МИНОБРНАУКИ РОССИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
Институт математики, механики и компьютерных наук им. И. И. Воровича
Кафедра математического моделирования

# «Проект 2 курс»

***Мелехов Андрей Петрович***

# Задание 4. Пакет matplotlib

**Ростов-на-Дону**

**2022**

# Задание

1. Прочитайте текст из User's Guide "Pyplot tutorial"
https://matplotlib.org/stable/tutorials/introductory/pyplot.html#sphx-glr-tutorials-introductory-pyplot-py
и выполните команды из него в Jupiter. В тексте ниже приведен он же (с переводом).
2. Выберите пункт из User's Guide https://matplotlib.org/stable/users/index.html. Изучите его и выполните команды из него.

# Введение в интерфейс pyplot

https://matplotlib.org/stable/tutorials/introductory/pyplot.html#sphx-glr-tutorials-introductory-pyplot-py

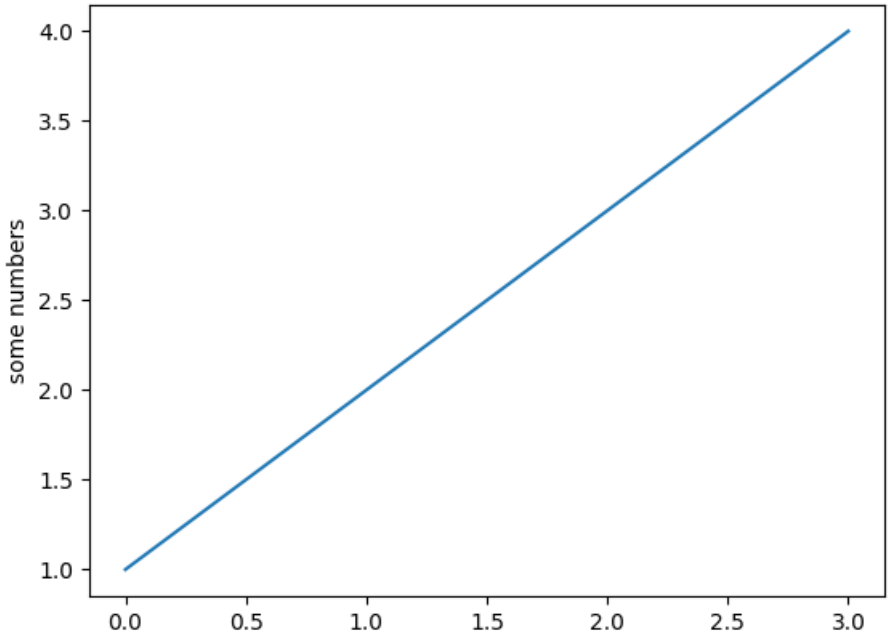
`matplotlib.pyplot` – это набор функций matplotlib, с графическими командами в стиле MATLAB. Каждая функция pyplot меняет какие-то параметры рисунка: например, создает фигуру (figure – окно для рисования), создает область рисования на фигуре, строит некоторые линии в области построения, добавляет метки и т. п.
В `matplotlib.pyplot` различные состояния сохраняются при вызовах функций, так что он отслеживает такие вещи, как текущая фигура и область построения.

**Замечание**

pyplot API, как правило, менее гибок, чем объектно-ориентированный API. Большинство вызовов функций, которые вы здесь видите, также могут быть вызваны как методы из объекта `Axes` (с такими же или похожими именами).

Вывод графика с помощью pyplot:

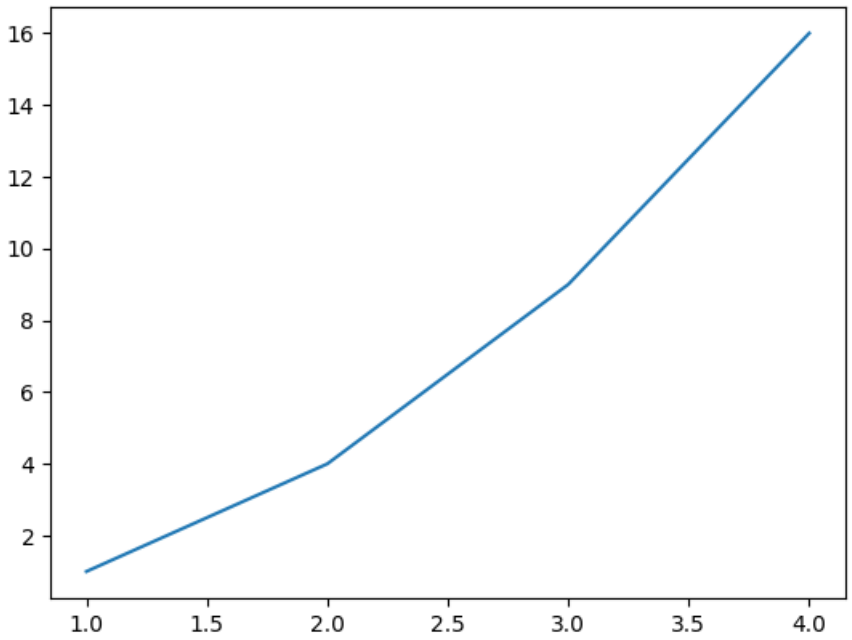
```
import matplotlib.pyplot as plt
plt.plot([1, 2, 3, 4])
plt.ylabel('some numbers')
plt.show()
```



Возможно, вам интересно, почему ось x изменяется от 0 до 3, а ось y от 1 до 4. Если вы передаете один список для построения, matplotlib предполагает, что это значения y, и автоматически генерирует значения x вида: [0, 1, 2, 3].

`plot` является функцией с большим количеством возможных параметров. Например, чтобы построить график (x, y), вы можете написать:

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



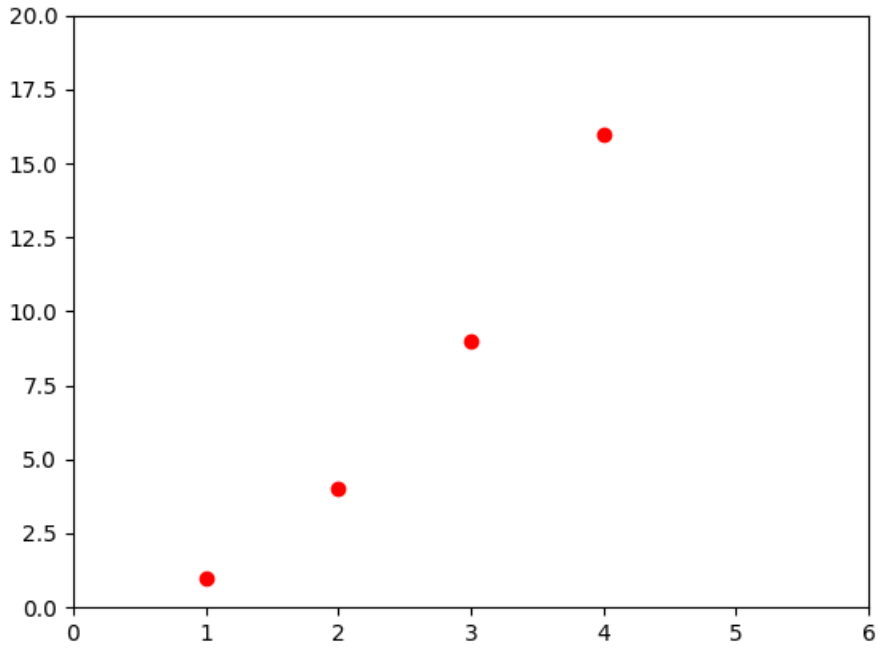

Out:

```
[<matplotlib.lines.Line2D object at 0x7f1c6bee4940>]
```

### Задание стилей функции plot

Для каждой пары аргументов x, y существует третий необязательный аргумент, строка формата, указывающая цвет и тип линий графика. Буквы и символы строки формата такие же как в MATLAB. Строка формата по умолчанию  равна "b-", что означает сплошную синюю линию. Например, чтобы задать красные кругами, надо написать ‘ro’ (r – красный; o - круг):

```
plt.plot([1, 2, 3, 4], [1, 4, 9, 16], 'ro')
plt.axis([0, 6, 0, 20])
plt.show()
```

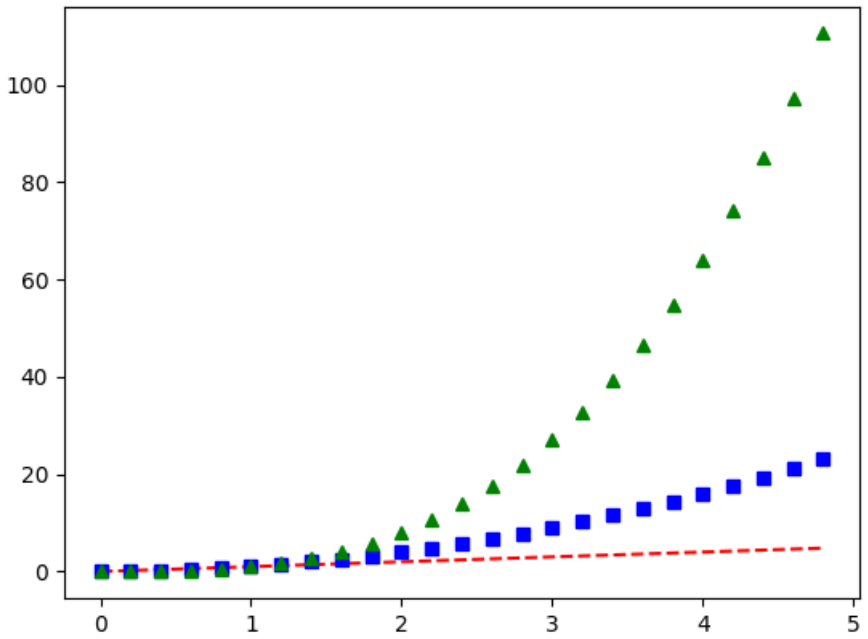



Полный список стилей линий и строк формата см. в документации по `plot`. Функция `axis` в приведенном выше примере принимает список [xmin, xmax, ymin, ymax] и определяет размеры по осям.

Если бы matplotlib ограничивался работой со списками, он был бы совершенно бесполезен для числовой обработки. Как правило, вы будете использовать массивы numpy. Фактически, все данные внутри преобразуются в массивы numpy. В примере ниже показано построение нескольких графиков с различными стилями в одном вызове функции plot с использованием массивов.

```
import numpy as np
# evenly sampled time at 200ms intervals
```

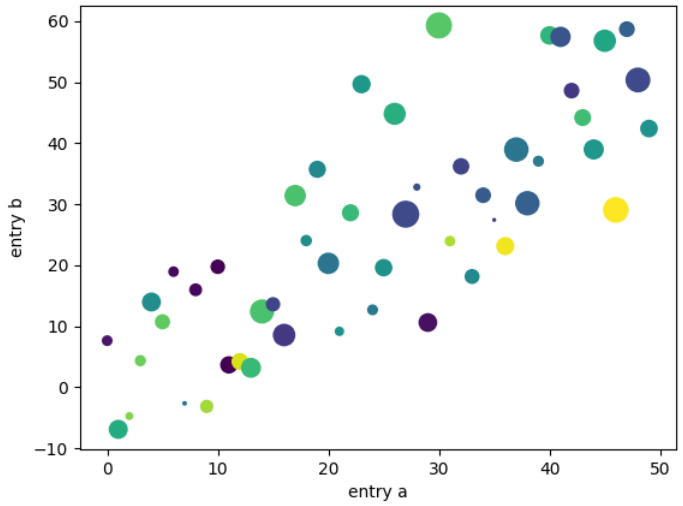
```
t = np.arange(0., 5., 0.2)

# red dashes, blue squares and green triangles
plt.plot(t, t, 'r--', t, t**2, 'bs', t, t**3, 'g^')
plt.show()
```




**Передача данных с помощью словаря с ключами строками**

Некоторые команды, которые имеют параметр data, позволяют передавать данные в виде словаря. Например, команды `numpy.recarray` или `pandas.DataFrame`.

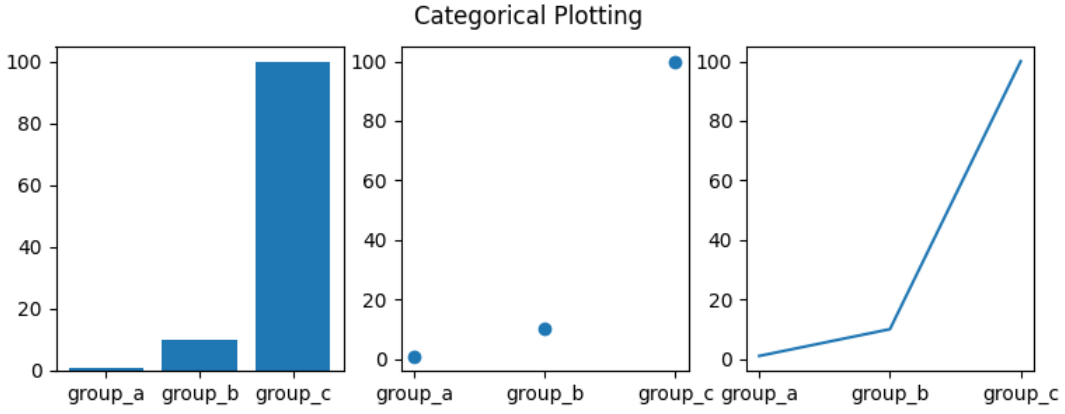
```
data = {'a': np.arange(50),
        'c': np.random.randint(0, 50, 50),
        'd': np.random.randn(50)}
data['b'] = data['a'] + 10 * np.random.randn(50)
data['d'] = np.abs(data['d']) * 100
plt.scatter('a', 'b', c='c', s='d', data=data)
plt.xlabel('entry a')
plt.ylabel('entry b')
plt.show()
```




## Построение графиков с категориальными переменными

Matplotlib позволяет передавать категориальные переменные непосредственно во многие функции построения графиков. Например:

```
names = ['group_a', 'group_b', 'group_c'] # по оси x – категории (не числа)
values = [1, 10, 100]
plt.figure(figsize=(9, 3)) # размеры окна

plt.subplot(131) # - задаем 3 окна и выбираем 1-е
plt.bar(names, values) # данные по осям x и y
plt.subplot(132) # 2-е окно
plt.scatter(names, values)
plt.subplot(133) № 3-е окно
plt.plot(names, values)
plt.suptitle('Categorical Plotting')
plt.show()
```




**Управление свойствами линий**

Линии имеют много атрибутов: linewidth (ширина линии), dash style (стиль), antialiased (сглаживание) и т.д.; См. `matplotlib.lines.Line2D`. Существует несколько способов задать свойства линий.

