples

```
>>> df = pd.DataFrame(np.random.randn(3, 3))
>>> df
   0         1         2
0 -0.029638  1.081563  1.280300
1  0.647747  0.831136 -1.549481
2  0.513416 -0.884417  0.195343
>>> df = df.applymap(lambda x: '%.2f' % x)
>>> df
   0         1         2
0 -0.03      1.08      1.28
1  0.65      0.83     -1.55
2  0.51     -0.88      0.20
```

Many of the most common array statistics (like sum and mean) are DataFrame methods, so using apply is not necessary.

Element-wise Python functions can be used, too. Suppose you wanted to compute a formatted string from each floating point value in frame. You can do this with applymap:

```
In [120]: format = lambda x: '%.2f' % x
```

```
In [121]: frame.applymap(format)
```

```
Out[121]:
```

	b	d	e
Utah	-0.03	1.08	1.28
Ohio	0.65	0.83	-1.55
Texas	0.51	-0.88	0.20
Oregon	-0.49	-0.48	-0.31

Map

pandas.Series.map

`Series.map(arg, na_action=None)`

[\[source\]](#)

Map values of Series using input correspondence (a dict, Series, or function).

Parameters:

arg : *function, dict, or Series*

Mapping correspondence.

na_action : *{None, 'ignore'}*

If 'ignore', propagate NA values, without passing them to the mapping correspondence.

Returns:

y : *Series*

Same index as caller.

The reason for the name `applymap` is that `Series` has a `map` method for applying an element-wise function:

```
In [122]: frame['e'].map(format)
Out[122]:
Utah      1.28
Ohio     -1.55
Texas      0.20
Oregon    -0.31
Name: e, dtype: object
```

Apply, Applymap, Map

`apply` – обрабатывает элементы `DataFrame` (т.е. `Series`), например может сложить два `Series`.

`applymap` – применяет функцию к элементам `Series` для всего `DataFrame`

`map` – применяет функцию к элементам выбранной `Series`

Map: It iterates over each element of a series.

`df['column1'].map(lambda x: 10+x)`, this will add 10 to each element of column1.

`df['column2'].map(lambda x: 'AV'+x)`, this will concatenate "AV" at the beginning of each element of column2 (column format is string).

Apply: As the name suggests, applies a function along any axis of the DataFrame.

`df[['column1','column2']].apply(sum)`, it will return the sum of all the values of column1 and column2.

ApplyMap: This helps to apply a function to each element of dataframe.

`func = lambda x: x+2`

`df.applymap(func)`, it will add 2 to each element of dataframe (all columns of dataframe must be numeric type)

- `DataFrame.apply` operates on entire rows or columns at a time.
- `DataFrame.applymap`, `Series.apply`, and `Series.map` operate on one element at time.
- **`apply`** takes the whole column as a parameter and then assign the result to this column
- **`applymap`** takes the separate cell value as a parameter and assign the result back to this cell.

If the function has variables that need to compare within a column/ row, use `apply`.
e.g.: `lambda x: x.max()-x.mean()`.

If the function is to be applied to each element:

- 1> If a column/row is located, use `apply`
- 2> If apply to entire dataframe, use `applymap`

pandas.Series.apply

`Series.apply(func, convert_dtype=True, args=(), **kwargs)`

[\[source\]](#)

Invoke function on values of Series. Can be ufunc (a NumPy function that applies to the entire Series) or a Python function that only works on single values

Parameters:

func : function

convert_dtype : boolean, default True

Try to find better dtype for elementwise function results. If False, leave as dtype=object

args : tuple

Positional arguments to pass to function in addition to the value

Additional keyword arguments will be passed as keywords to the function

Returns:

v : Series or DataFrame if func returns a Series

pandas.Series.map

`Series.map(arg, na_action=None)`

[\[source\]](#)

Map values of Series using input correspondence (a dict, Series, or function).

Parameters:

arg : function, dict, or Series

Mapping correspondence.

na_action : {None, 'ignore'}

If 'ignore', propagate NA values, without passing them to the mapping correspondence.

Returns:

y : Series

Same index as caller.

