




ВИЗУАЛИЗАЦИЯ ДАННЫХ В MATPLOTLIB

Тема 3.5

- 
- Несмотря на наличие мощных библиотек визуализации в Python, пакет matplotlib (www.matplotlib.org) считается эталоном, предлагающим самые надежные и эффективные решения.
 - С одной стороны, он позволяет легко строить стандартные графики; с другой стороны, он обеспечивает достаточную гибкость при настройке сложных диаграмм.
 - Кроме того, он тесно интегрирован с библиотеками NumPy и Pandas и поддерживает реализованные в них структуры данных.

СТАТИЧЕСКИЕ ДВУХМЕРНЫЕ ГРАФИКИ

Прежде чем формировать выборку данных и строить графики, необходимо импортировать соответствующие модули и задать ряд базовых настроек:

```
>>> import matplotlib as mpl
```

```
>>> mpl.__version__
```

```
3.2.2
```

```
>>> import matplotlib.pyplot as plt
```

Выберем стиль для графиков (все доступные предустановленные стили:
`plt.style.available`)

```
>>> plt.style.use('seaborn')
```



Применим шрифт serif для всех подписей:

```
>>> mpl.rcParams['font.family'] = 'serif'
```

При работе с Jupyter notebook необходимо указывать, где размещать графики:

```
>>> %matplotlib inline
```

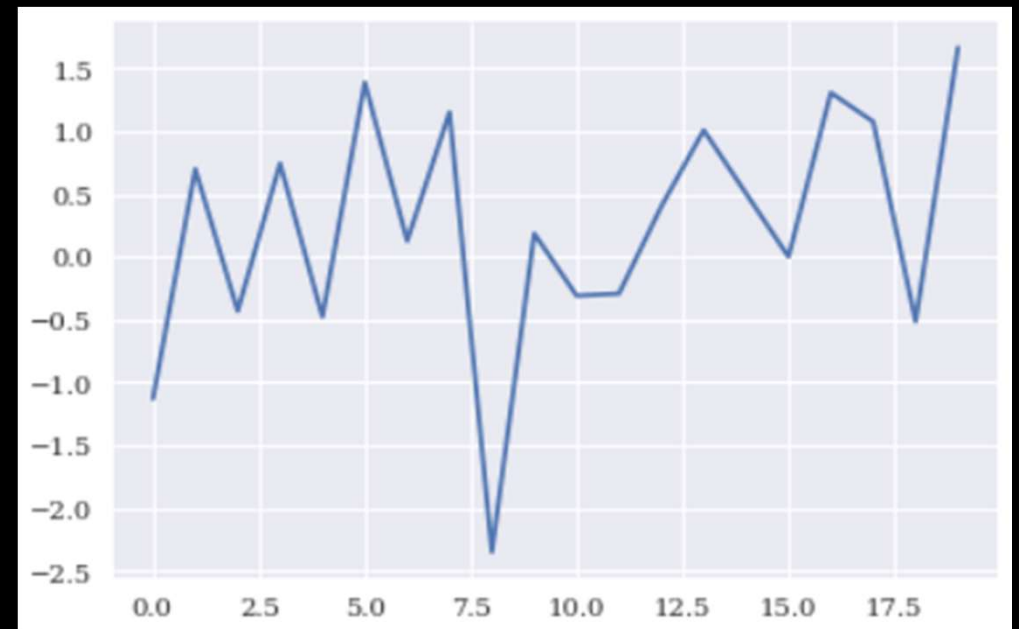
ОДНОМЕРНЫЕ НАБОРЫ ДАННЫХ

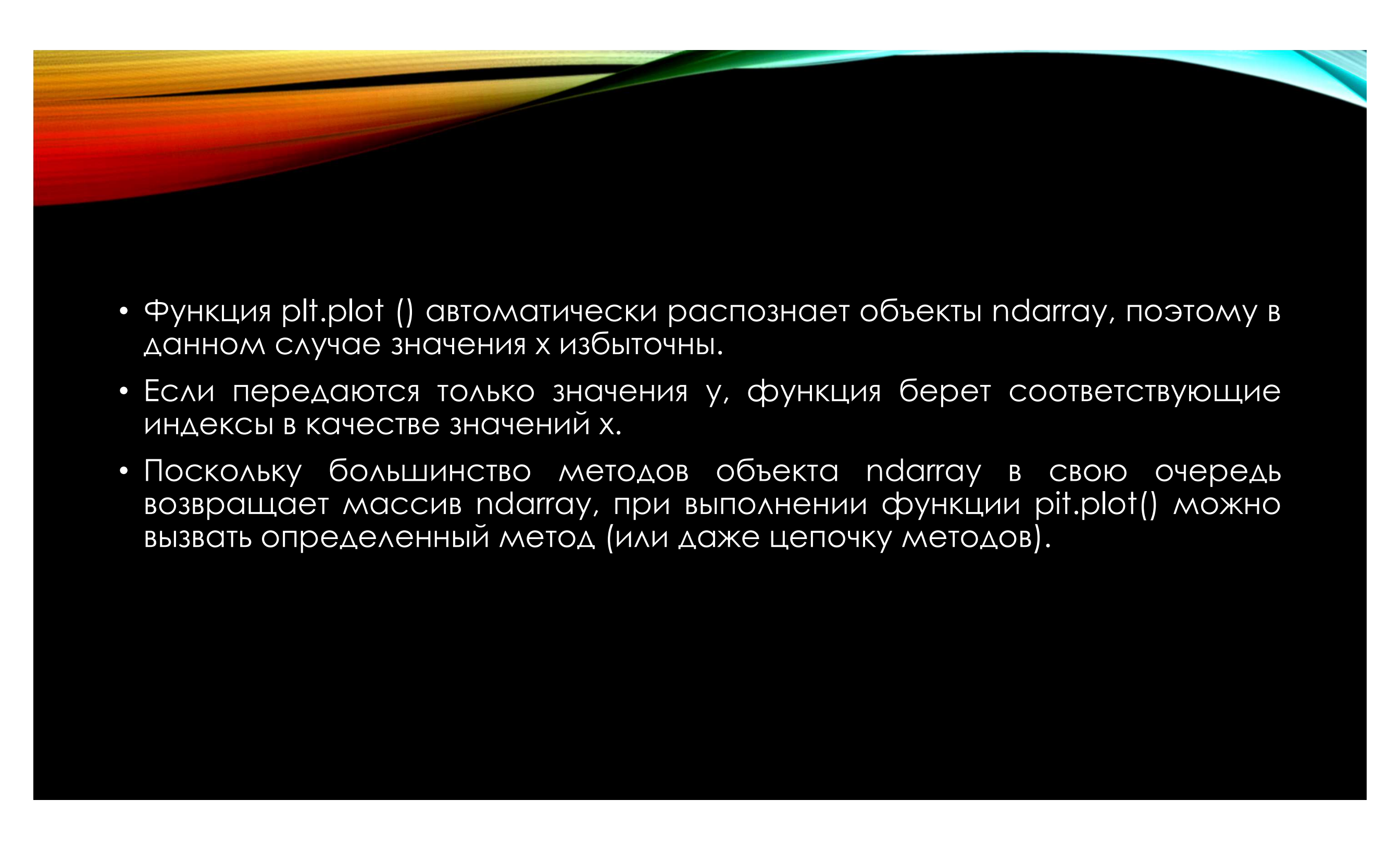
Основная функция построения диаграмм — `plt.plot()`. Ей необходимо передать два набора значений.

Координаты x	Список или массив, содержащий координаты x (значения по оси абсцисс).
Координаты y	Список или массив, содержащий координаты y (значения по оси ординат).