- Используя ключевые слова:

```
plt.plot(x, y, linewidth=2.0)
```

- Используя set-методы объектов-линий Line2D. Команда plot возвращает список таких объектов; Например, line1, line2 = plot(x1, y1, x2, y2).

```
line, = plt.plot(x, y, '-')
line.set_antialiased(False) # turn off antialiasing
```

- Используя метод `setp`. В приведенном ниже примере используется функция в стиле MATLAB для задания нескольких свойств. Можно использовать либо присваивания (параметр = значение), либо набор пар параметров: строка и значение.

```
lines = plt.plot(x1, y1, x2, y2)
# use keyword args
plt.setp(lines, color='r', linewidth=2.0)
# or MATLAB style string value pairs
plt.setp(lines, 'color', 'r', 'linewidth', 2.0)
```

## Список свойств линий `Line2D`.

| Свойство | Тип значений |
|---|---|
| alpha | float |
| animated | [True \| False] |
| antialiased or aa | [True \| False] |
| clip_box | a matplotlib.transform.Bbox instance |
| clip_on | [True \| False] |
| clip_path | a Path instance and a Transform instance, a Patch |
| color or c | any matplotlib color |
| contains | the hit testing function |
| dash_capstyle | `['butt' \| 'round' \| 'projecting']` |
| dash_joinstyle | `['miter' \| 'round' \| 'bevel']` |
| dashes | sequence of on/off ink in points |
| data | (np.array xdata, np.array ydata) |
| figure | a matplotlib.figure.Figure instance |
| label | any string |
| linestyle or ls | `[ '-' \| '--' \| '-.' \| ':' \| 'steps' \| ...]` |
| linewidth or lw | float value in points |
| marker | `[ '+' \| ',' \| '.' \| '1' \| '2' \| '3' \| '4' ]` |

| Свойство | Тип значений |
|---|---|
| markeredgecolor or mec | any matplotlib color |
| markeredgewidth or mew | float value in points |
| markerfacecolor or mfc | any matplotlib color |
| markersize or ms | float |
| markevery | [ None \| integer \| (startind, stride) ] |
| picker | used in interactive line selection |
| pickradius | the line pick selection radius |
| solid_capstyle | ['butt' \| 'round' \| 'projecting'] |
| solid_joinstyle | ['miter' \| 'round' \| 'bevel'] |
| transform | a matplotlib.transforms.Transform instance |
| visible | [True \| False] |
| xdata | np.array |
| ydata | np.array |
| zorder | any number |

Получить список всех свойств можно, вызвав функцию `setp` только с объектом-линией, без остальных параметров:

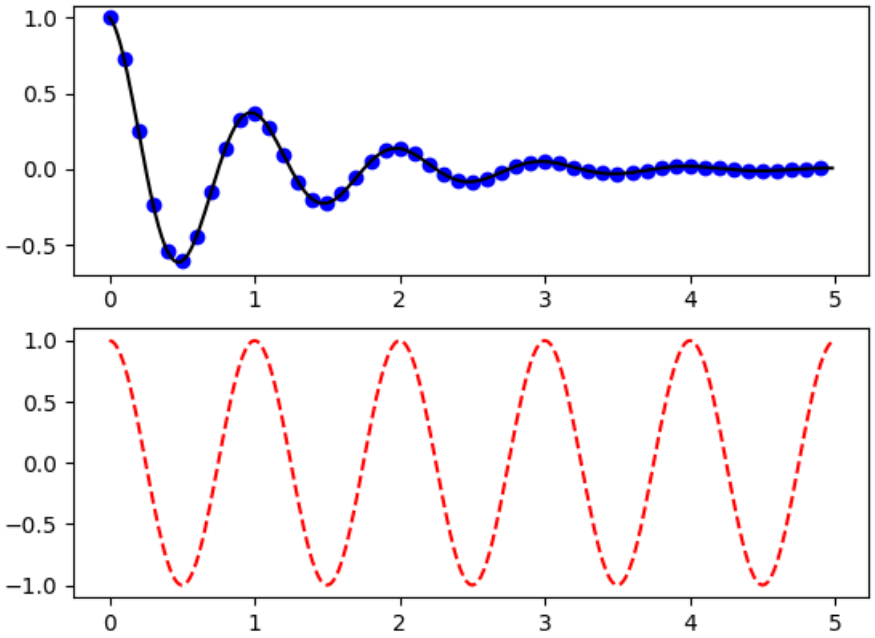
```
In [69]: lines = plt.plot([1, 2, 3])
In [70]: plt.setp(lines)
  alpha: float
```

```
  animated: [True | False]
  antialiased or aa: [True | False]
  ...snip
```

## Работа с несколькими фигурами и осями

В MATLAB, и `pyplot`, есть понятие текущей фигуры figure (окна) и текущих осей axes (в одном окне может несколько осей). Все операции применяются к текущем окну и осям. Функция `gca` возвращает ссылку на текущие оси (объект `matplotlib.axes.Axes`), а функция `gcf` возвращает ссылку на текущее окно (объект `matplotlib.figure.Figure`). Обычно не нужно беспокоиться об этом. Ниже приведен сценарий с двумя осями в одном окне.

```
def f(t):
    return np.exp(-t) * np.cos(2*np.pi*t)

t1 = np.arange(0.0, 5.0, 0.1)
t2 = np.arange(0.0, 5.0, 0.02)

plt.figure()
plt.subplot(211) # или так: plt.subplot(2, 1, 1)
plt.plot(t1, f(t1), 'bo', t2, f(t2), 'k')

plt.subplot(212)
plt.plot(t2, np.cos(2*np.pi*t2), 'r--')
plt.show()
```




Вызов команды `figure` необязателен, потому что фигура будет создана автоматически, если нет ее явного вызова. Также будут созданы оси axes (эквивалентна явному вызову subplot()), если нет их явного вызова.

Команде `subplot` при вызове передаются параметры `numrows, numcols, plot_number`, где `plot_number` изменяется от 1 до `numrows*numcols`. Запятые между параметрами можно не ставить, если numrows*numcols<10. Вызов `subplot`(211) идентичен `subplot`(2, 1, 1).

Вы можете создать произвольное количество подокон subplots и осей axes. Если вы хотите размещать оси вручную, т. е. не на прямоугольной сетке, используйте команду `axes`, которая позволяет указать местоположение в виде `axes([left, bottom, width, height])`, где все зна-

чения задаются в виде дробей (от 0 до 1) – относительные координаты размещения окна. См. Axes Demo примеры размещения осей вручную и Basic Subplot Demo примеры с большим количеством subplots.

Вы можете создать несколько `figure` вызвав их с разными номерами. Каждая figure может содержать несколько axes и subplots:

```
import matplotlib.pyplot as plt
plt.figure(1)                # the first figure
plt.subplot(211)             # the first subplot in the first figure
plt.plot([1, 2, 3])
plt.subplot(212)             # the second subplot in the first figure
plt.plot([4, 5, 6])


plt.figure(2)                # a second figure
plt.plot([4, 5, 6])          # creates a subplot() by default

plt.figure(1)                # figure 1 current; subplot(212) still current
plt.subplot(211)             # make subplot(211) in figure1 current
plt.title('Easy as 1, 2, 3') # subplot 211 title
```

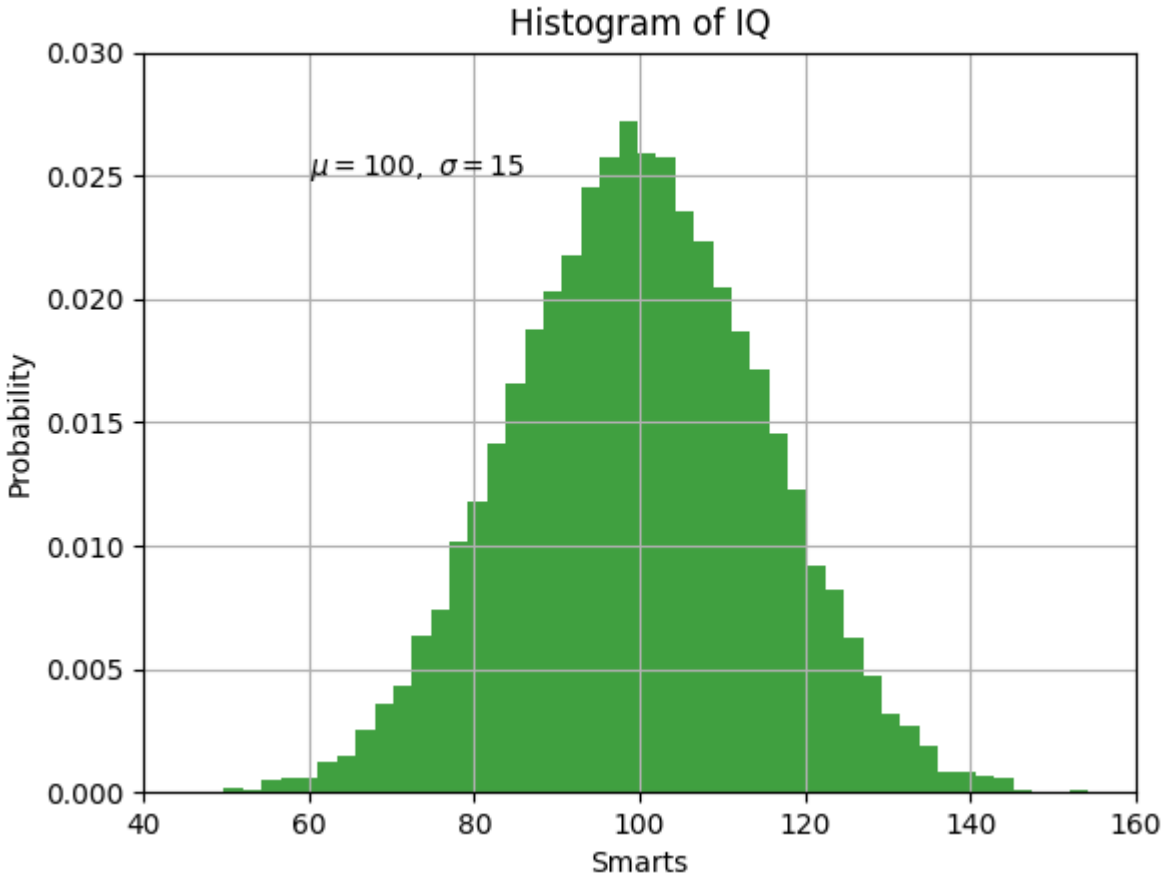
Очистить текущую фигуру можно с помощью команды clf, а текущие оси с помощью cla. Использование текущих значений (в частности, текущее изображение, фигура и оси) – это всего лишь тонкая оболочка с отслеживанием состояния вокруг объектно-ориентированного API. Если это неудобно, можно самостоятельно их задавать вместо этого (см. Artist tutorial).

Если вы делаете много рисунков, вам нужно знать еще одну вещь: память, необходимая для фигуры, не освобождается полностью до тех пор, пока фигура не будет явно закрыта с помощью `close`. Удаление всех ссылок на фигуру и/или использование оконного менеджера для уничтожения окна, в котором фигура появляется на экране, недостаточно, поскольку pyplot поддерживает внутренние ссылки до тех пор, пока не будет вызвано закрытие.

## Работа с текстом

Команду `text` можно использовать для добавления текста в произвольное место рисунка, а `xlabel`, `ylabel` и `title` используются для добавления текста в определенные места (более подробный пример см. в разделе Text in Matplotlib Plots).

```
mu, sigma = 100, 15
x = mu + sigma * np.random.randn(10000)

# the histogram of the data
n, bins, patches = plt.hist(x, 50, density=1, facecolor='g', alpha=0.75)

plt.xlabel('Smarts')
plt.ylabel('Probability')
plt.title('Histogram of IQ')
plt.text(60, .025, r'$\mu=100,\ \sigma=15$')
plt.axis([40, 160, 0, 0.03])
plt.grid(True)
plt.show()
```




Все функции `text` возвращают объекты `matplotlib.text.Text`. Также как и со свойствами линий, Вы можете задать свойства с использованием ключевых слов внутри текстовой функции или используя функцию `setp`:

```
t = plt.xlabel('my data', fontsize=14, color='red')
```

Эти свойства более подробно описаны в Text properties and layout.

### Использование математических выражений в тексте

matplotlib поддерживает TeX. Например, для записи выражения $\sigma_i=15$ в заголовке, надо написать TeX-выражение внутри знаков доллара:

```
plt.title(r'$\sigma_i=15$')
```

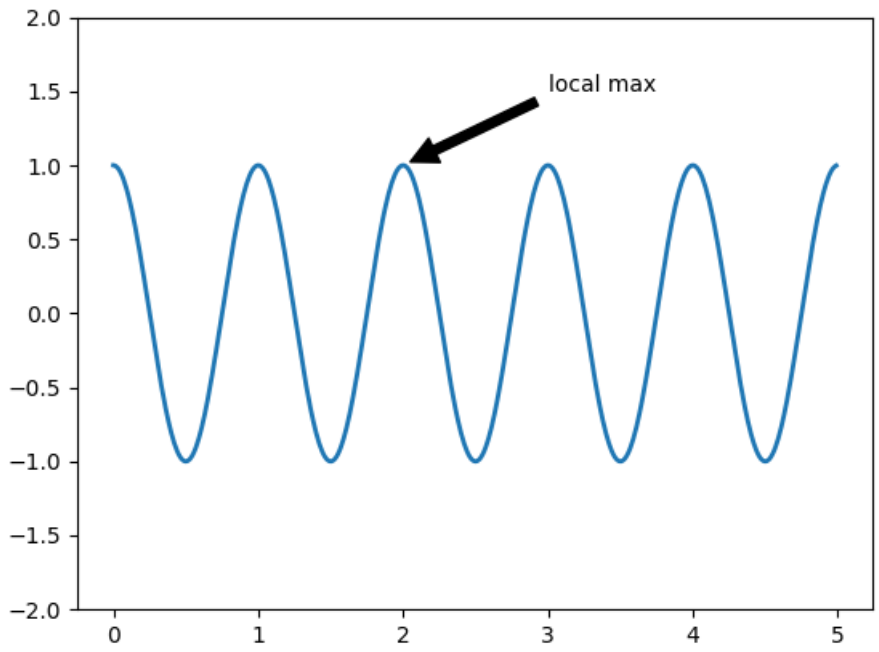
Знак r, предшествующий строке, важен – это означает, что это сырая строка и не должна обратные косые черты как экранирование python. matplotlib имеет встроенный синтаксический анализатор выражений TeX и механизм компоновки, а также поставляет свои собственные математические шрифты-подробности см. в разделе see Writing mathematical expressions. Таким образом, вы можете использовать математический текст на разных платформах, без установки TeX. Для тех, у кого установлены LaTeX и dvipng, вы также можете использовать LaTeX для форматирования текста и включения выходных данных непосредственно в отображаемые рисунки или сохраненный postscript-см. раздел Text rendering With LaTeX.

### Аннотации (подписи)

Функция `text` помещает текст в произвольное место. Обычно текст используется для подписи к какой-либо части рисунка, а метод `annotate` обеспечивает вспомогательную функциональность. В `annotate` задаются 2 точки: расположение аннотируемого объекта xy и расположение текста xytext. Оба этих аргумента задаются кортежами вида (x, y).

```
ax = plt.subplot()

t = np.arange(0.0, 5.0, 0.01)
s = np.cos(2*np.pi*t)
```

```
line, = plt.plot(t, s, lw=2)

plt.annotate('local max', xy=(2, 1), xytext=(3, 1.5),
             arrowprops=dict(facecolor='black', shrink=0.05)  )

plt.ylim(-2, 2)
plt.show()
```



Более подробную информацию см. Basic annotation и Advanced Annotations. Много примеров можно найти в Annotating Plots.