Seaborn

```
In [3]: # Python 2 and 3 compatibility
# pip install future
from __future__ import (absolute_import, division,
print_function, unicode_literals)
# отключим предупреждения Anaconda
import warnings
warnings.simplefilter('ignore')
# будем отображать графики прямо в jupyter'e
%pylab inline
#увеличим дефолтный размер графиков
from pylab import rcParams
rcParams['figure.figsize'] = 8, 5
import pandas as pd
import seaborn as sns
```

Populating the interactive namespace from numpy and matplotlib

```
In [4]: df = pd.read_csv('Video_Games_Sales_as_at_22_Dec_2016.csv ')
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16719 entries, 0 to 16718
Data columns (total 16 columns):
Name                16717 non-null object
Platform            16719 non-null object
Year_of_Release     16450 non-null float64
Genre               16717 non-null object
Publisher            16665 non-null object
NA_Sales             16719 non-null float64
EU_Sales             16719 non-null float64
JP_Sales             16719 non-null float64
Other_Sales          16719 non-null float64
Global_Sales         16719 non-null float64
Critic_Score         8137 non-null float64
Critic_Count         8137 non-null float64
User_Score           10015 non-null object
User_Count           7590 non-null float64
Developer            10096 non-null object
Rating              9950 non-null object
dtypes: float64(9), object(7)
memory usage: 2.0+ MB
```

```
In [5]: # Данные об оценках есть не для всех фильмов, поэтому давайте
# оставим только те записи, в которых нет пропусков с помощью метода dropna
df = df.dropna()
print(df.shape)
```

(6825, 16)

```
In [6]: # Всего в таблице 6825 объектов и 16 признаков для них. Посмотрим на несколько первых записей
# с помощью метода head, чтобы убедиться, что
# все распарсилось правильно.
useful_cols = ['Name', 'Platform', 'Year_of_Release', 'Genre', 'Global_Sales', 'Critic_Score', 'Critic_Count', 'User_Score', 'User_Count', 'Rating']
df[useful_cols].head()
```

Out[6]:

	Name	Platform	Year_of_Release	Genre	Global_Sales	Critic_Score	Critic_Count	User_Score	User_Count	Rating
0	Wii Sports	Wii	2006.0	Sports	82.53	76.0	51.0	8	322.0	E
2	Mario Kart Wii	Wii	2008.0	Racing	35.52	82.0	73.0	8.3	709.0	E
3	Wii Sports Resort	Wii	2009.0	Sports	32.77	80.0	73.0	8	192.0	E
6	New Super Mario Bros.	DS	2006.0	Platform	29.80	89.0	65.0	8.5	431.0	E
7	Wii Play	Wii	2006.0	Misc	28.92	58.0	41.0	6.6	129.0	E

```
In [7]: sales_df = df[[x for x in df.columns if 'Sales' in x] + ['Year_of_Release']]
sales_df.head()
```

Out[7]:

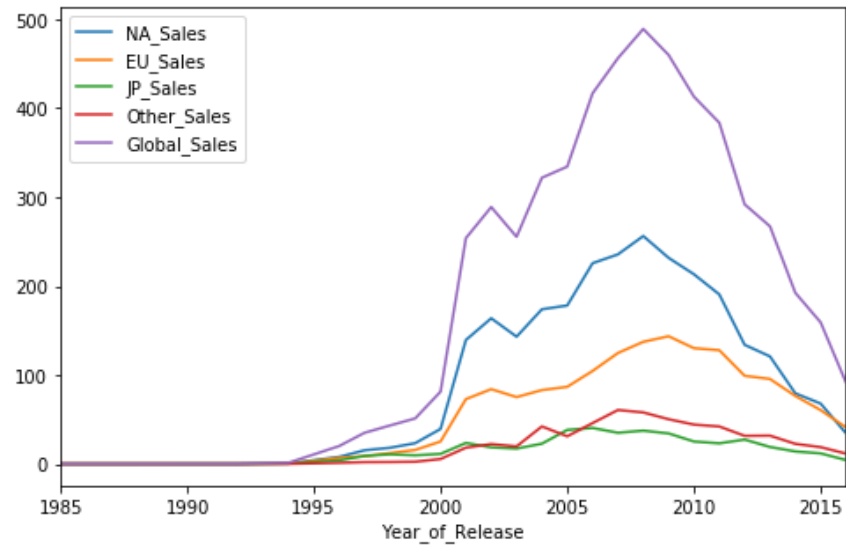
	NA_Sales	EU_Sales	JP_Sales	Other_Sales	Global_Sales	Year_of_Release
0	41.36	28.96	3.77	8.45	82.53	2006.0
2	15.68	12.76	3.79	3.29	35.52	2008.0
3	15.61	10.93	3.28	2.95	32.77	2009.0
6	11.28	9.14	6.50	2.88	29.80	2006.0
7	13.96	9.18	2.93	2.84	28.92	2006.0

```
In [8]: df1 = sales_df.groupby('Year_of_Release').sum()
print(df1)
```