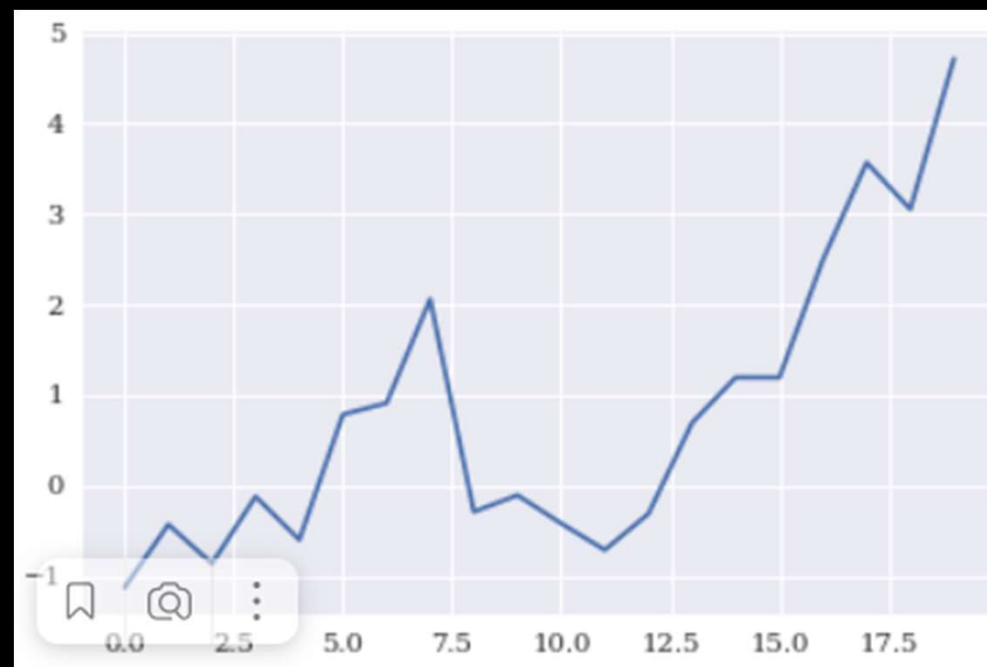
Разумеется, количество значений в обоих наборах должно совпадать.

```
>>> import numpy as np
>>> np.random.seed(1000)
>>> y = np.random.standard_normal(20)
>>> x = np.arange(len(y))
>>> plt.plot(x, y)
```



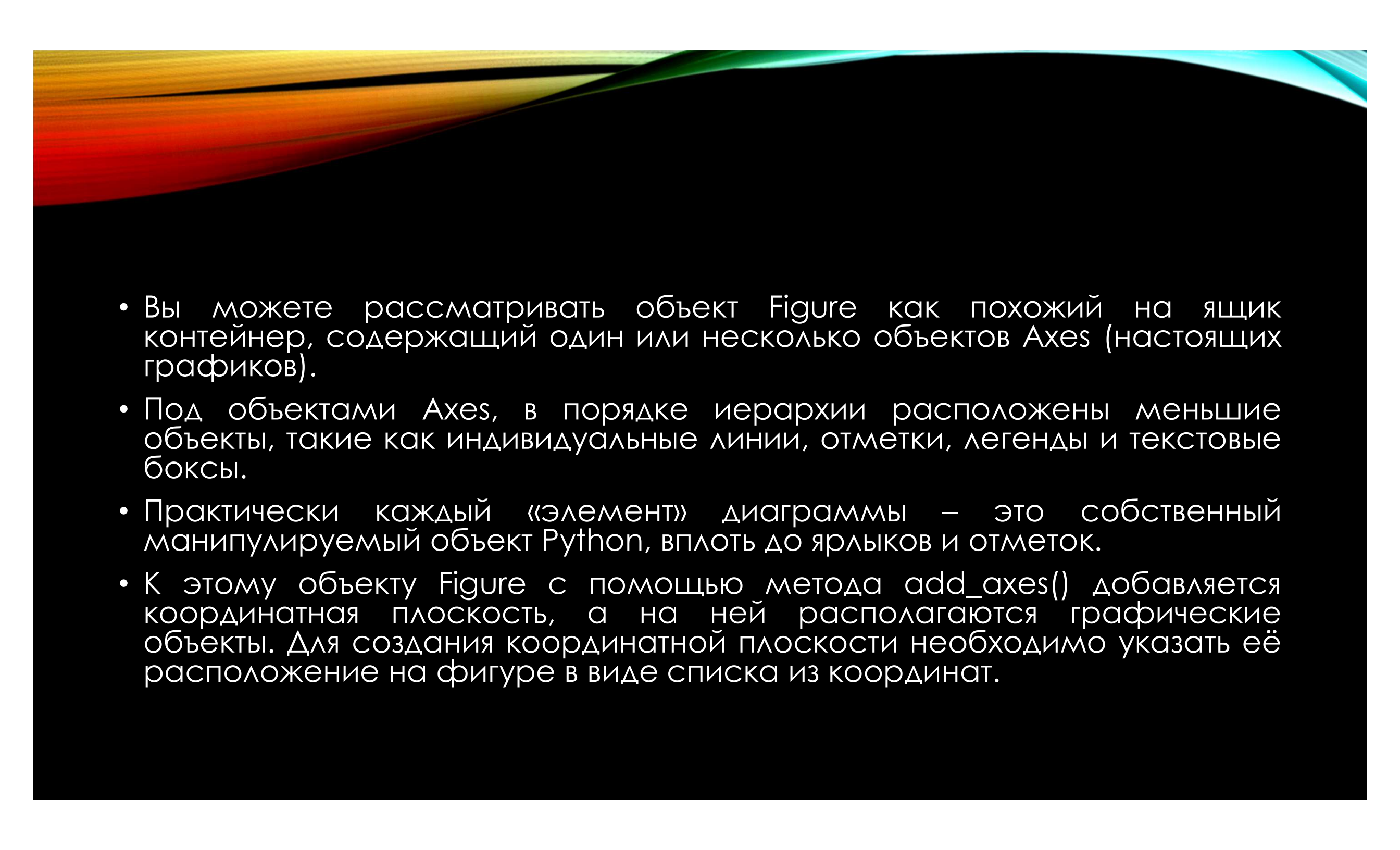
- 
- Функция `plt.plot()` автоматически распознает объекты `ndarray`, поэтому в данном случае значения `x` избыточны.
 - Если передаются только значения `y`, функция берет соответствующие индексы в качестве значений `x`.
 - Поскольку большинство методов объекта `ndarray` в свою очередь возвращает массив `ndarray`, при выполнении функции `plt.plot()` можно вызвать определенный метод (или даже цепочку методов).

Например, при вызове метода `cumsum()` будет построен график изменения накопительной суммы:



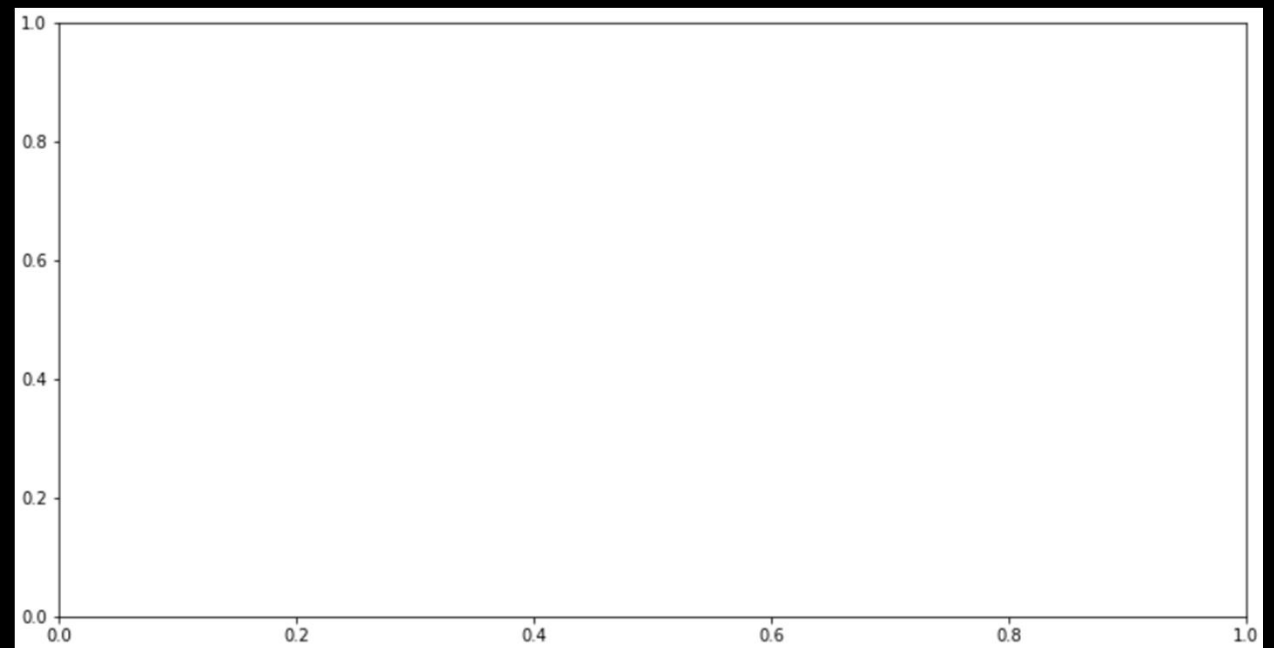
ИЕРАРХИЯ ОБЪЕКТОВ MATPLOTLIB

- Для полноценного использования визуализаций, необходимо понимать, как именно Matplotlib создает свои графики.
- Процесс работы над графиком максимально прозрачен: сначала создаётся объект фигуры Figure (fig), содержащий необходимую информацию и настройки, например размер в дюймах (figsize, восемь дюймов в ширину, четыре — в высоту).
- Объект Figure — это самый важный внешний контейнер для графики matplotlib, который может включать в себя несколько объектов Axes.
- Причиной сложности в понимании может быть название: Axes (оси), на самом деле, превращаются в то, что мы подразумеваем под индивидуальным графиком или диаграммой.