## Логарифмические и нелинейные оси координат

`matplotlib.pyplot` поддерживает не только линейные оси, но также логарифмические и logit оси. Обычно они используются, если данные различаются на несколько порядков. Изменение осей просто:

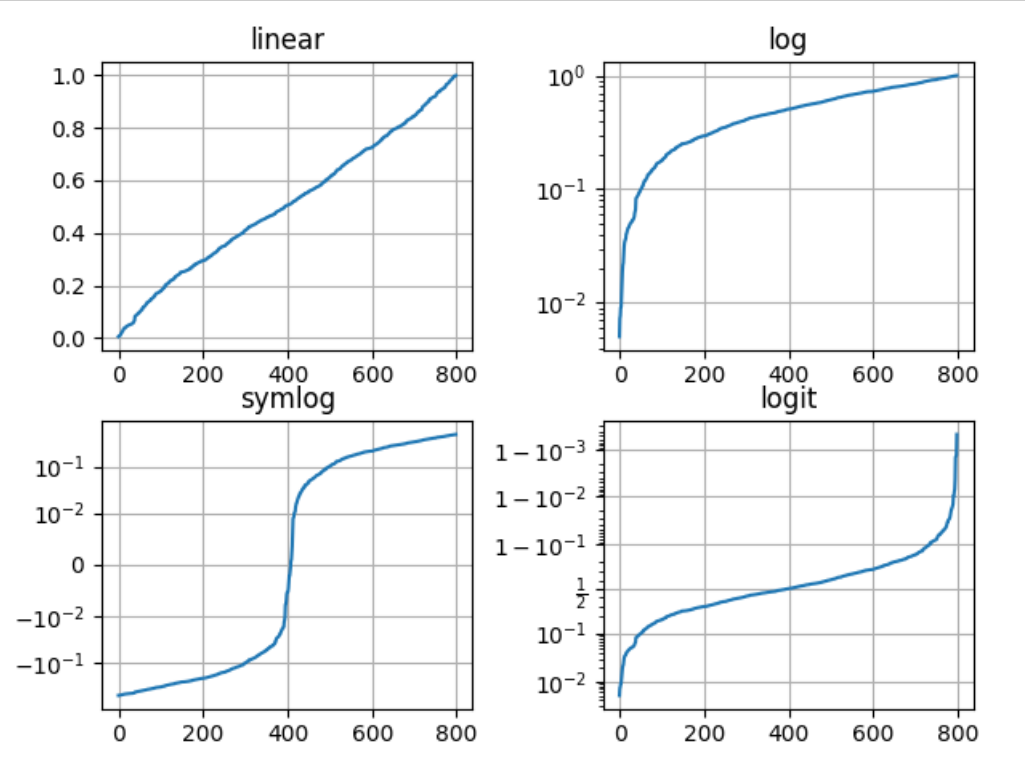
plt.xscale('log')

Ниже приведен пример четырех графиков с одинаковыми данными и разными масштабами.

```
# Fixing random state for reproducibility
np.random.seed(19680801)
# make up some data in the open interval (0, 1)
y = np.random.normal(loc=0.5, scale=0.4, size=1000)
y = y[(y > 0) & (y < 1)]
y.sort()
x = np.arange(len(y))

# plot with various axes scales
plt.figure()

# linear
plt.subplot(221)
plt.plot(x, y)
plt.yscale('linear')
plt.title('linear')
```

```
plt.grid(True)

# log
plt.subplot(222)
plt.plot(x, y)
plt.yscale('log')
plt.title('log')
plt.grid(True)

# symmetric log
plt.subplot(223)
plt.plot(x, y - y.mean())
plt.yscale('symlog', linthresh=0.01)
plt.title('symlog')
plt.grid(True)

# logit
plt.subplot(224)
plt.plot(x, y)
plt.yscale('logit')
plt.title('logit')
plt.grid(True)
# Adjust the subplot layout, because the logit one may take more space
# than usual, due to y-tick labels like "1 - 10^{-3}"
plt.subplots_adjust(top=0.92, bottom=0.08, left=0.10, right=0.95, hspace=0.25,
```

```
                    wspace=0.35)
```




Можно добавить собственную шкалу, подробнее см. Developer's guide for creating scales and transformations.

## Визуализация данных

https://pandas.pydata.org/pandas-docs/stable/user_guide/visualization.html

Используем стандартное соглашение для подключения matplotlib API:

```
In [1]: import matplotlib.pyplot as plt
        import numpy as np
        import pandas as pd
In [2]: plt.close("all")
```

Мы демонстрируем базовые возможности pandas создание графиков. См. раздел ecosystem – обзор дополнительных библиотек визуализации, которые выходят за рамки описанных здесь основ.

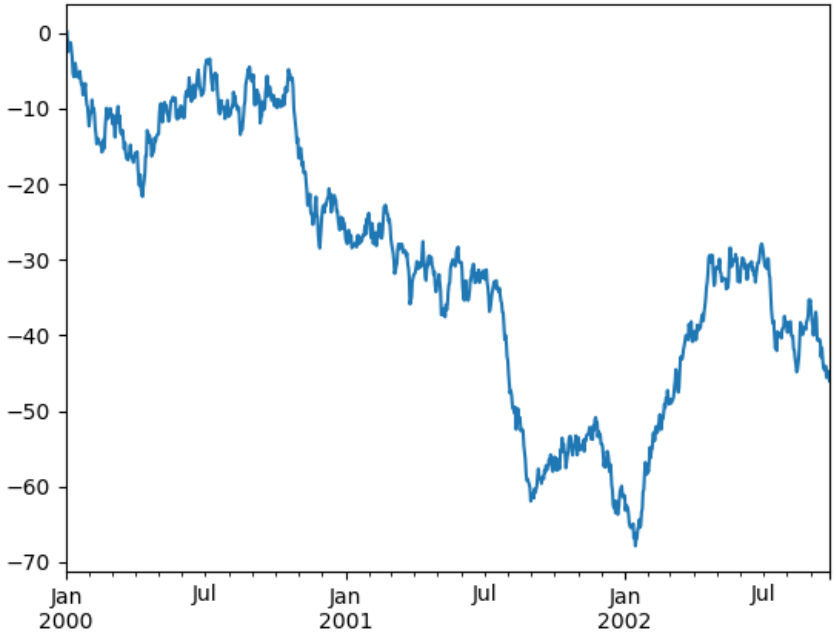
**Замечание.** Во всех вызовах `np.random` используется инициализация генератора с номером 123456.

### Основы построения графиков: команда `plot`

Мы продемонстрируем основы, более подробную информацию см. cookbook.

Команда `plot` объектов Series и DataFrame – это просто оболочка вокруг команды **`plt.plot()`**:

```
In [3]: ts = pd.Series(np.random.randn(1000), index =
                                pd.date_range("1/1/2000", periods=1000))
In [4]: ts = ts.cumsum()

In [5]: ts.plot();
```

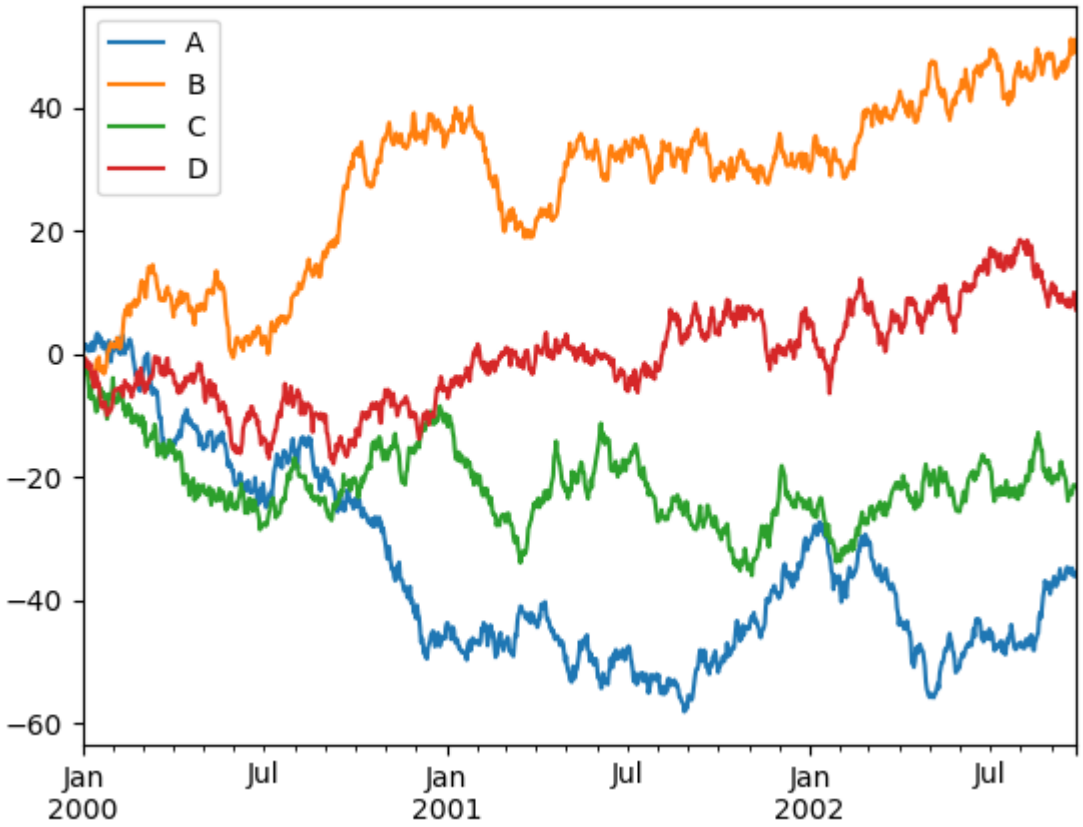



Если индекс содержит даты, он вызывает **`gcf().autofmt_xdate()`** t чтобы попытаться правильно отформатировать ось x, как указано выше.

В DataFrame, функция **`plot()`** удобна для построения всех столбцов с метками:

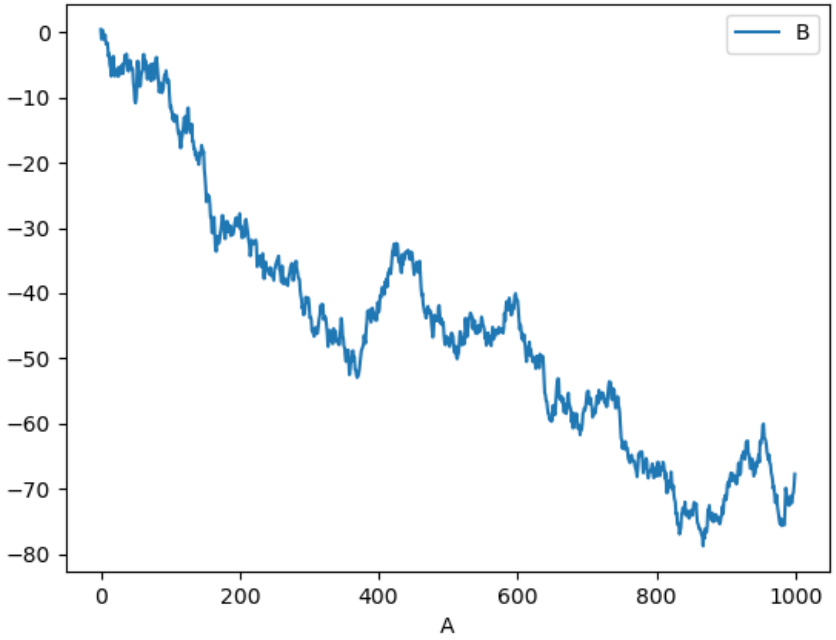
```
In [6]: df = pd.DataFrame(np.random.randn(1000, 4), index=ts.index, col-
umns=list("ABCD"))

In [7]: df = df.cumsum()

In [8]: plt.figure();
```

```
In [9]: df.plot();
```



Вы можете нарисовать зависимость одного столбца от другого, используя ключевые слова x и y в **`plot()`**:

```
In [10]: df3 = pd.DataFrame(np.random.randn(1000, 2),
                            columns=["B", "C"]).cumsum()
In [11]: df3["A"] = pd.Series(list(range(len(df))))
In [12]: df3.plot(x="A", y="B");
```




**Замечание.** Дополнительные параметры форматирования и стиля см. в разделе formatting ниже.

**Другие графики**

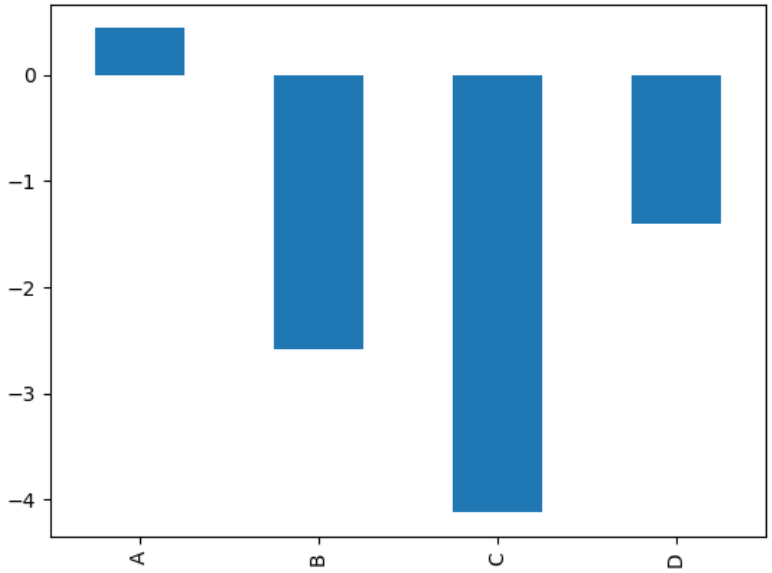
Plotting methods allow for a handful of plot styles other than the default line plot. These methods can be provided as the `kind` keyword argument to **`plot()`**, and include:

Следующие методы построения графиков позволяют использовать стили отличные от линейного графика. Эти методы могут быть заданы в качестве аргумента ключевого слова kind для функции **`plot()`**:

- 'bar' или 'barh' прямоугольники
- 'hist' гистограмма
- 'box' for boxplot
- 'kde' or 'density' for density plots
- 'area' график с заполнением цветом
- 'scatter' точечная диаграмма
- 'hexbin' for hexagonal bin plots
- 'pie' for pie plots

Например, гистограмму (тип bar) можно создать следующим образом:

```
In [13]: plt.figure();

In [14]: df.iloc[5].plot(kind="bar");
```




Вы можете также создать другие графики, используя `DataFrame.plot.<kind>` вместо задания `kind` аргумента.

```
In [15]: df = pd.DataFrame()

In [16]: df.plot.<TAB>  # noqa: E225, E999
df.plot.area     df.plot.barh     df.plot.density  df.plot.hist
df.plot.line     df.plot.scatter
df.plot.bar      df.plot.box      df.plot.hexbin   df.plot.kde
df.plot.pie
```

В дополнение к этим методам, существуют еще методы DataFrame.hist(), и DataFrame.boxplot(), которые используют другой интерфейс.

Еще есть несколько методом построения графиков plotting functions в `pandas.plotting`, которые принимают объекты **`Series`** или **`DataFrame`** в качестве аргумента. К ним относятся:

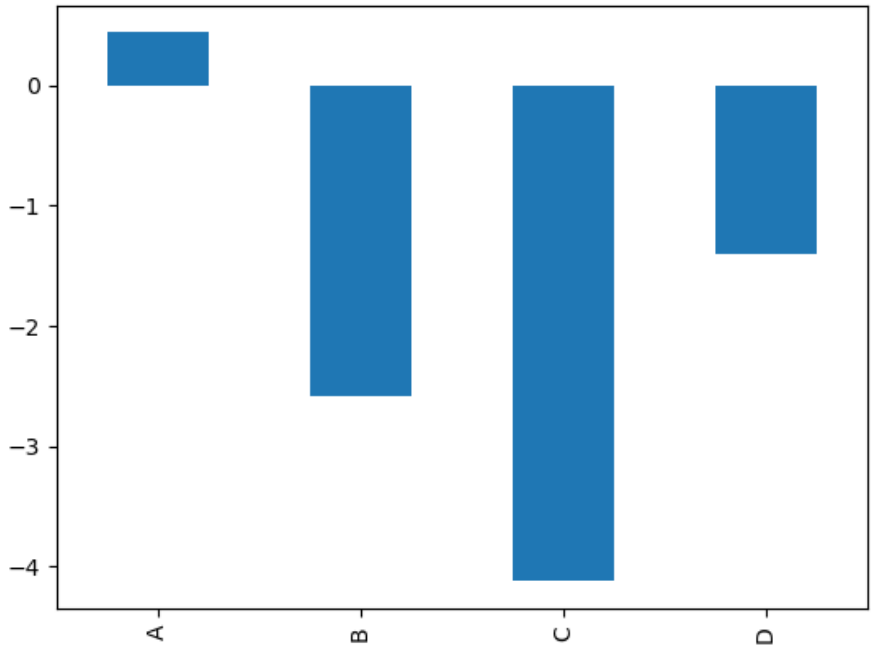
- Scatter Matrix
- Andrews Curves
- Parallel Coordinates
- Lag Plot
- Autocorrelation Plot
- Bootstrap Plot
- RadViz

В графики также могут быть добавлены errorbars или tables.