	NA_Sales	EU_Sales	JP_Sales	Other_Sales	Global_Sales
Year_of_Release					
1985.0	0.00	0.03	0.00	0.01	0.03
1988.0	0.00	0.02	0.00	0.01	0.03
1992.0	0.02	0.00	0.00	0.00	0.03
1994.0	0.39	0.26	0.53	0.08	1.27
1996.0	7.91	6.88	4.06	1.24	20.10
1997.0	15.34	8.67	9.01	2.02	35.01
1998.0	18.13	12.13	10.81	2.14	43.18
1999.0	23.32	15.69	9.67	2.45	51.17
2000.0	39.34	25.20	11.27	5.49	81.24
2001.0	139.32	72.85	23.57	18.26	253.88
2002.0	163.76	84.03	18.61	22.30	288.84
2003.0	143.08	75.16	17.24	19.68	255.35
2004.0	173.88	83.01	22.74	42.14	321.78
2005.0	178.15	86.70	38.23	31.05	334.32
2006.0	225.69	104.53	40.43	45.90	416.72
2007.0	235.61	124.71	35.04	60.62	456.23
2008.0	256.25	137.31	37.42	57.89	489.12
2009.0	231.72	143.56	34.28	50.25	459.85
2010.0	213.24	130.13	25.19	44.24	412.96
2011.0	190.62	127.86	23.16	42.10	383.69
2012.0	133.94	99.08	27.36	31.57	291.93
2013.0	120.89	95.54	19.05	31.80	267.17
2014.0	79.38	76.42	14.02	22.58	192.43
2015.0	67.85	60.51	11.85	18.86	159.16
2016.0	34.52	41.03	4.34	11.59	91.56

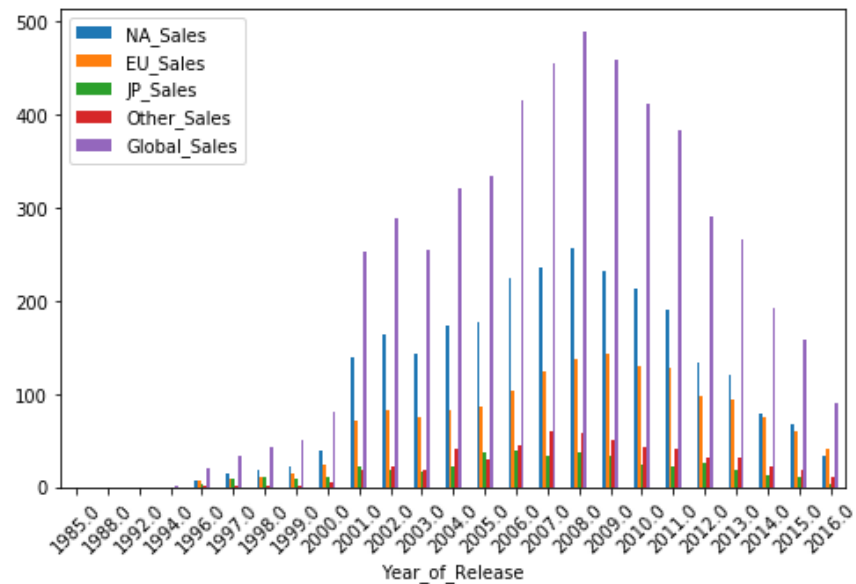
```
In [10]: sales_df.groupby('Year_of_Release').sum().plot()
```

```
Out[10]: <matplotlib.axes._subplots.AxesSubplot at 0xecd808ea58>
```