- 
- Вы можете рассматривать объект Figure как похожий на ящик контейнер, содержащий один или несколько объектов Axes (настоящих графиков).
 - Под объектами Axes, в порядке иерархии расположены меньшие объекты, такие как индивидуальные линии, отметки, легенды и текстовые боксы.
 - Практически каждый «элемент» диаграммы – это собственный манипулируемый объект Python, вплоть до ярлыков и отметок.
 - К этому объекту Figure с помощью метода `add_axes()` добавляется координатная плоскость, а на ней располагаются графические объекты. Для создания координатной плоскости необходимо указать её расположение на фигуре в виде списка из координат.

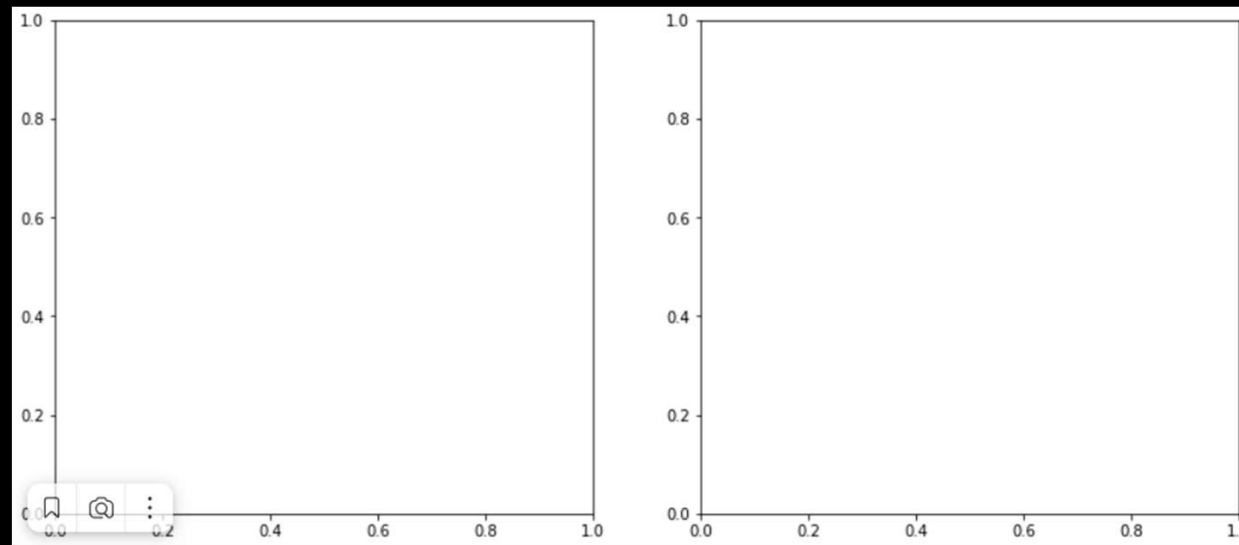
Например, она начинается в левом нижнем углу без отступов (координаты 0, 0) и занимает всё отведённое место в области (100%, ширина и высота равны 1):

```
>>> fig = plt.figure(figsize=(8, 4))  
>>> axes = fig.add_axes([0, 0, 1, 1])
```



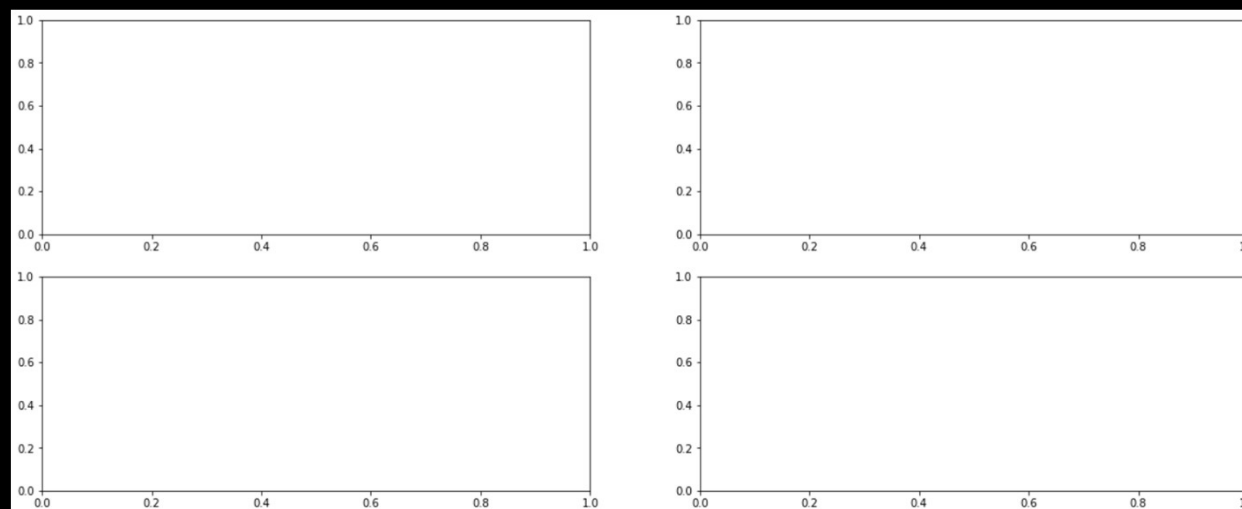
Для создания нескольких графиков необходимо создать несколько осей. При этом в объекте Figure используется функция `subplots`:

```
>>> fig, ax = plt.subplots(1, 2, figsize=(14, 6))
```



Данная инструкция создает объект Figure fig, набор графиков ax (типом которого является ndarray) в формате 1 строка, 2 столбца. Общий размер фигуры 14x6 (1 ед = 72 пикселя). Отметим, что фигура и оси создаются автоматически (аналог конструкции fig = plt.figure()). Для сравнения:

```
>>> fig, ax = plt.subplots(2,2, figsize=(20, 8))
```



ПАРАМЕТРЫ ОСЕЙ

Параметр	Описание
	Возвращает текущие предельные значения осей
off	Скрывает линии осей и подписи к ним
equal	Выравнивает масштаб осей
scaled	Выравнивает масштаб за счет изменения размерностей
tight	Приводит к отображению всех данных (за счет сжатия осей)
image	Приводит к отображению всех данных (предельные значения определяются данными)
[xmin, xmax, ymin, ymax]	Устанавливает предельные значения на осях

ТЕКСТОВЫЕ ЭЛЕМЕНТЫ ГРАФИКА

Добавим подписи. Заголовок графика:

```
>>> plt.title("Случайные данные")
```

Подписи осей:

```
>>> plt.xlabel("x")
```

```
>>> plt.ylabel("y")
```


СЛУЧАЙ С НЕСКОЛЬКИМИ ГРАФИКАМИ

```
>>> import numpy as np
>>> import matplotlib as mpl
>>> import matplotlib.pyplot as plt
```


Импортируем необходимые библиотеки.

```
>>> np.random.seed(666)
>>> y =
np.random.standard_normal(100)
```

```
>>> x = np.arange(len(y))
```

Генерируем объект данных.

```
>>> plt.figure(figsize=(16,6))
>>> plt.figtext(0.2, 0, "Рис. 1. Графики
нормального распределения")
>>> plt.suptitle("Нормальное
распределение", fontsize = 20,
color='green', weight='bold')
```



Оформляем первичный объект-контейнер Figure. Его размер 16x6. Подпись рисунка задается методом `.figtext()`, координаты соответствуют внутренним координатам объекта Figure (0 – 0%, 1 – 100%) по горизонтали / вертикали. Заголовок для всей фигуры (общий) – метод `.suprtitle()`


```
>>> plt.subplot(121)
>>> plt.plot(x,y,"bo")
```

Создаем график. Формат «121» означает, что выполняется настройка графика в расположении 1 строка, 2 столбца, график № 1.



```
>>> plt.title("Случайные величины", fontsize = 15, color='blue', weight='bold')
>>> plt.xlabel("x", fontsize = 15, color='blue')
>>> plt.ylabel("y", fontsize = 15, color='blue')
```

Выполняем подпись для графика, а также настраиваем подписи его осей.



```
>>> plt.text(0.2, 0.25, "Начало")
```

```
>>> plt.annotate("Выброс", xy = (61, 3.4), xytext = (20, 2.2), color = 'blue',  
weight = "bold", arrowprops = dict(facecolor = 'blue', shrink=0.1))
```

Выполняем пояснительные надписи на графике. Координаты соответствуют координатным осям графика.