## Bar plots

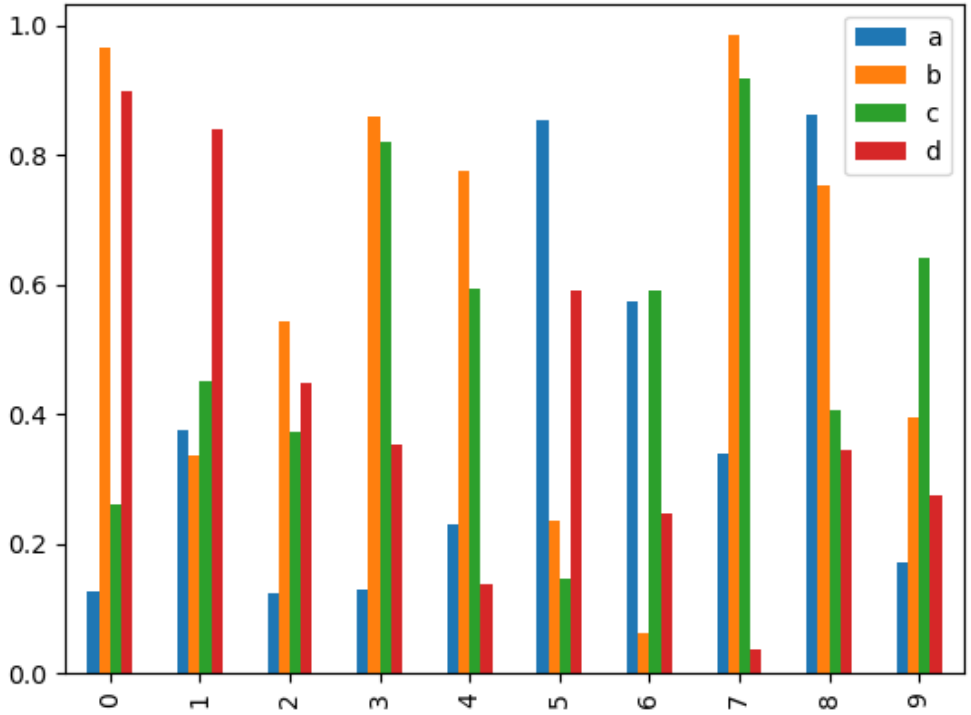
Для помеченных данных, не относящихся к временным рядам, вы можете создать гистограмму:

```
In [17]: plt.figure();

In [18]: df.iloc[5].plot.bar();

In [19]: plt.axhline(0, color="k");
```



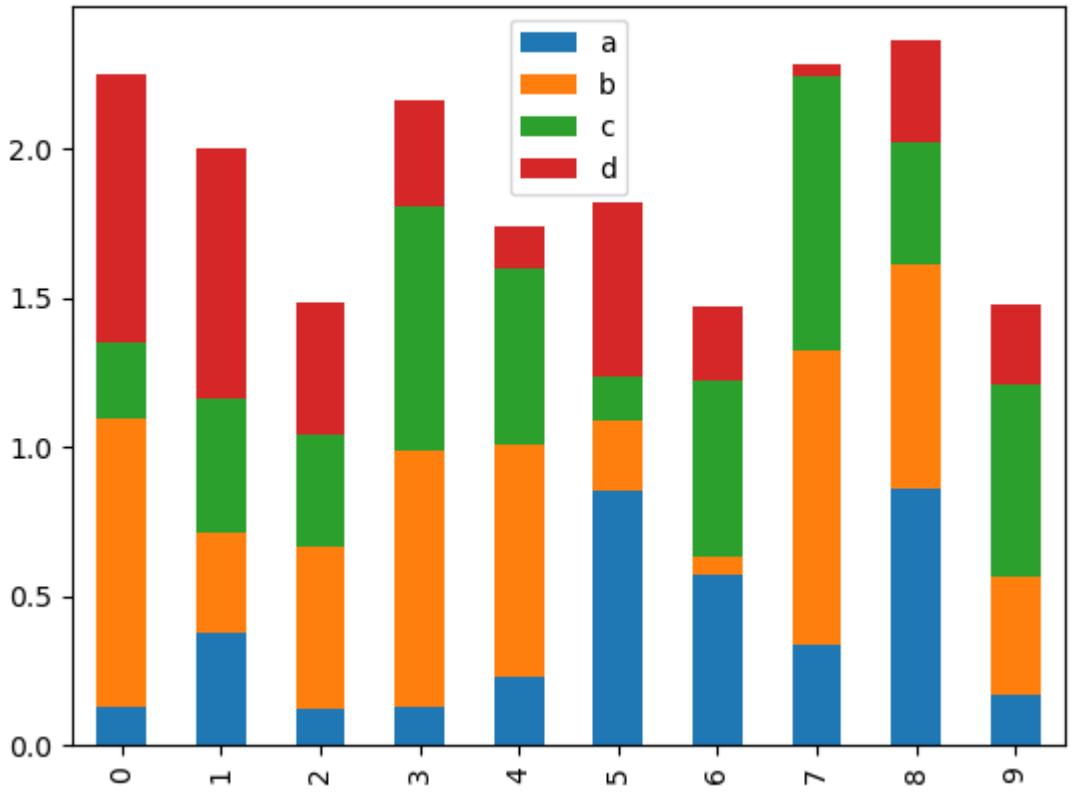
Вызов метода DataFrame's **`plot.bar()`** создает несколько bar-графиков:

```
In [20]: df2 = pd.DataFrame(np.random.rand(10, 4), columns=["a", "b", "c",
"d"])
In [21]: df2.plot.bar();
```



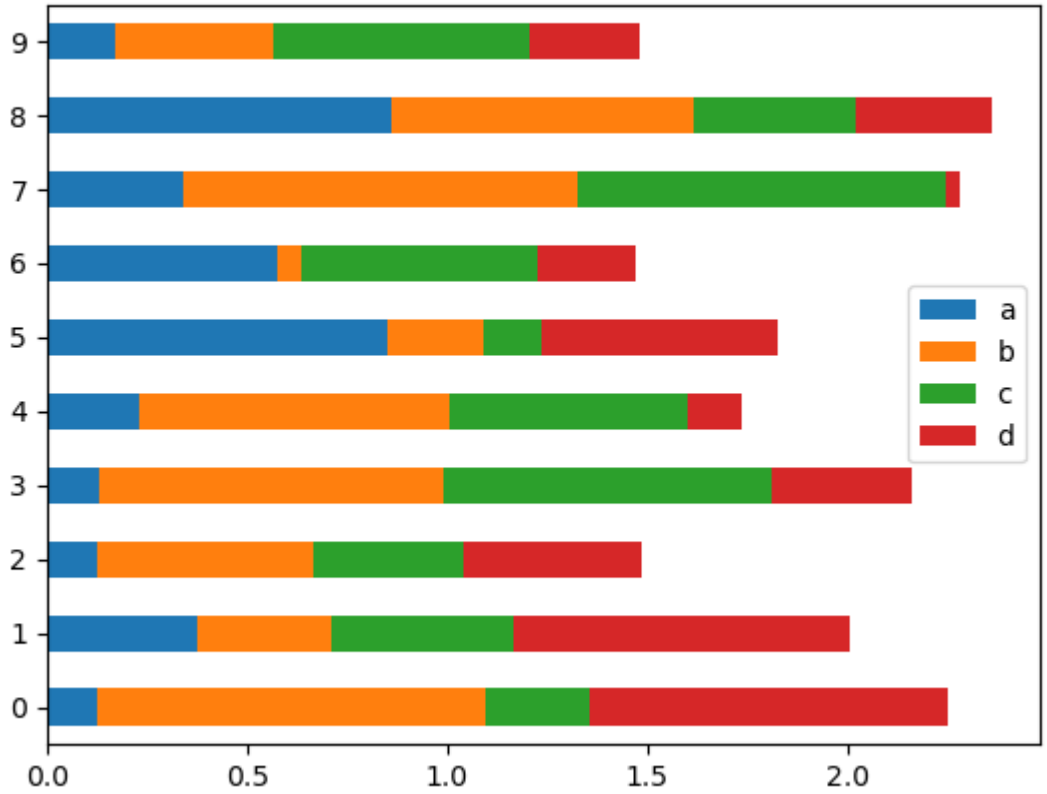
Чтобы создать столбчатую диаграмму, задайте `stacked=True`:

```
In [22]: df2.plot.bar(stacked=True);
```



Чтобы получить горизонтальные полосы, используйте метод barh:

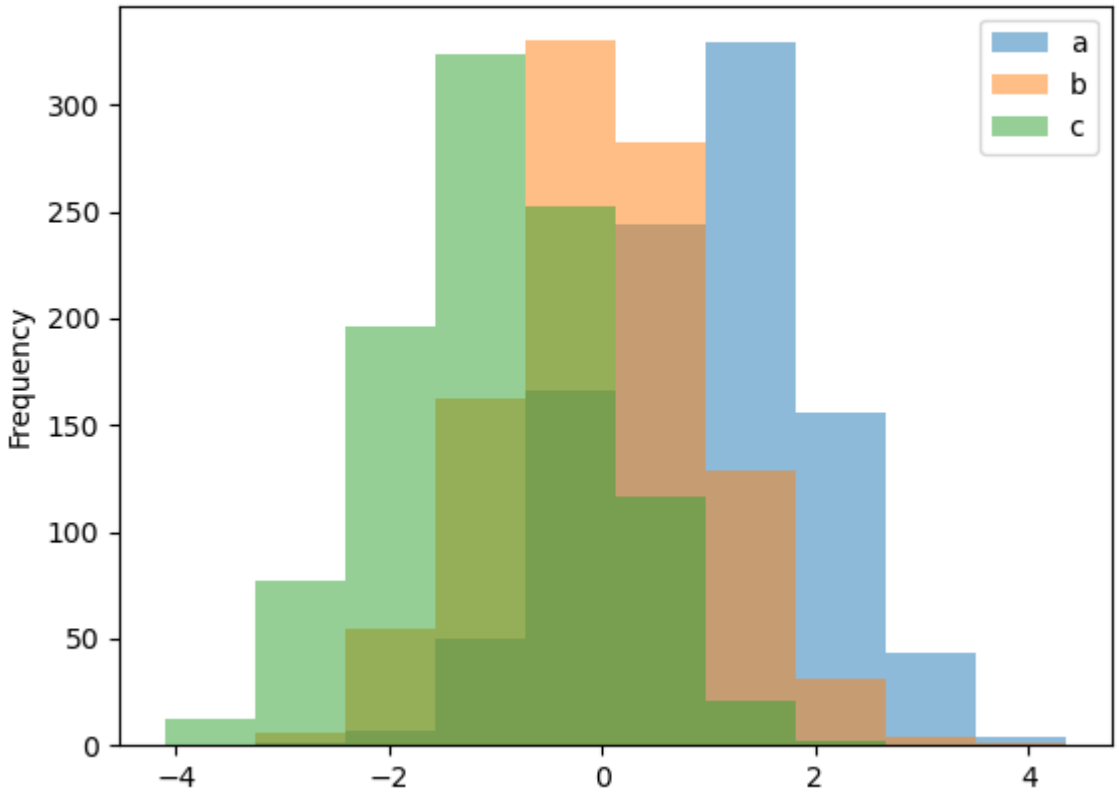
```
In [23]: df2.plot.barh(stacked=True);
```



### Гистограммы (Histograms)

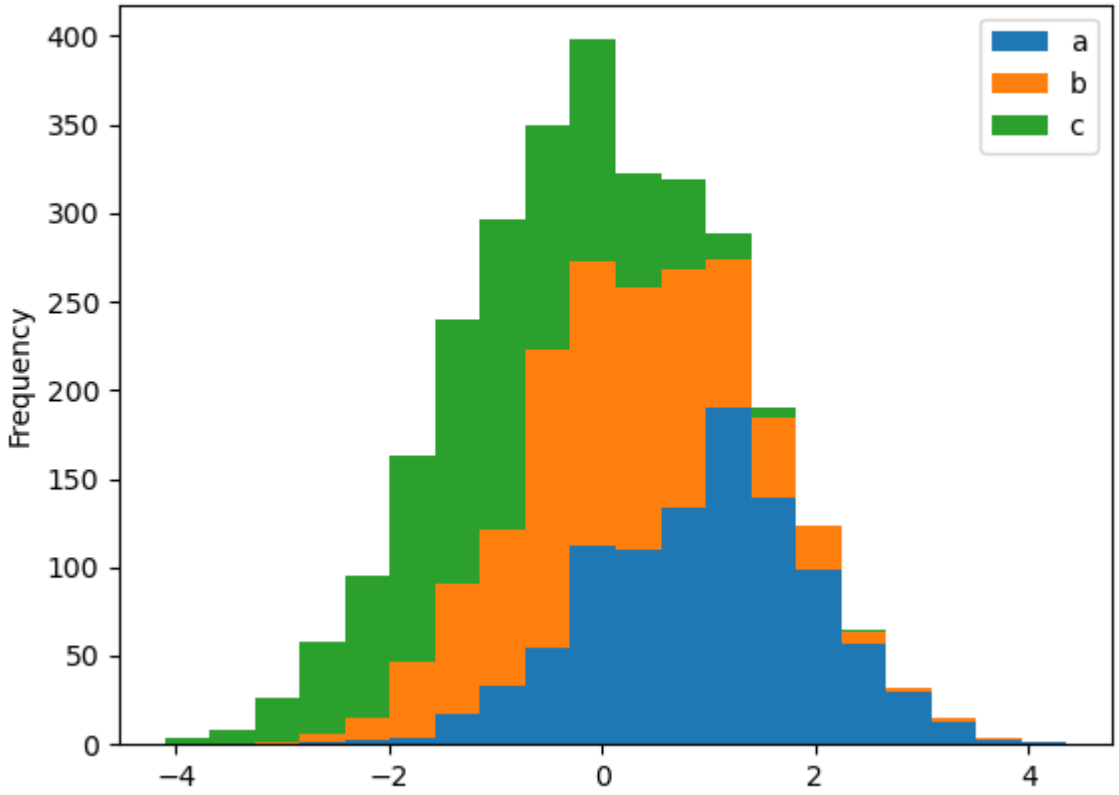
Гистограммы можно нарисовать с помощью методов **`DataFrame.plot.hist()`** и **`Series.plot.hist()`**.

```
In [24]: df4 = pd.DataFrame(
   ....:     {
   ....:         "a": np.random.randn(1000) + 1,
   ....:         "b": np.random.randn(1000),
   ....:         "c": np.random.randn(1000) - 1,
   ....:     },
   ....:     columns=["a", "b", "c"],
   ....: )

In [25]: plt.figure();

In [26]: df4.plot.hist(alpha=0.5);
```


a
b
c
Frequency
300
250
200
150
100
50
0
−4
−2
0
2
4

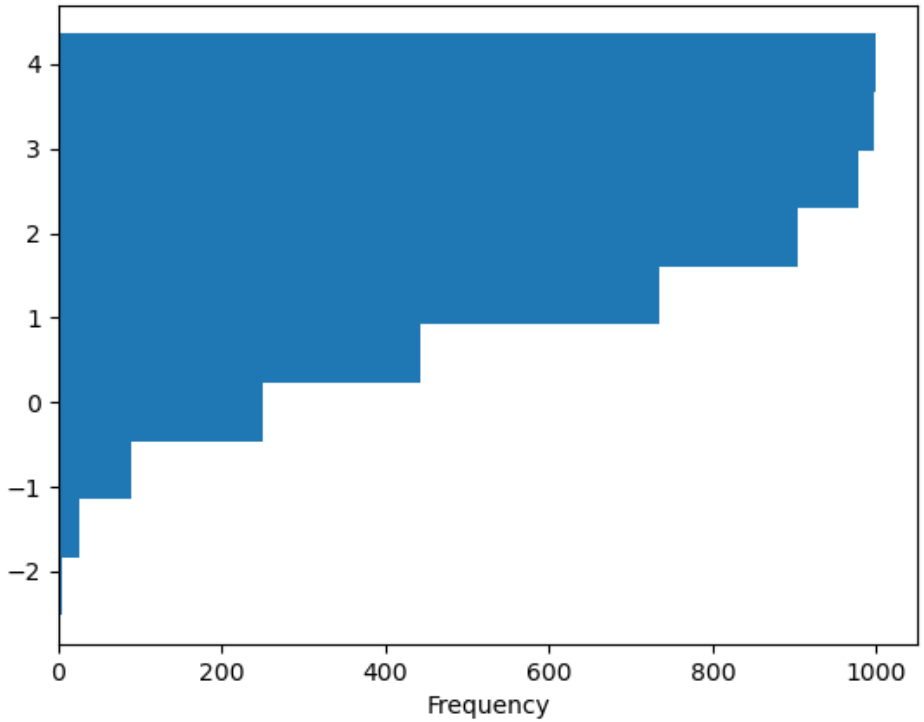
Гистограмму можно сложить в стопку, используя `stacked=True`. Размер ячейки можно изменить с помощью ключевого слова `bins`.

```
In [27]: plt.figure();

In [28]: df4.plot.hist(stacked=True, bins=20);
```



Вы можете задать другие данные, поддерживаемые matplotlib `hist`. Например, горизонтальные и кумулятивные гистограммы могут быть нарисованы с помощью `orientation='horizontal'` and `cumulative=True`.