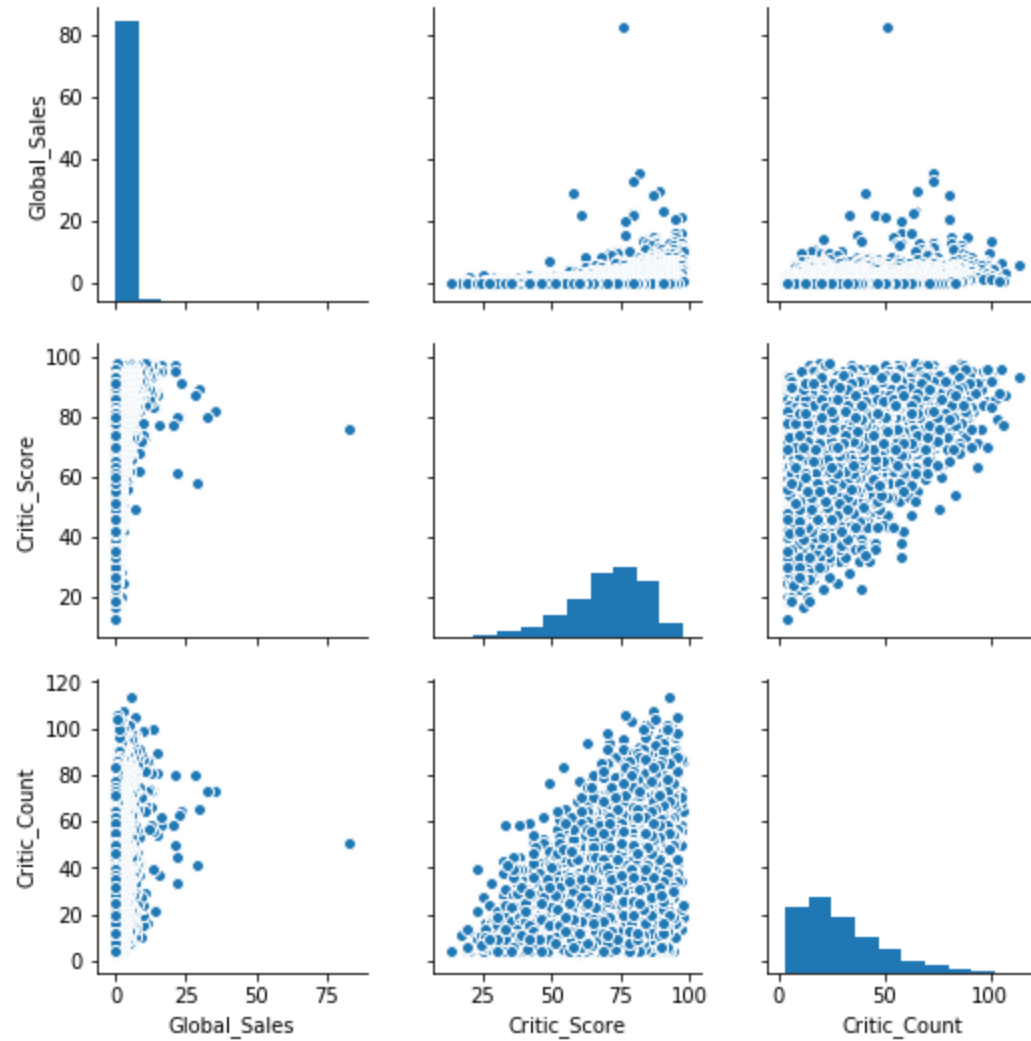


```
In [11]: sales_df.groupby('Year_of_Release').sum().plot(kind='bar', rot=45)
```

```
Out[11]: <matplotlib.axes._subplots.AxesSubplot at 0xecd8853208>
```