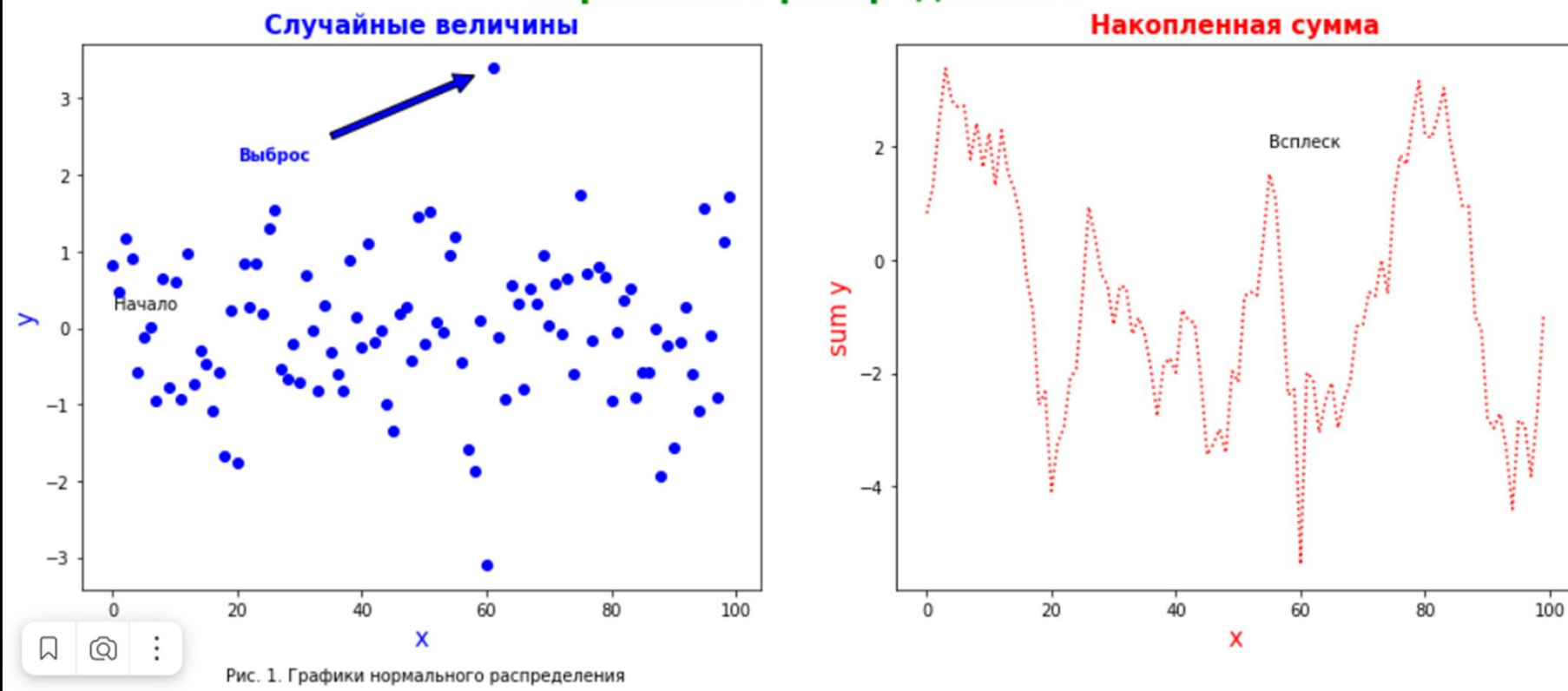
```
>>> plt.subplot(122)
>>> plt.plot(x,y.cumsum(), "r:")
```

Создаем график в расположении 1 строка, 2 столбца, график № 2.

```
>>> plt.title("Накопленная сумма",fontsize = 15, color='red', weight='bold')
>>> plt.xlabel("x", fontsize = 15,color='red')
>>> plt.ylabel("sum y", fontsize = 15, color='red')
```

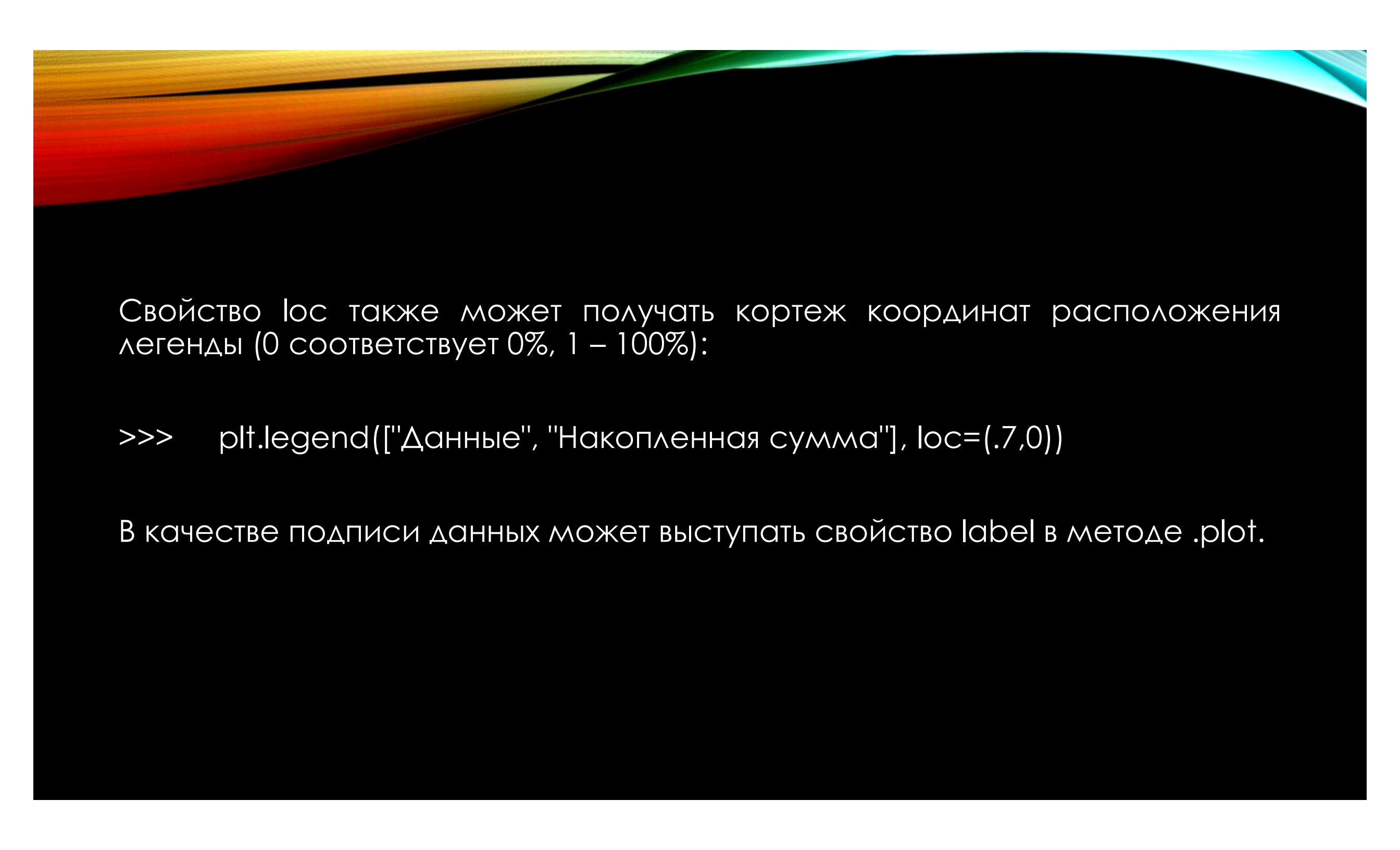
Выполняем подпись для графика, а также настраиваем подписи его осей.