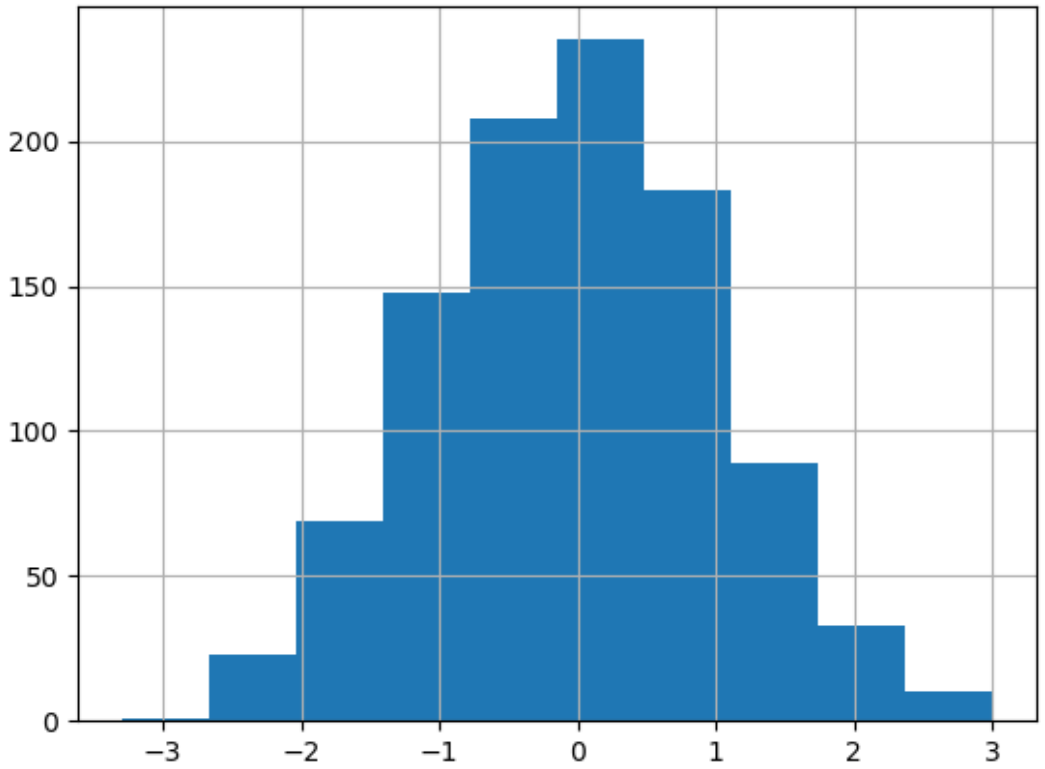
```
In [29]: plt.figure();

In [30]: df4["a"].plot.hist(orientation="horizontal", cumulative=True);
```



Подробнее см. метод **`hist`** и [matplotlib hist documentation](#).

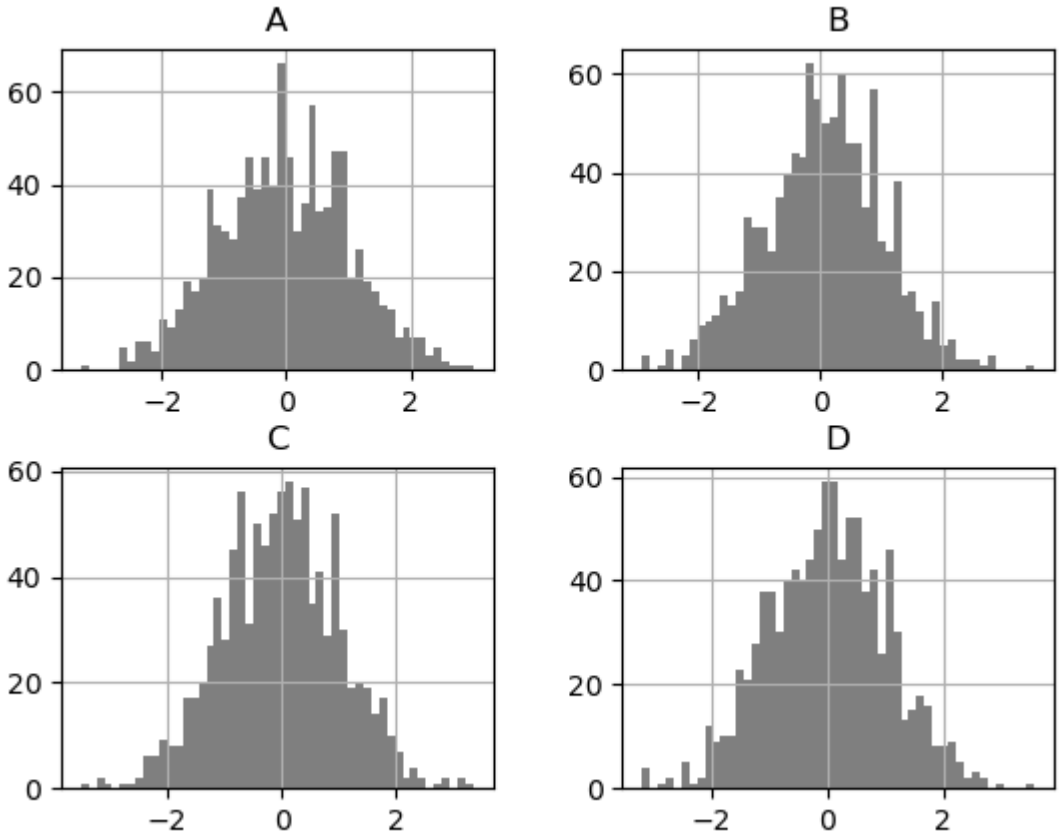
Можно еще использовать для построения гистограмм интерфейс `DataFrame.hist`.

```
In [31]: plt.figure();

In [32]: df["A"].diff().hist();
```



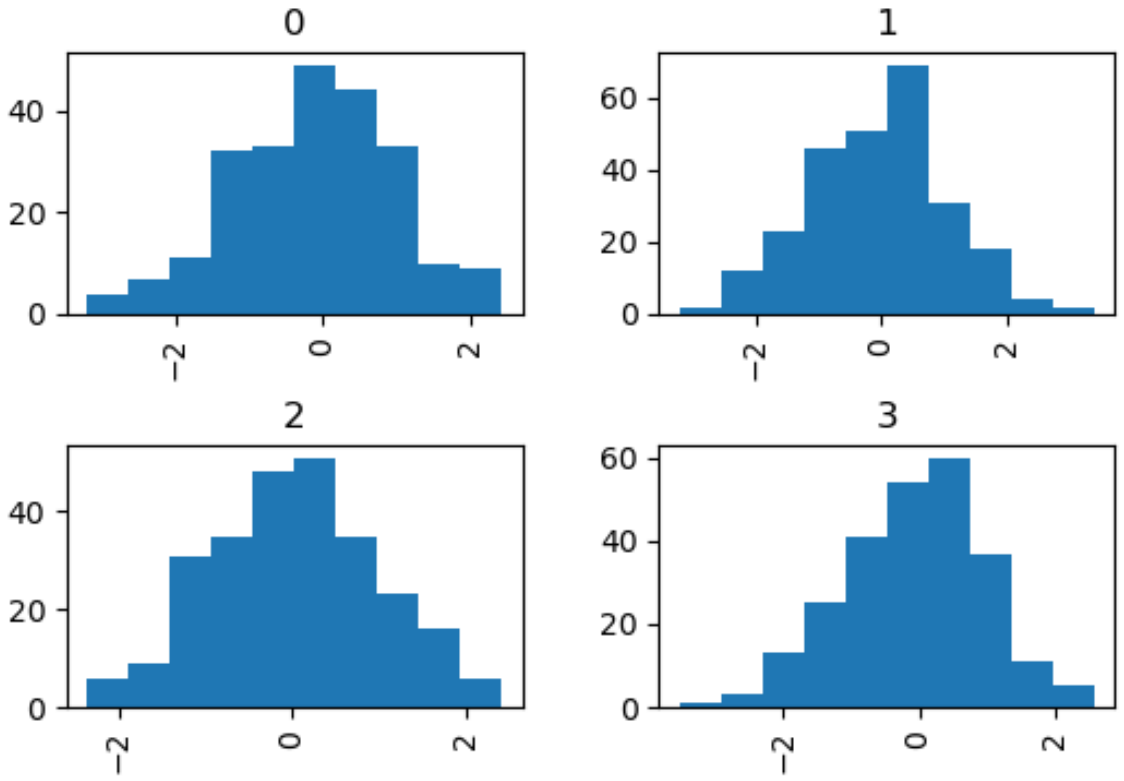
**`DataFrame.hist()`** выводит гистограммы столбцов в отдельные окна subplots:

```
In [33]: plt.figure();

In [34]: df.diff().hist(color="k", alpha=0.5, bins=50);
```



С помощью by можно задать построение группы гистограмм:

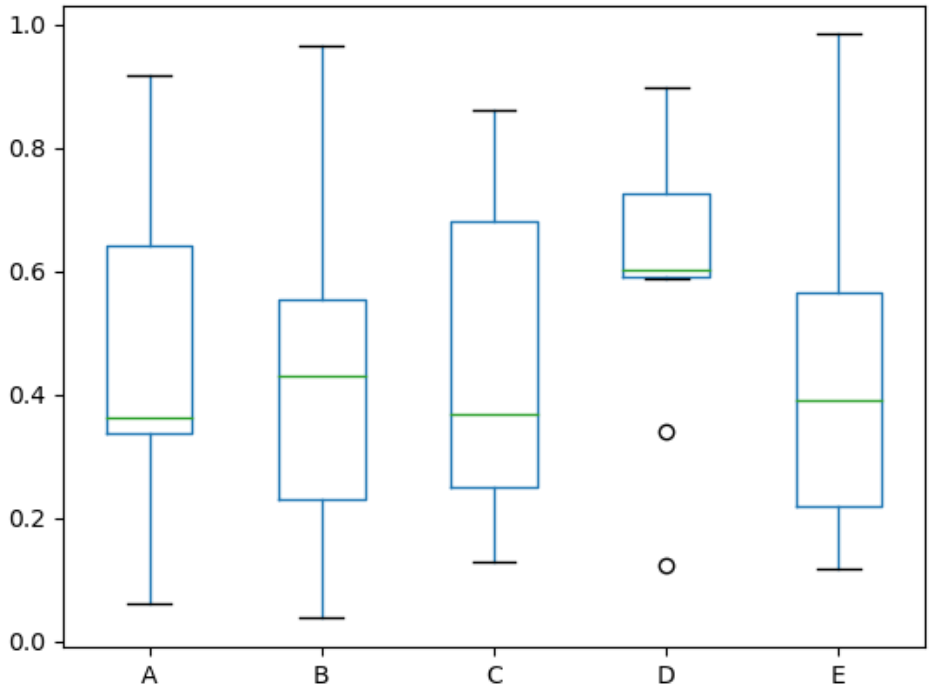
```
In [35]: data = pd.Series(np.random.randn(1000))

In [36]: data.hist(by=np.random.randint(0, 4, 1000), figsize=(6, 4));
```



## Box plots

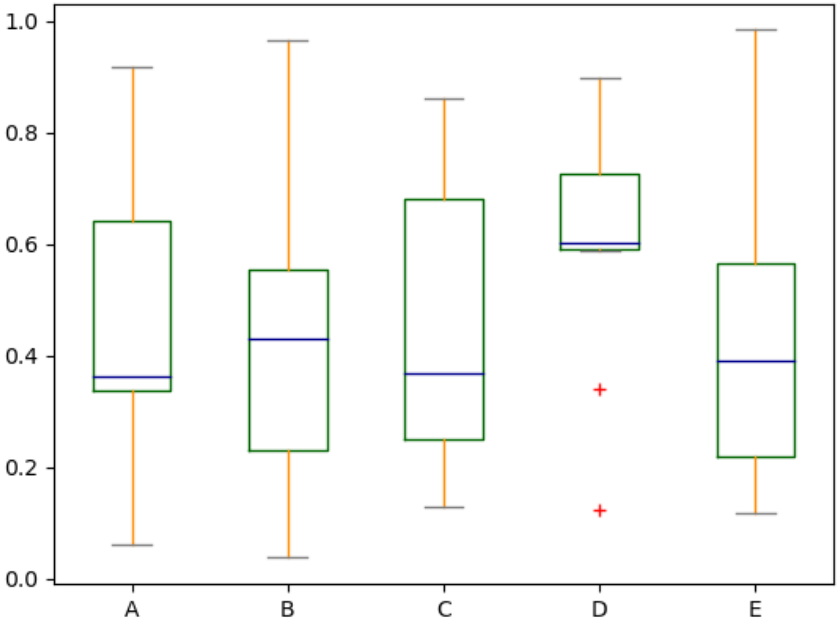
Могут быть построены с помощью **`Series.plot.box()`** и **`DataFrame.plot.box()`**, или **`DataFrame.boxplot()`** для визуализации распределения значений в каждом столбце. Например, график пяти испытаний из 10 наблюдений однородной случайной величины на [0,1).

```
In [37]: df = pd.DataFrame(np.random.rand(10, 5), columns=["A", "B", "C",
"D", "E"])
In [38]: df.plot.box();
```