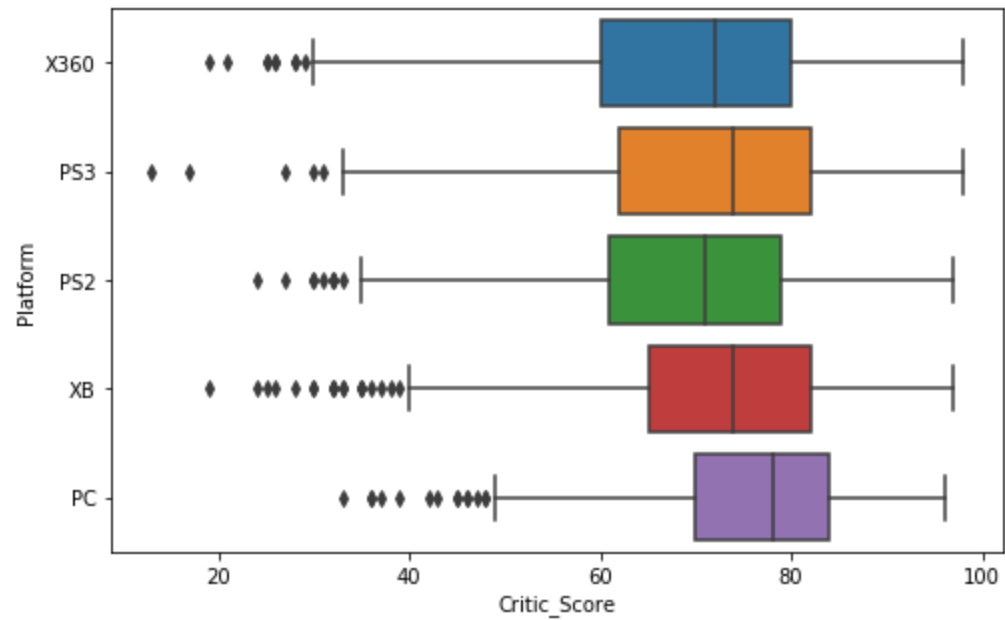


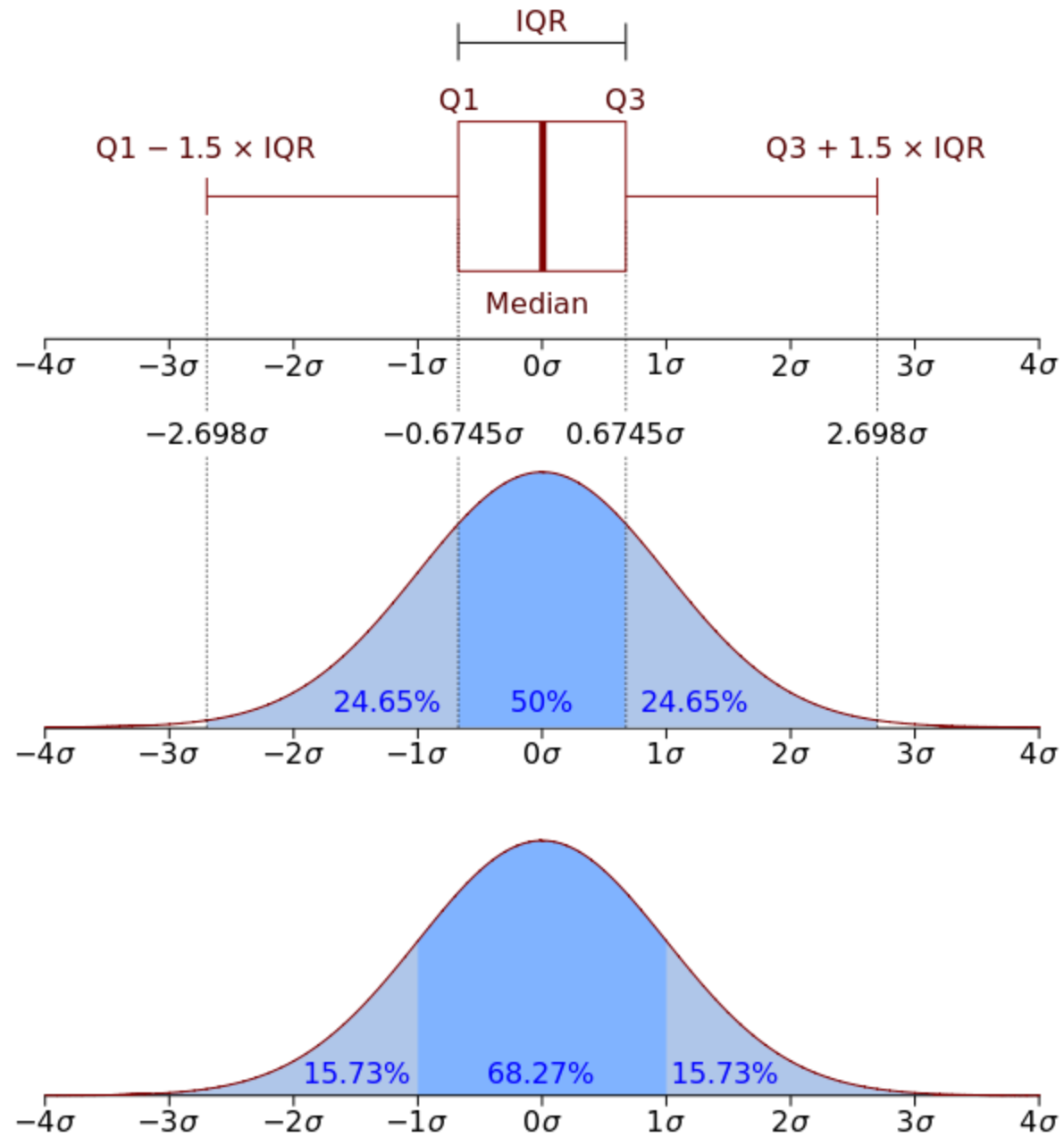

```
In [15]: cols = ['Global_Sales', 'Critic_Score', 'Critic_Count']  
sns_plot = sns.pairplot(df[cols])
```



```
In [17]: top_platforms = df.Platform.value_counts().sort_values(ascending = False).head(5).index.values
sns.boxplot(y="Platform", x="Critic_Score", data=df[df.Platform.isin(top_platforms)], orient="h")
```