Нормальное распределение



НАСТРОЙКА ЛЕГЕНДЫ

Расположение	Описание
По умолчанию	Справа вверху
0	В самом удобном месте
1	Справа вверху
2	Слева вверху
3	Слева внизу
4	Справа внизу
5	Справа
6	Слева по центру
7	Справа по центру
8	Внизу по центру
9	Вверху по центру
10	По центру



Свойство `loc` также может получать кортеж координат расположения легенды (0 соответствует 0%, 1 – 100%):

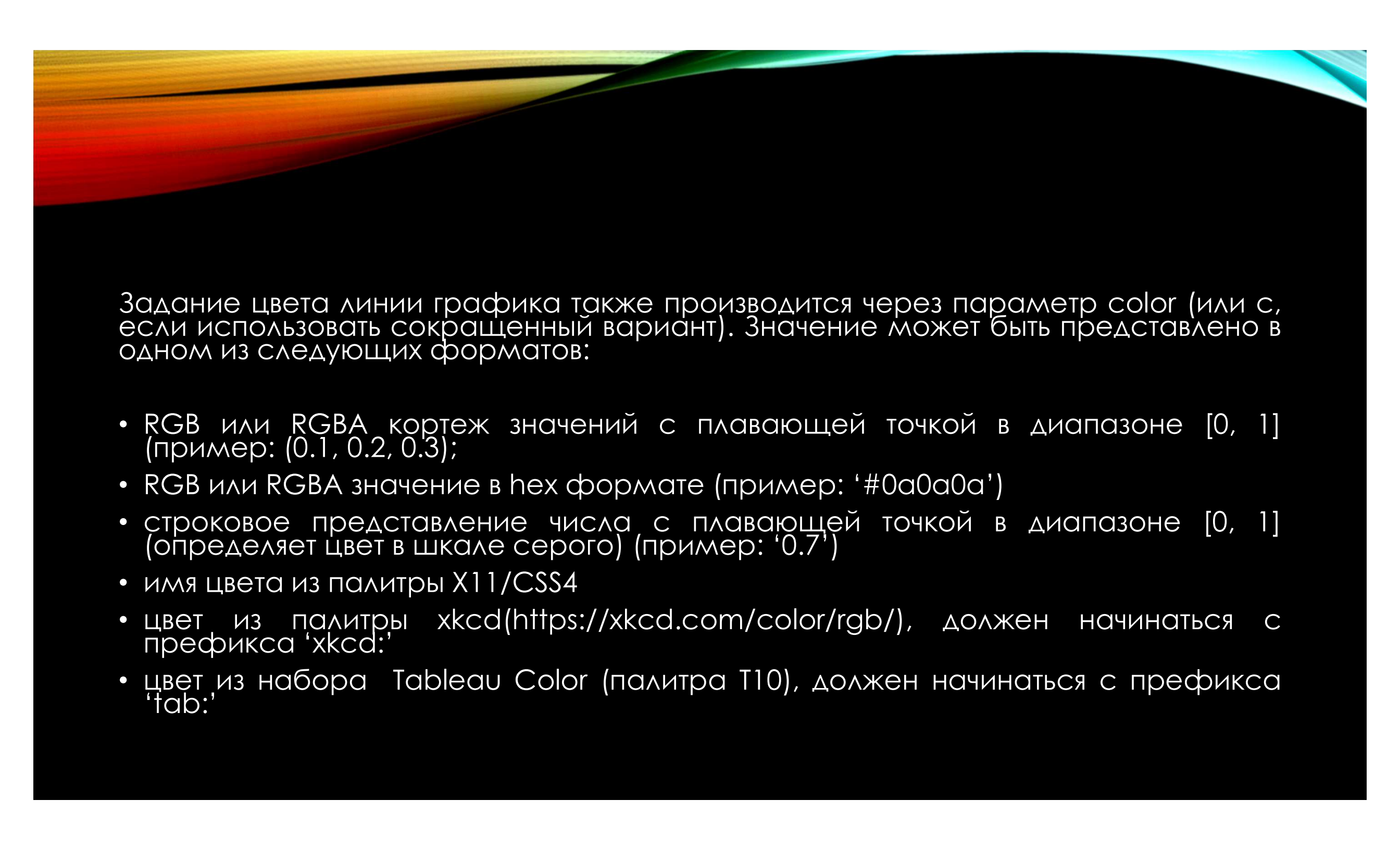
```
>>> plt.legend(["Данные", "Накопленная сумма"], loc=(.7,0))
```

В качестве подписи данных может выступать свойство `label` в методе `.plot`.

Параметр	Тип	Описание
fontsize	int или float или {'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', 'xx-large'}	Размера шрифта надписи легенды
frameon	bool	Отображение рамки легенды
framealpha	None или float	Прозрачность легенды
facecolor	None или str	Цвет заливки
edgecolor	None или str	Цвет рамки
title	None или str	Текст заголовка
title_fontsize	None или str	Размер шрифта

ИСПОЛЬЗОВАНИЕ ЦВЕТА

Символ	Цвет
b	Синий
g	Зеленый
r	Красный
c	Голубой
m	Пурпурный
y	Желтый
k	Черный
w	Белый



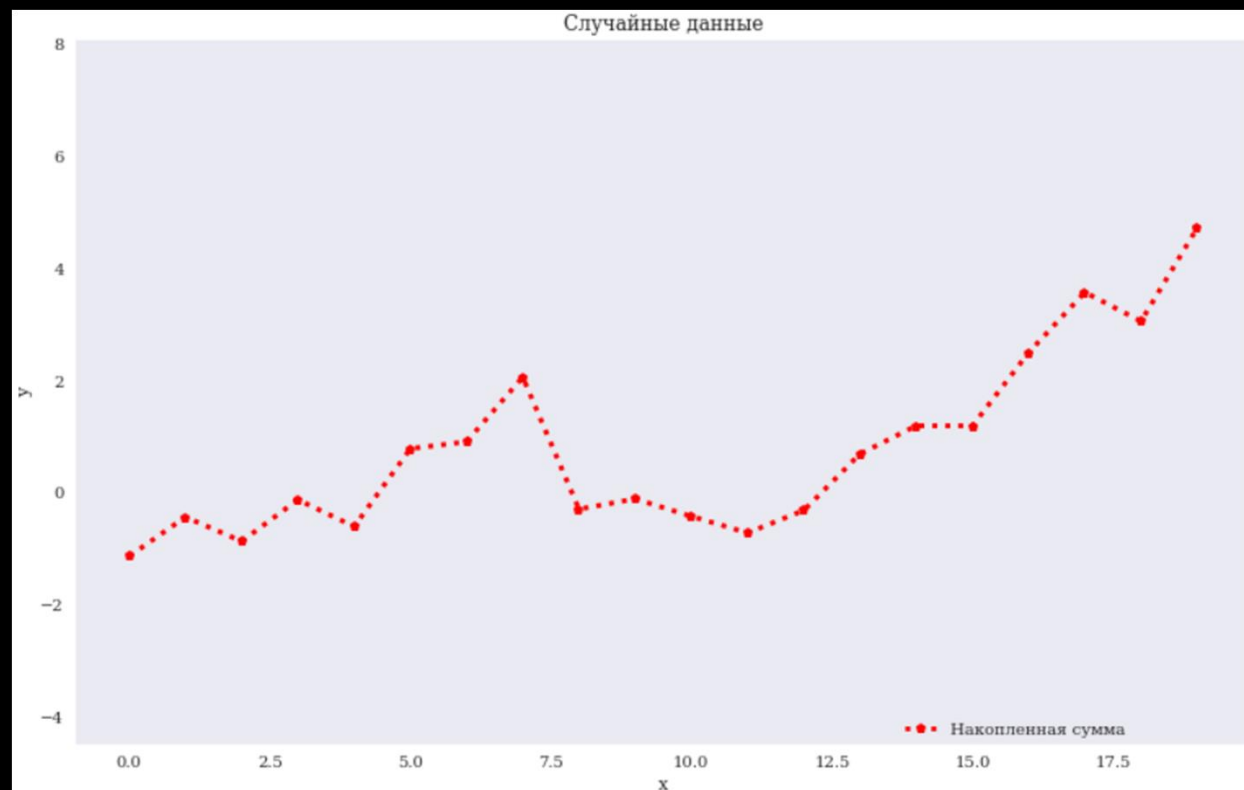
Задание цвета линии графика также производится через параметр color (или c, если использовать сокращенный вариант). Значение может быть представлено в одном из следующих форматов:

- RGB или RGBA кортеж значений с плавающей точкой в диапазоне [0, 1] (пример: (0.1, 0.2, 0.3));
- RGB или RGBA значение в hex формате (пример: '#0a0a0a')
- строковое представление числа с плавающей точкой в диапазоне [0, 1] (определяет цвет в шкале серого) (пример: '0.7')
- имя цвета из палитры X11/CSS4
- цвет из палитры xkcd(<https://xkcd.com/color/rgb/>), должен начинаться с префикса 'xkcd:'
- цвет из набора Tableau Color (палитра T10), должен начинаться с префикса 'tab:'