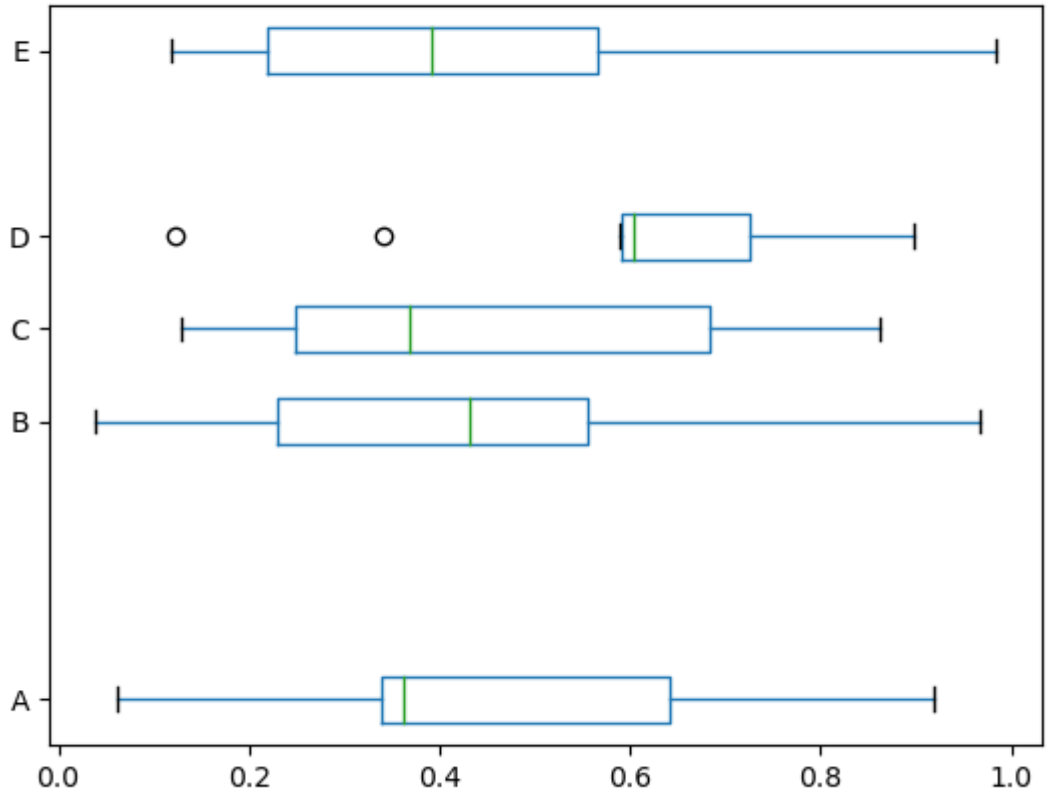
В Boxplot можно изменить оформление, задав color. Можно передать словарь, ключами которого являются `boxes`, `whiskers`, `medians` and `caps`. Если в словаре отсутствуют некоторые ключи, для них используются цвета по умолчанию. Кроме того, boxplot имеет параметр `sym` для указания стиля отдельных точек.

```
In [39]: color = { "boxes": "DarkGreen",
    ....:         "whiskers": "DarkOrange",
    ....:          "medians": "DarkBlue",
    ....:             "caps": "Gray",}
In [40]: df.plot.box(color=color, sym="r+");
```



Кроме того, вы можете передать другие ключевые слова, поддерживаемые matplotlib `boxplot`. Например, можно нарисовать горизонтальную и произвольно расположенную прямоугольную диаграмму с помощью `vert=False` и `positions`.

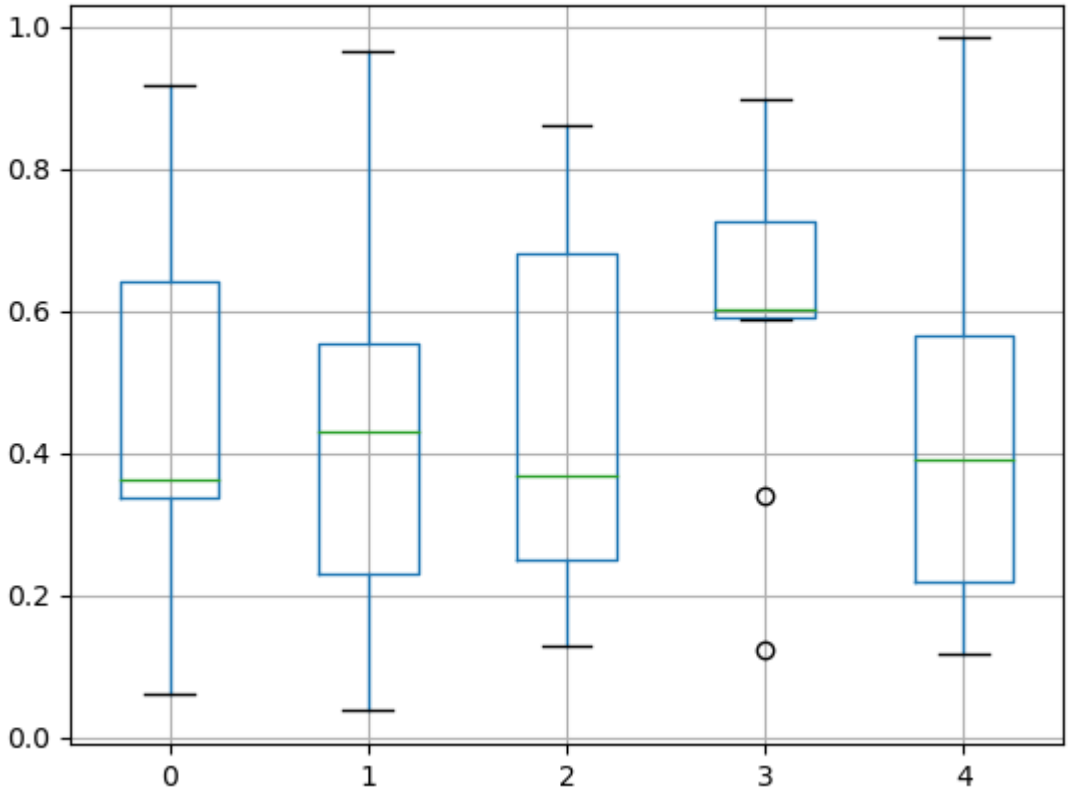
```
In [41]: df.plot.box(vert=False, positions=[1, 4, 5, 6, 8]);
```



Подробней см. метод **`boxplot`** и matplotlib boxplot documentation.

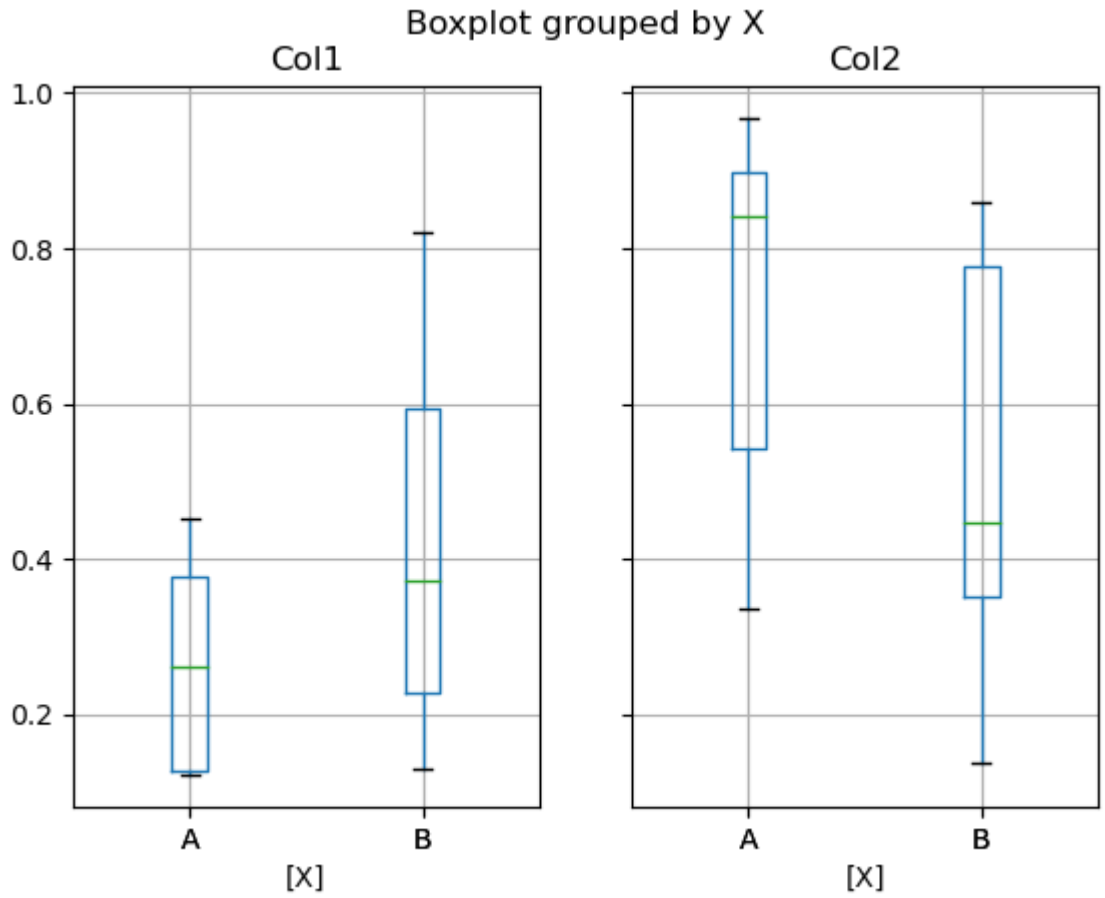
Также можно использовать `DataFrame.boxplot`.

```
In [42]: df = pd.DataFrame(np.random.rand(10, 5))
In [43]: plt.figure();
In [44]: bp = df.boxplot()
```



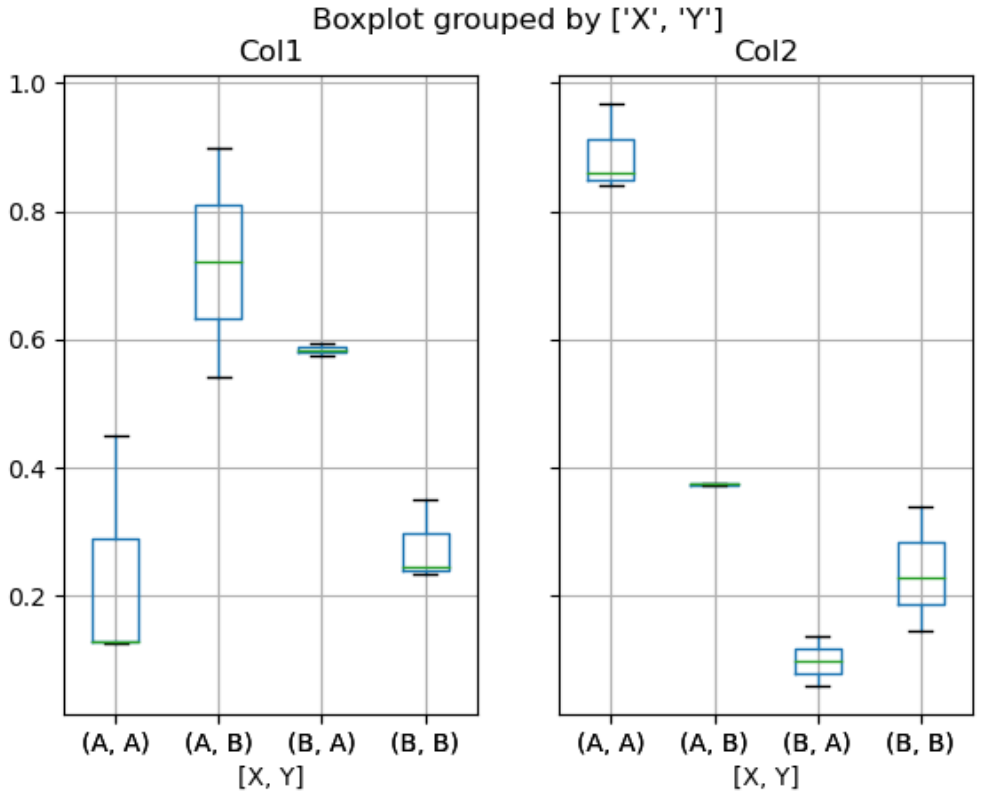
Можно создать стратифицированный boxplot, используя by для группировки. Например,

```
In [45]: df = pd.DataFrame(np.random.rand(10, 2), columns=["Col1", "Col2"])
In [46]: df["X"] = pd.Series(["A","A","A","A","A", "B","B","B","B","B"])
In [47]: plt.figure();
In [48]: bp = df.boxplot(by="X")
```



Можно также задать подмножество столбцов для рисования, а также сгруппировать столбцы:

```
df = pd.DataFrame(np.random.rand(10, 3), columns=["Col1", "Col2", "Col3"])
df["X"] = pd.Series(["A", "A", "A", "A", "A", "B", "B", "B", "B", "B"])
df["Y"] = pd.Series(["A", "B", "A", "B", "A", "B", "A", "B", "A", "B"])
plt.figure();
bp = df.boxplot(column=["Col1", "Col2"], by=["X", "Y"])
```

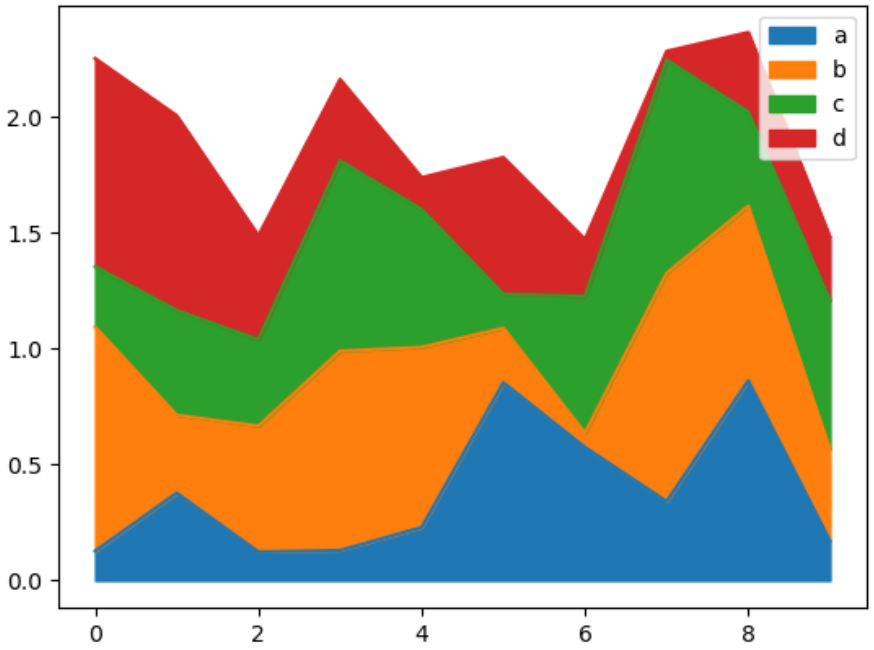


## Area plot (график с закрашенной областью)

Создаются **`Series.plot.area()`** и **`DataFrame.plot.area()`**. По умолчанию участки области складываются. Чтобы создать график области с накоплением, каждый столбец должен содержать либо все положительные, либо все отрицательные значения.