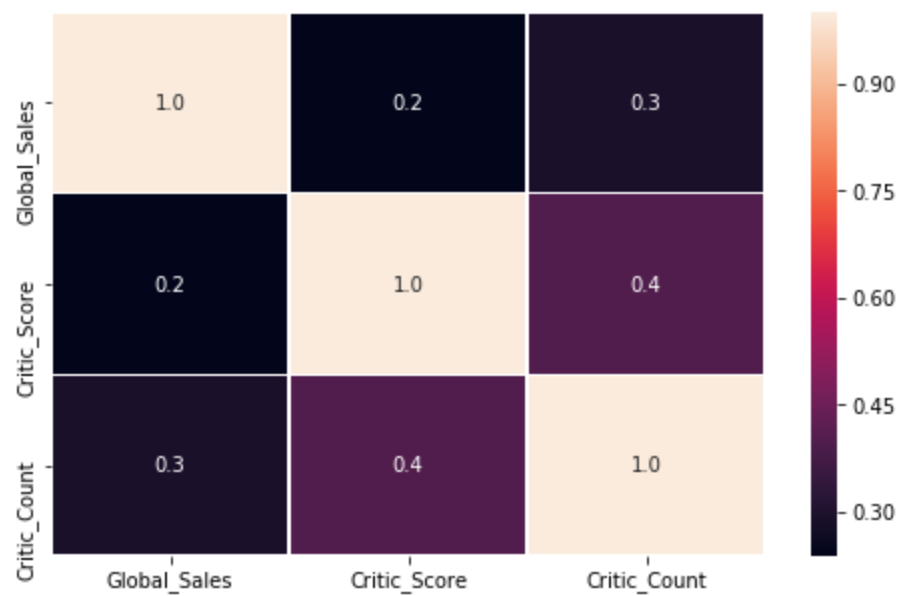
Out[17]: <matplotlib.axes._subplots.AxesSubplot at 0xecdea3f28>





```
In [34]: sns.heatmap(df[cols].corr(), annot=True, fmt=".1f", linewidths=.5)
```

```
Out[34]: <matplotlib.axes._subplots.AxesSubplot at 0xec836ba898>
```



Ссылки

- <https://khashtamov.com/ru/pandas-introduction/>
- <http://devpractice.ru/pandas-series-and-dataframe-part2/>
- <http://devpractice.ru/pandas-indexing-part3/>
- <http://devpractice.ru/pandas-work-with-nan-part4/>
- https://pandas.pydata.org/pandas-docs/stable/user_guide/groupby.html
- <https://docs.scipy.org/doc/numpy/user/basics.broadcasting.html>
- <https://stackoverflow.com/questions/29954263/what-does-the-term-broadcasting-mean-in-pandas-documentation>
- <https://stackoverflow.com/questions/51079543/pandas-groupby-apply-vs-transform-with-specific-functions>
- <https://pythonforbiologists.com/when-to-use-aggregatefiltertransform-in-pandas>
- https://pandas.pydata.org/pandas-docs/stable/user_guide/groupby.html
- https://pandas.pydata.org/pandas-docs/stable/user_guide/groupby.html#transformation
- [! https://stackoverflow.com/questions/19798153/difference-between-map-applymap-and-apply-methods-in-pandas](https://stackoverflow.com/questions/19798153/difference-between-map-applymap-and-apply-methods-in-pandas)
- <https://pythonforbiologists.com/when-to-use-aggregatefiltertransform-in-pandas>