Символ	Стиль	Символ	Стиль
-	Сплошная линия	3	Маркер в виде трехлучевой звезды с вершиной влево
--	Штриховая линия	4	Маркер в виде трехлучевой звезды с вершиной вправо
-.:	Штрих-пунктирная линия	s	Маркер в виде квадрата
:	Пунктирная линия	p	Маркер в виде пятиугольника
.	Маркер в виде точки	*	Маркер в виде звездочки
,	Маркер в виде пикселя	h	Маркер в виде шестиугольника, тип 1
o	Маркер в виде кружка	H	Маркер в виде шестиугольника, тип 2
v	Маркер в виде треугольника с вершиной вниз	+	Маркер в виде знака «плюс»
^	Маркер в виде треугольника с вершиной вверх	x	Маркер в виде символа «x»
<	Маркер в виде треугольника с вершиной влево	D	Маркер в виде ромба
>	Маркер в виде треугольника с вершиной вправо	d	Маркер в виде узкого ромба
1	Маркер в виде трехлучевой звезды с вершиной вниз		Маркер в виде вертикальной линии
2	Маркер в виде трехлучевой звезды с вершиной вверх	_	Маркер в виде горизонтальной линии

Толщина линий задается параметром `lw`. Применим к графику стиль - красная линия пунктирная с маркером в виде пятиугольника, толщиной в 3:

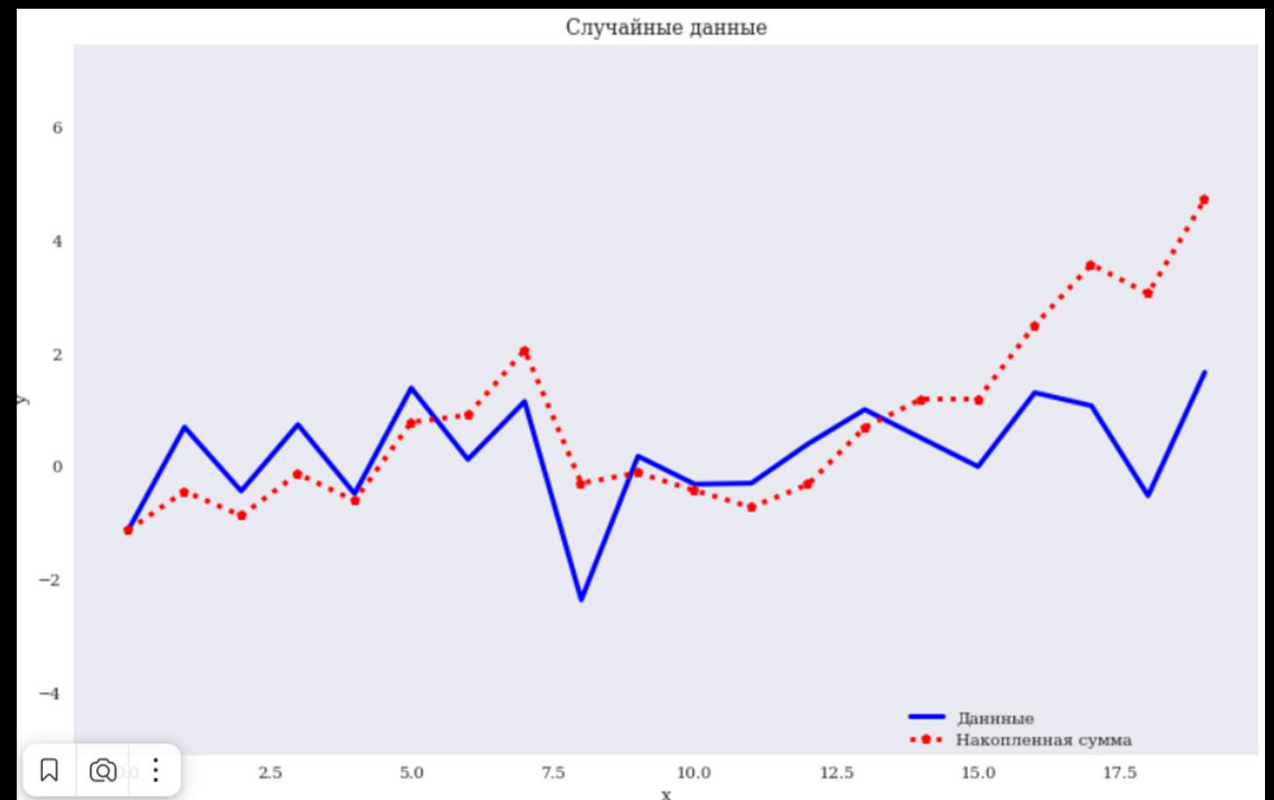
```
>>> plt.plot(y.cumsum(), "rp:", lw = 3)
```




ПОСТРОЕНИЕ НЕСКОЛЬКИХ ГРАФИКОВ

Графики в одном масштабе отображаются автоматически, достаточно перечислить их в методе `.plot`:

```
>>> axes.plot(x, y, "b", lw=3)  
>>> axes.plot(x, y.cumsum(), "rp:", lw=3)
```





Для отображения данных в разном масштабе необходимо создавать дополнительные оси:

```
>>> import matplotlib as mpl
>>> import matplotlib.pyplot as plt
>>> import numpy as np

>>> plt.style.use('seaborn')
>>> mpl.rcParams['font.family']='serif'
```

```
>>> np.random.seed(400)
>>> y =
np.random.standard_normal(50)
>>> x = np.arange(len(y))

>>> fig = plt.figure(figsize=(10, 6))
>>> fig, ax1 = plt.subplots()
```

```
>>> plt.title('Случайные данные')
```

```
>>> plt.xlabel("x")
```

```
>>> plt.ylabel("y")
```

```
>>> ax1.grid(False)
```

```
>>> ax1.axis('equal')
```

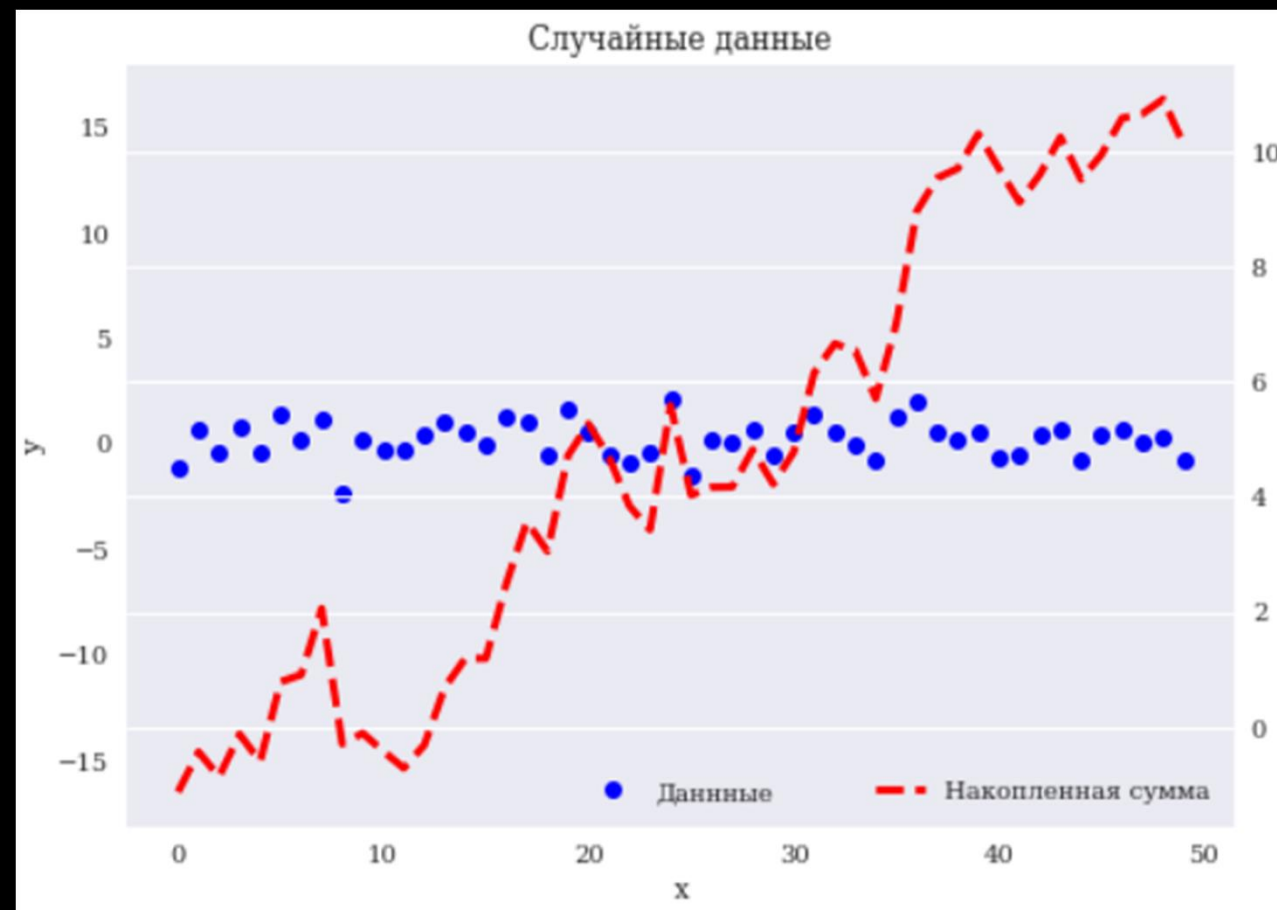
```
>>> ax1.plot(x, y, "bo", lw=3, label =  
"Данные")
```

```
>>> ax2 = ax1.twinx()
```

```
>>> ax2.plot(x, y.cumsum(), "r--", lw =  
3, label = "Накопленная сумма")
```

```
>>> ax1.legend(loc=8)
```

```
>>> ax2.legend(loc=4)
```