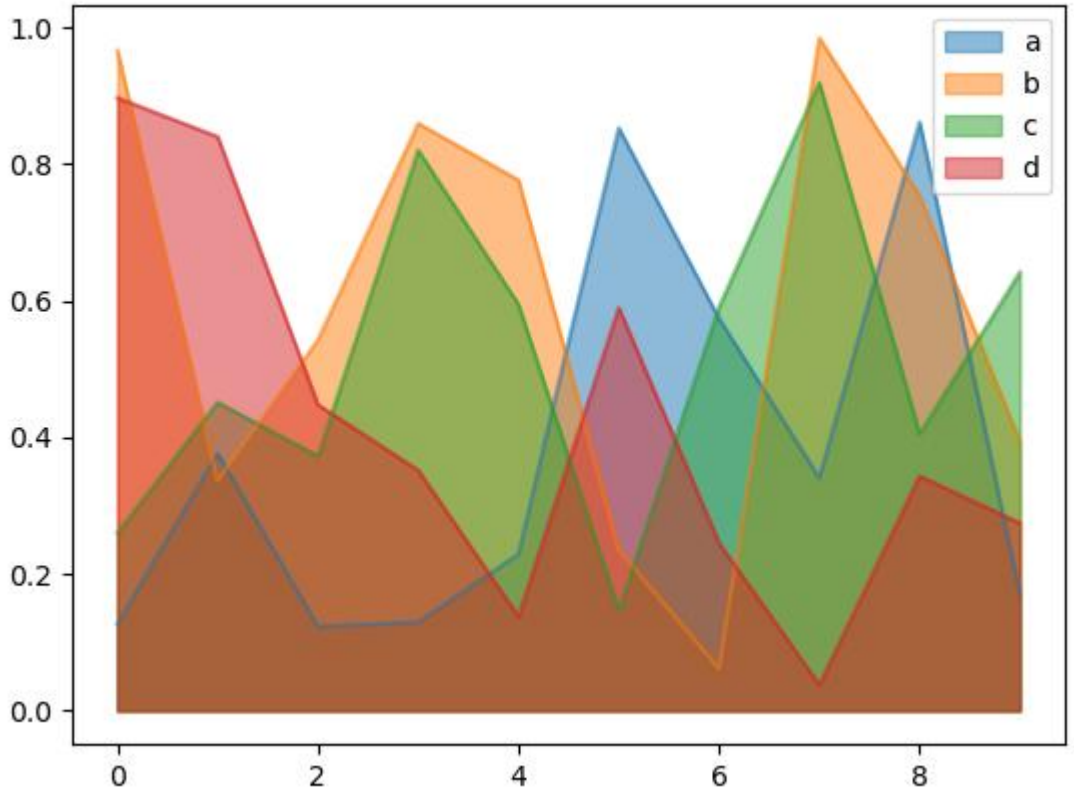
NaN автоматически заменяется на 0. Если хотите отбросить или заполнить их другими значениями, используйте **`dataframe.dropna()`** or **`dataframe.fillna()`** перед вызовом plot.

```
df = pd.DataFrame(np.random.rand(10, 4), columns=["a", "b", "c", "d"])
df.plot.area();
```



Чтобы создать неупакованный график, передайте stacked=False. Прозрачность (Alpha) устанавливается равной 0,5, если не указано иное:

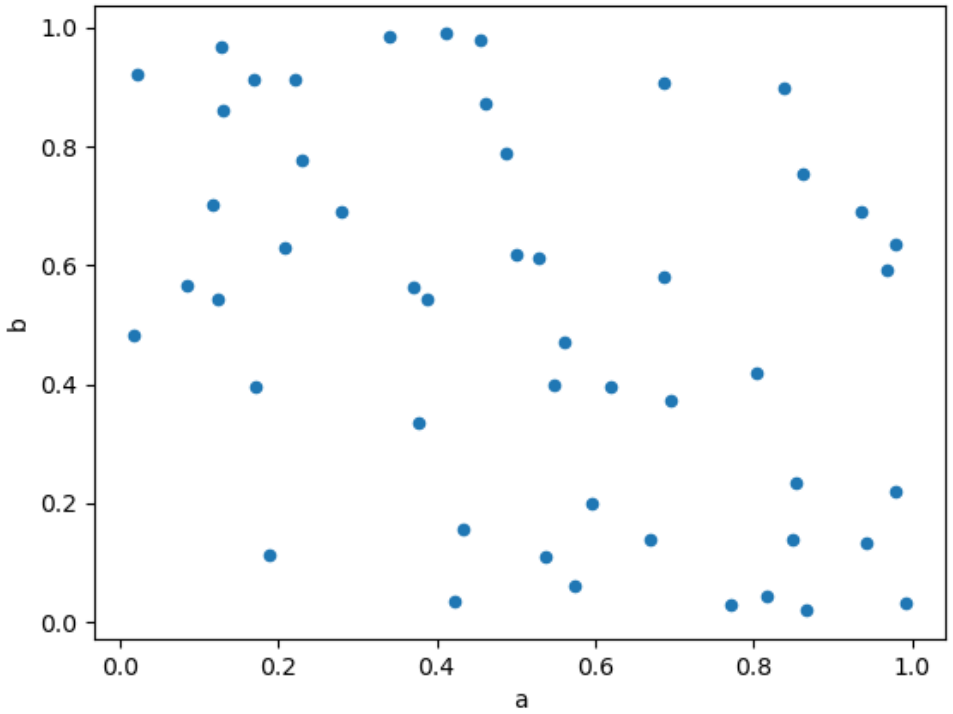
```
In [62]: df.plot.area(stacked=False);
```



### Scatter plot (Точечная диаграмма)

Создаются методом **`DataFrame.plot.scatter()`**. Для точечной диаграммы требуются числовые столбцы для осей x и y. Их можно задать ключевыми словами x и y.

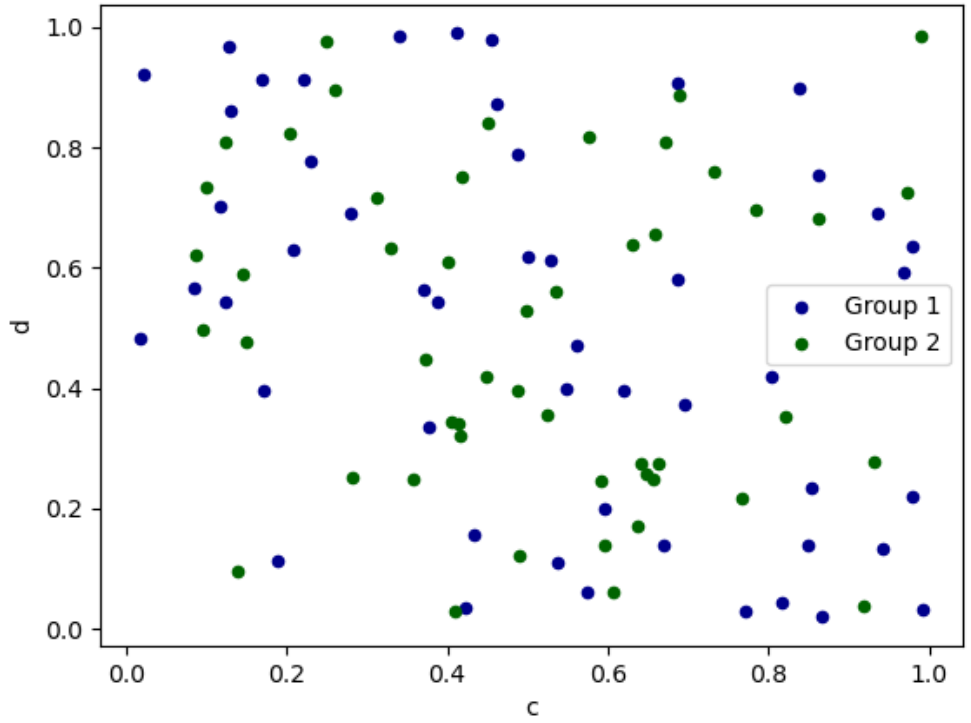
```
In [63]: df = pd.DataFrame(np.random.rand(50, 4), columns=["a", "b", "c",
"d"])
In [64]: df.plot.scatter(x="a", y="b");
```



To plot multiple column groups in a single axes, repeat `plot` method specifying target `ax`. It is recommended to specify `color` and `label` keywords to distinguish each groups.

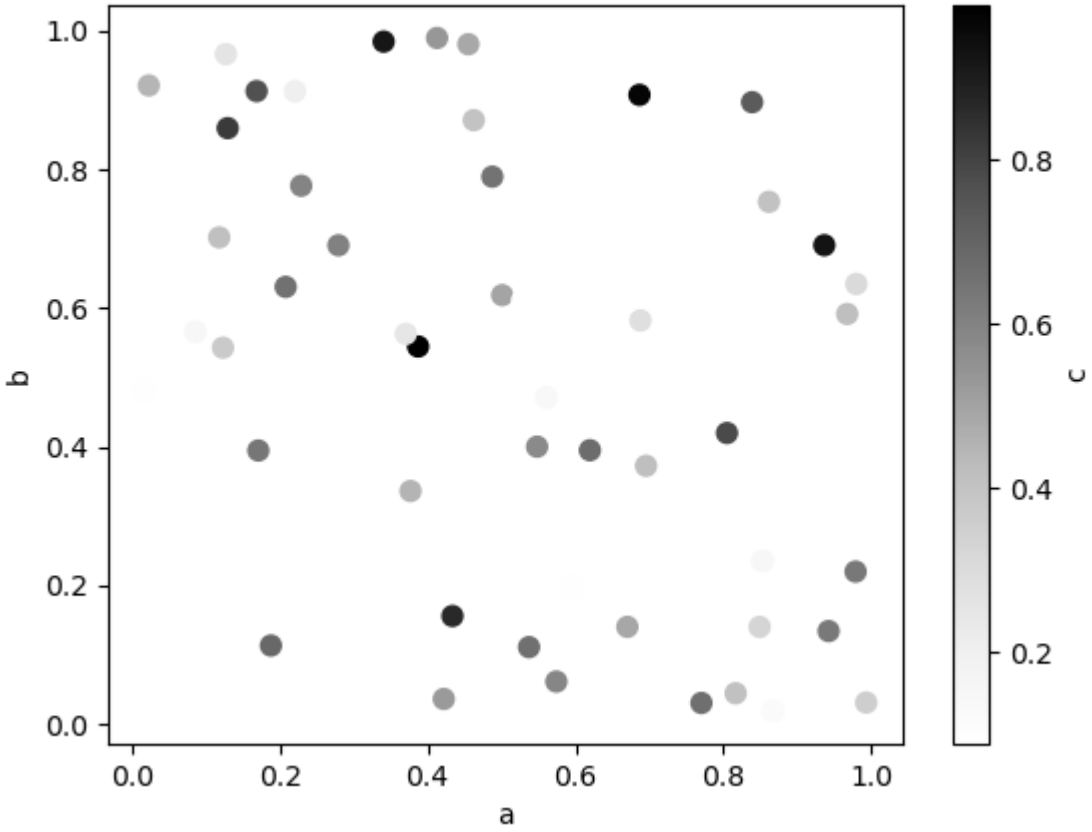
Чтобы построить несколько графиков вместе, повторите метод, указав целевую ось ax. Рекомендуется указать цвета и метки, чтобы различать группы.

```
ax = df.plot.scatter(x="a", y="b", color="DarkBlue", label="Group 1")
df.plot.scatter(x="c", y="d", color="DarkGreen", label="Group 2", ax=ax);
```



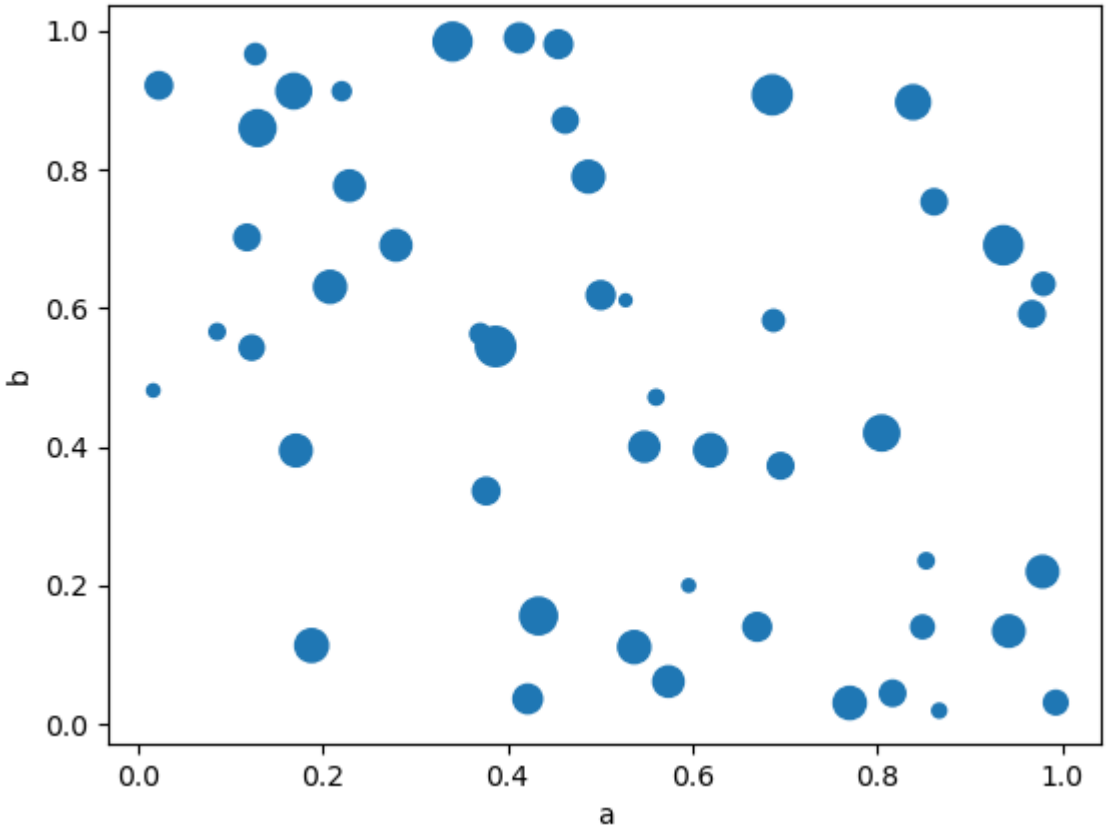
Можно задать еще столбец цветов для каждой точки:

```
In [67]: df.plot.scatter(x="a", y="b", c="c", s=50);
```



Для каждой точки можно еще задать свой размер:

```
In [68]: df.plot.scatter(x="a", y="b", s=df["c"] * 200);
```

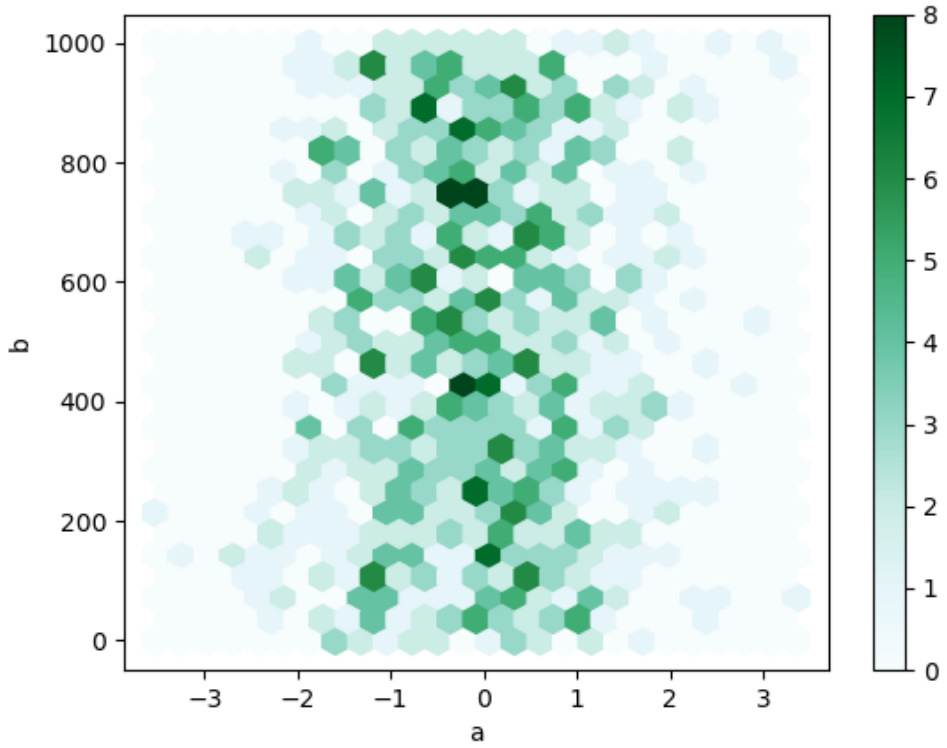



Подробней см. метод **`scatter`** и [matplotlib scatter documentation](#).

### Шестиугольники (Hexagonal bin plot)

Создаются методом **`DataFrame.plot.hexbin()`**. Графики hexbin могут быть альтернативой точечным диаграммам, если данные слишком плотны, чтобы построить каждую точку по отдельности.

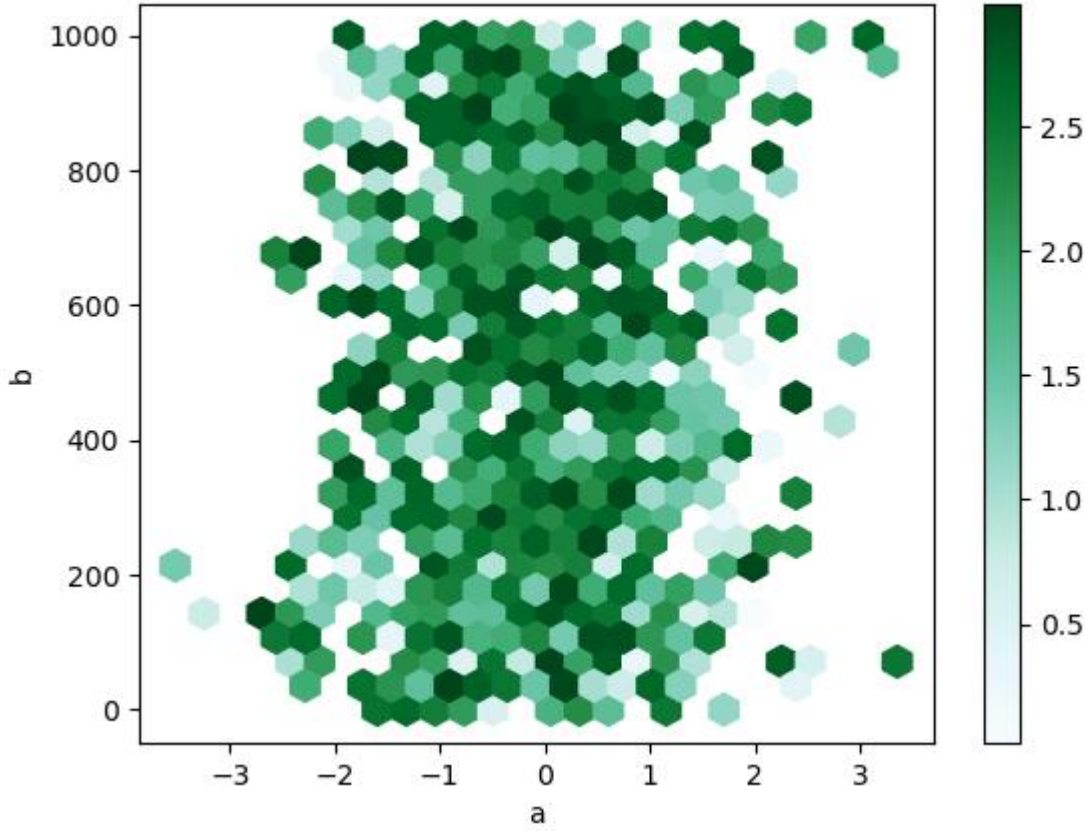
```
In [69]: df = pd.DataFrame(np.random.randn(1000, 2), columns=["a", "b"])
In [70]: df["b"] = df["b"] + np.arange(1000)
In [71]: df.plot.hexbin(x="a", y="b", gridsize=25);
```



Параметр gridsize задает количество шестиугольников в направлении x и по умолчанию равен 100. By default, a histogram of the counts around each `(x, y)` point is computed. You can specify alternative aggregations by passing values to the `C` and `reduce_C_function` arguments. `C` specifies the value at each `(x, y)` point and `reduce_C_function` is a function of one argument that reduces all the values in a bin to a single number (e.g. `mean`, `max`, `sum`, `std`). In this example the positions are given by columns `a` and `b`, while the value is given by column `z`. The bins are aggregated with NumPy's `max` function.

По умолчанию вычисляется количество точек вокруг каждой точки (x, y). Можно указать другие данные, задав C и reduce_C_function. C указывает значение в каждой точке (x, y), а функция reduce_C_function – это функция одного аргумента, которая сводит все значения в ячейке к одному числу (например, среднее, максимальное, сумма, std). В примере позиции задаются столбцами a и b, значение задается столбцом z. Ячейки агрегируются с помощью функции NumPy max.

```
In [72]: df = pd.DataFrame(np.random.randn(1000, 2), columns=["a", "b"])

In [73]: df["b"] = df["b"] = df["b"] + np.arange(1000)

In [74]: df["z"] = np.random.uniform(0, 3, 1000)

In [75]: df.plot.hexbin(x="a", y="b", C="z", reduce_C_function=np.max,
gridsize=25);
```

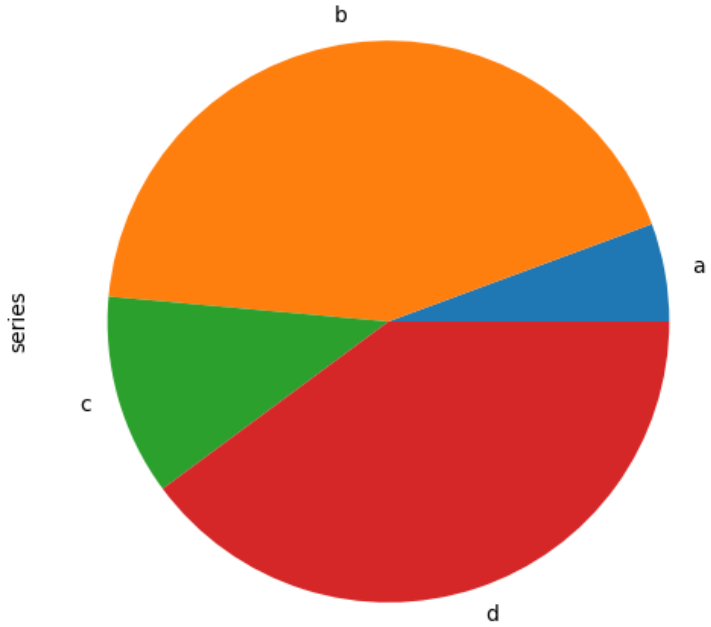



См. метод **`hexbin`** и matplotlib hexbin documentation.

### Круговой график (Pie plot)

Создается методом **`DataFrame.plot.pie()`** или **`Series.plot.pie()`**. NaN автоматически заменяется 0. Будет вызвано исключение ValueError, если среди чисел есть отрицательные числа.

```
In [76]: series = pd.Series(3 * np.random.rand(4), index=["a", "b", "c",
"d"], name="series")
In [77]: series.plot.pie(figsize=(6, 6));
```



### Построение графика с отсутствующими данными

Пакет pandas старается быть прагматичными при выводе `DataFrames` или `Series`, с отсутствующими данными. Пропущенные значения отбрасываются, пропускаются или заполняются в зависимости от типа графика.

| Plot Type | NaN Handling |
|---|---|
| Line | Leave gaps at NaNs |
| Line (stacked) | Fill 0's |
| Bar | Fill 0's |
| Scatter | Drop NaNs |
| Histogram | Drop NaNs (column-wise) |
| Box | Drop NaNs (column-wise) |
| Area | Fill 0's |
| KDE | Drop NaNs (column-wise) |
| Hexbin | Drop NaNs |
| Pie | Fill 0's |

Если действия по умолчанию Вам не подходят, используйте **`fillna()`** или **`dropna()`**, чтобы поменять данные перед выводом.

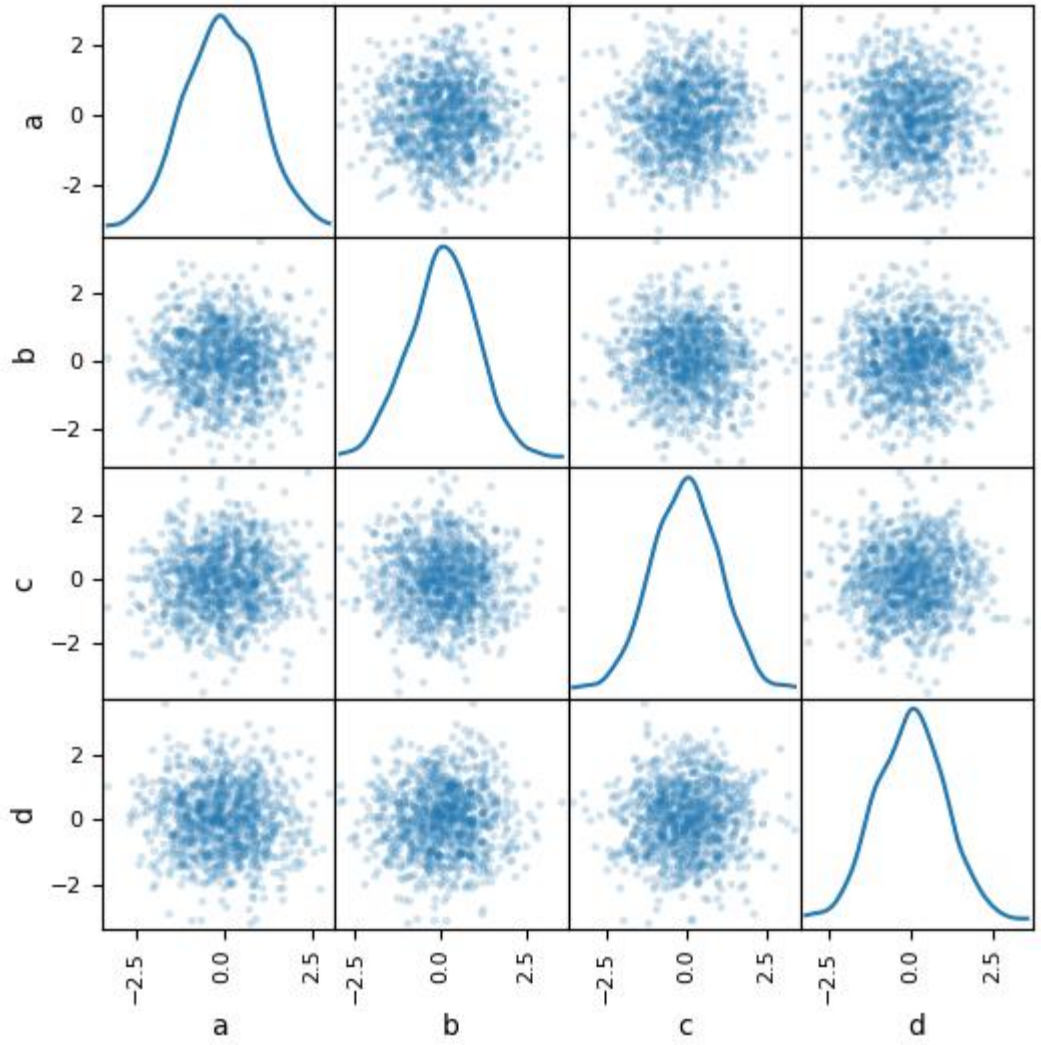
## Инструменты построения графиков (Plotting tools)

These functions can be imported from  and take a **`Series`** or **`DataFrame`** as an argument. Эти функции могут быть импортированы из `pandas.plotting` и принимать в качестве аргумента **`Series`** или **`DataFrame`**.

## График матрицы рассеяния (Scatter matrix plot)

Создается функцией `scatter_matrix` из `pandas.plotting`:

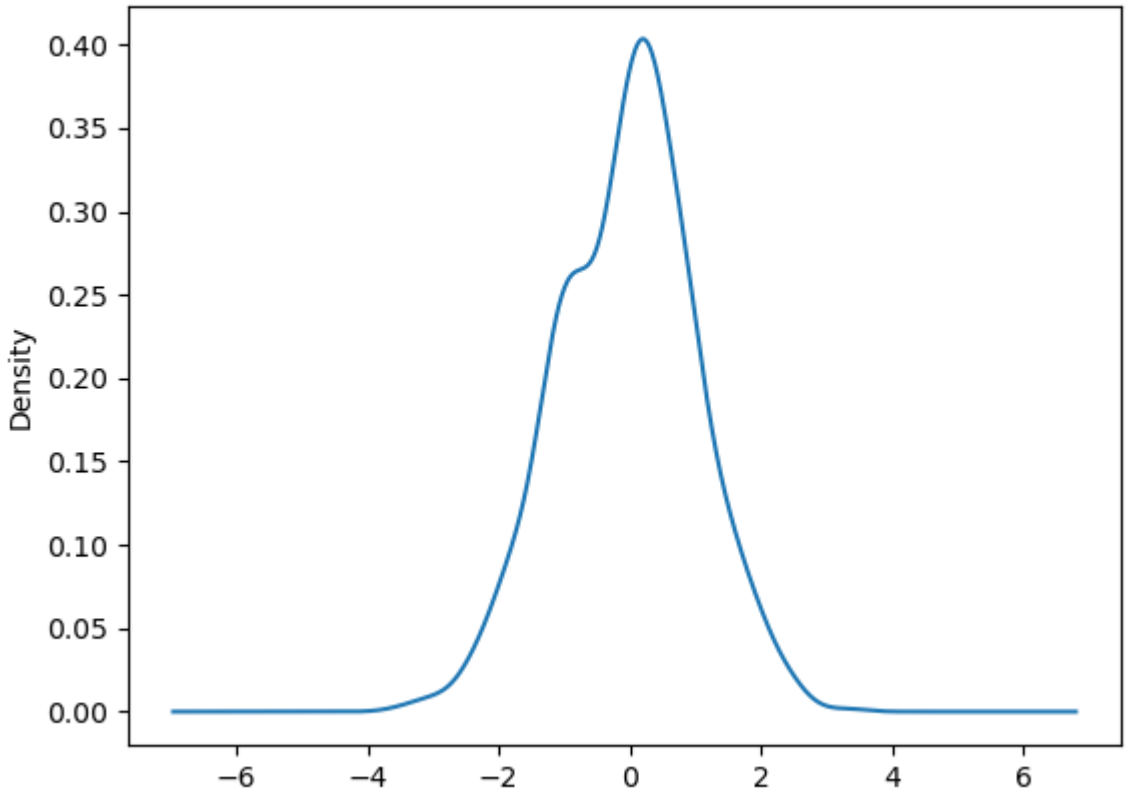
```
In [83]: from pandas.plotting import scatter_matrix

In [84]: df = pd.DataFrame(np.random.randn(1000, 4), columns=["a", "b",
"c", "d"])

In [85]: scatter_matrix(df, alpha=0.2, figsize=(6, 6), diagonal="kde");
```


a
b
c
d
2
0
−2
−2.5
0.0
2.5

## График плотности (Density plot)

Создается методами **`Series.plot.kde()`** и **`DataFrame.plot.kde()`**.

```
In [86]: ser = pd.Series(np.random.randn(1000))

In [87]: ser.plot.kde();
```



## Литература


1. https://matplotlib.org/stable/contents.html
2. **Плас Дж. Вандер Python для сложных задач: наука о данных и машинное обучение. Серия «Бестселлеры O'Reilly». СПб.: Питер, 2018. 576 с.**
3. **Уэс Маккинли Python и анализ данных. М.: ДМК Пресс, 2015. 482 с.**
4. **Доля П.Г. Введение в научный Python 2016. 265 с.**