Размещение нескольких графиков в рамках одной фигуры на разных полях:


- использование функции `subplot()` для указания места размещения поля с графиком;
- использование функции `subplots()` для предварительного задания сетки, в которую будут укладываться поля;
- использование `GridSpec`, для более гибкого задания геометрии размещения полей с графиками в сетке.

Самый простой способ представить графики в отдельных полях – это использовать функцию `subplot()` для задания их мест размещения

Чаще всего используют следующие варианты вызова subplot:

```
>>> subplot(nrows, ncols, index)
```

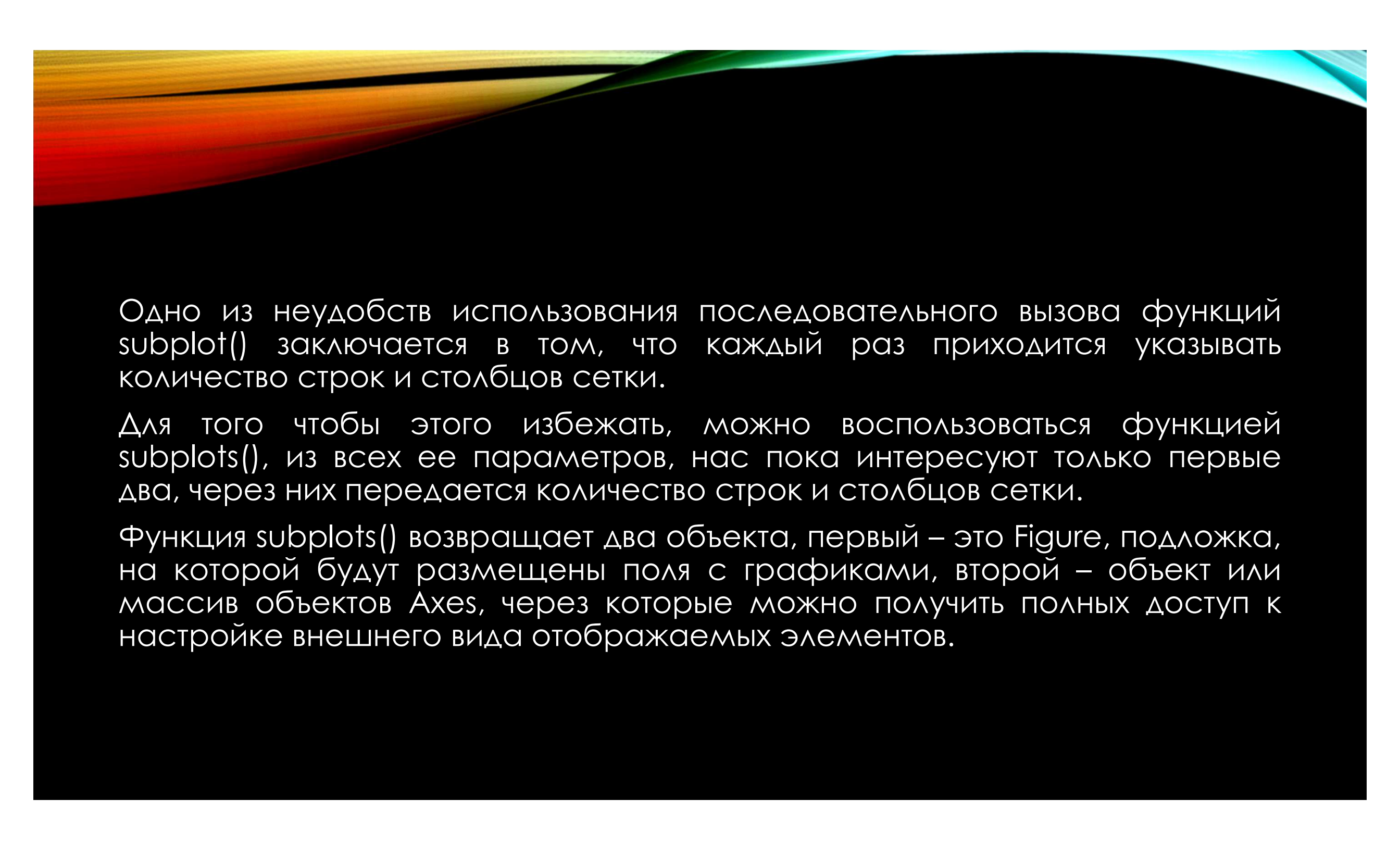
nrows: int	Количество строк.
ncols: int	Количество столбцов.
index: int	Местоположение элемента.



Позиция, в виде трехзначного числа, содержащего информацию о количестве строк, столбцов и индексе, например 212, означает подготовить разметку с двумя строками и одним столбцов, элемент вывести в первую позицию второй строки. Этот вариант можно использовать, если количество строк и столбцов сетки не более 10.

```
>>> plt.subplot(212)
```

Создаётся график в размещении 2 строки, 1 столбец, график №2 (нижний).



Одно из неудобств использования последовательного вызова функций `subplot()` заключается в том, что каждый раз приходится указывать количество строк и столбцов сетки.

Для того чтобы этого избежать, можно воспользоваться функцией `subplots()`, из всех ее параметров, нас пока интересуют только первые два, через них передается количество строк и столбцов сетки.

Функция `subplots()` возвращает два объекта, первый – это `Figure`, подложка, на которой будут размещены поля с графиками, второй – объект или массив объектов `Axes`, через которые можно получить полный доступ к настройке внешнего вида отображаемых элементов.


```
>>> x = [1, 5, 10, 15, 20]
```

```
>>> y1 = [1, 7, 3, 5, 11]
```

```
>>> y2 = [i*1.2 + 1 for i in y1]
```

```
>>> y3 = [i*1.2 + 1 for i in y2]
```

```
>>> y4 = [i*1.2 + 1 for i in y3]
```

```
>>> plt.figure(figsize=(12, 7))
```

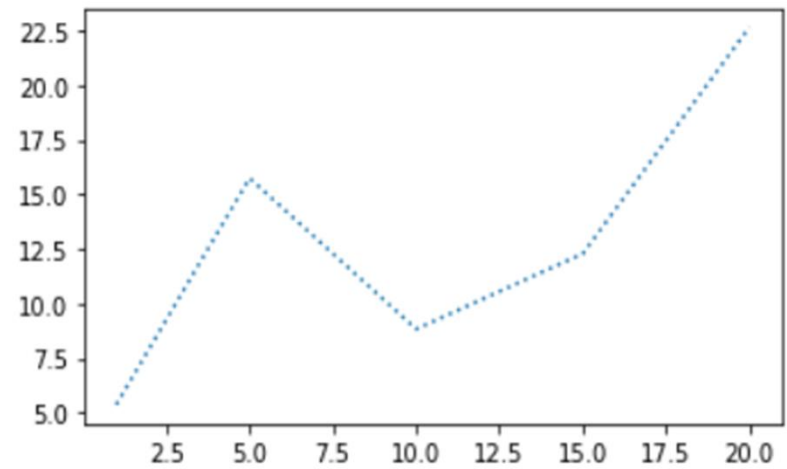
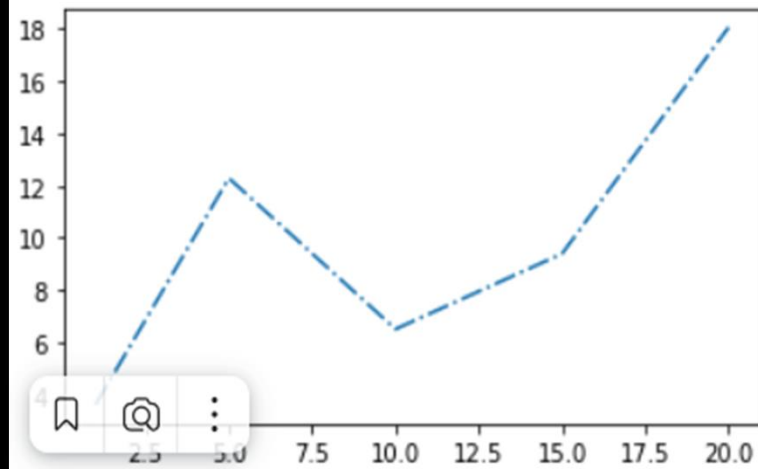
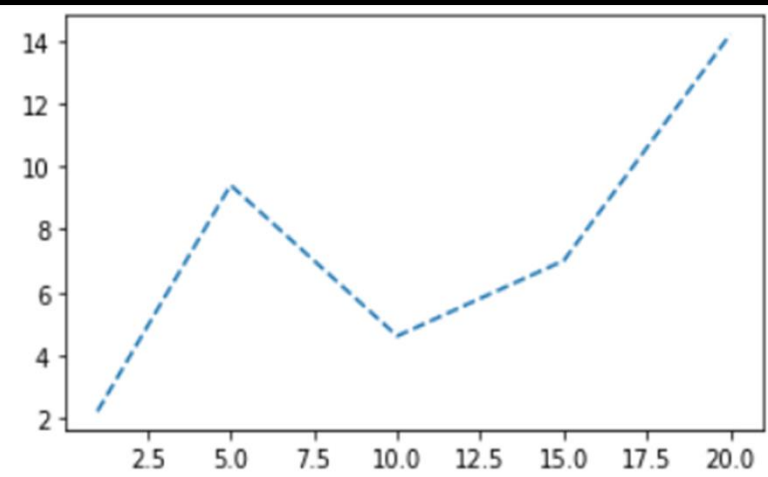
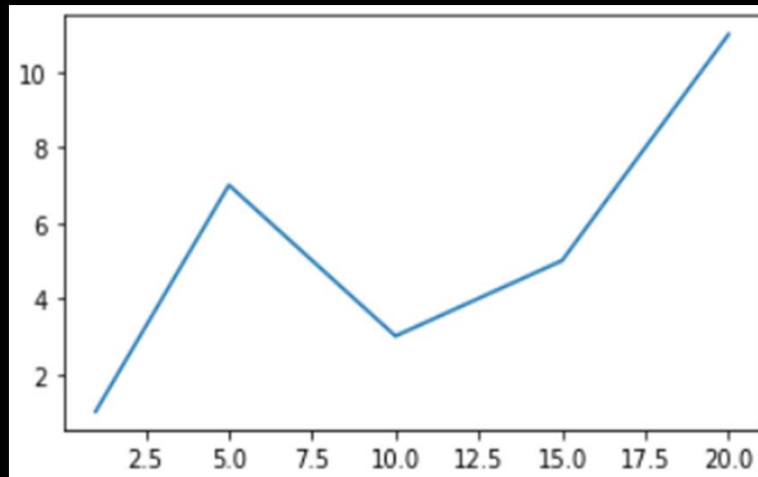
```
>>> fig, axs = plt.subplots(2, 2,  
figsize=(12, 7))
```

```
>>> axs[0, 0].plot(x, y1, '-')
```

```
>>> axs[0, 1].plot(x, y2, '--')
```

```
>>> axs[1, 0].plot(x, y3, '-.')
```

```
>>> axs[1, 1].plot(x, y4, ':')
```



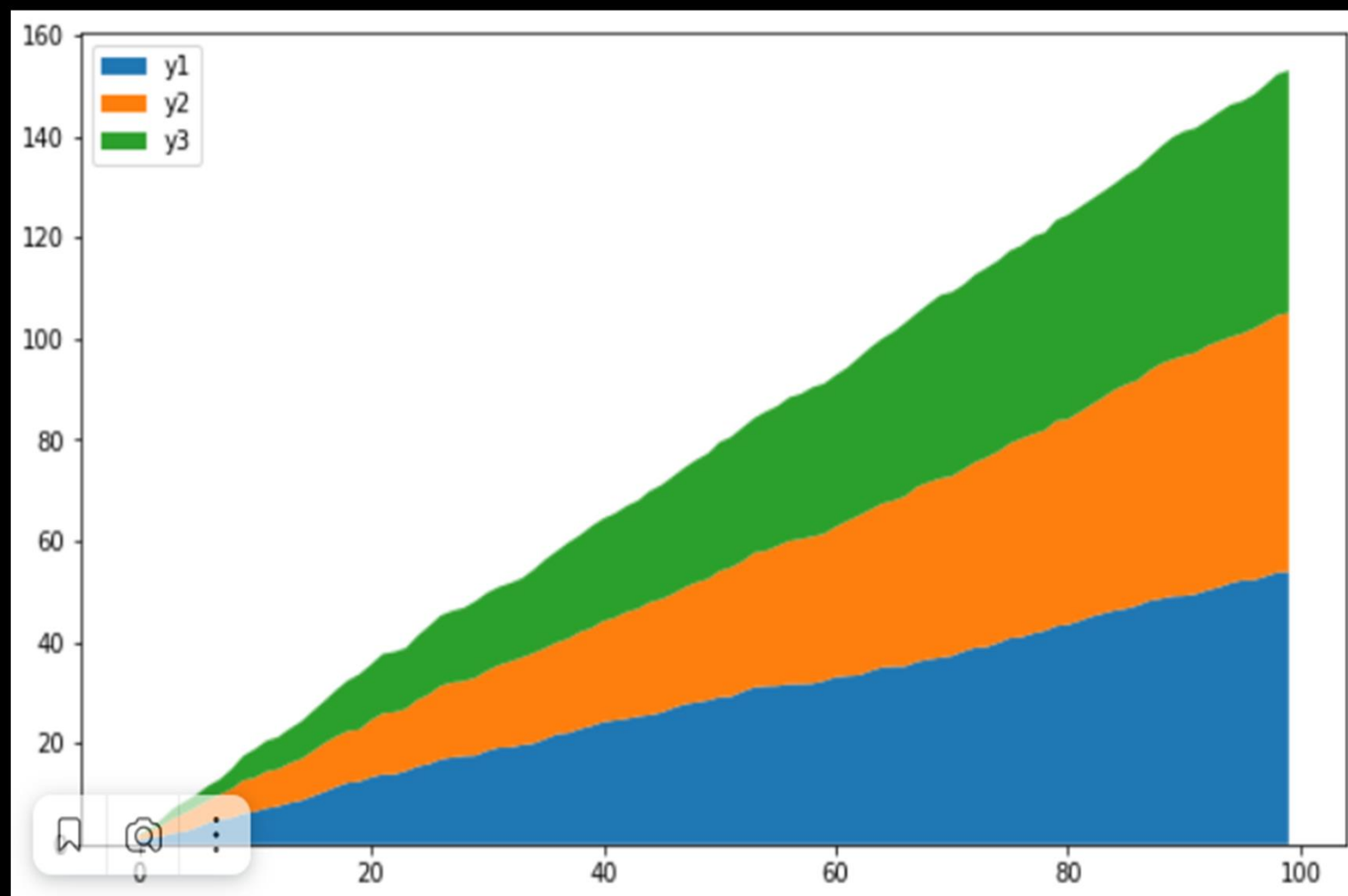
СТЕКОВЫЙ ГРАФИК

В стековом графике отдельные графики отображаются друг над другом, и каждый следующий является суммой предыдущего и заданного набора данных. Для построения стекового графика используется функция `stackplot()`:

```
>>> import numpy as np
>>> import matplotlib.pyplot as plt

>>> x = np.arange(100)
>>> y1 = np.random.rand(100).cumsum()
```

```
>>> y2 = np.random.rand(100).cumsum()
>>> y3 = np.random.rand(100).cumsum()
>>> labels = ["y1", "y2", "y3"]
>>> fig, ax = plt.subplots()
>>> ax.stackplot(x, y1, y2, y3, labels=labels)
>>> ax.legend(loc='upper left')
```



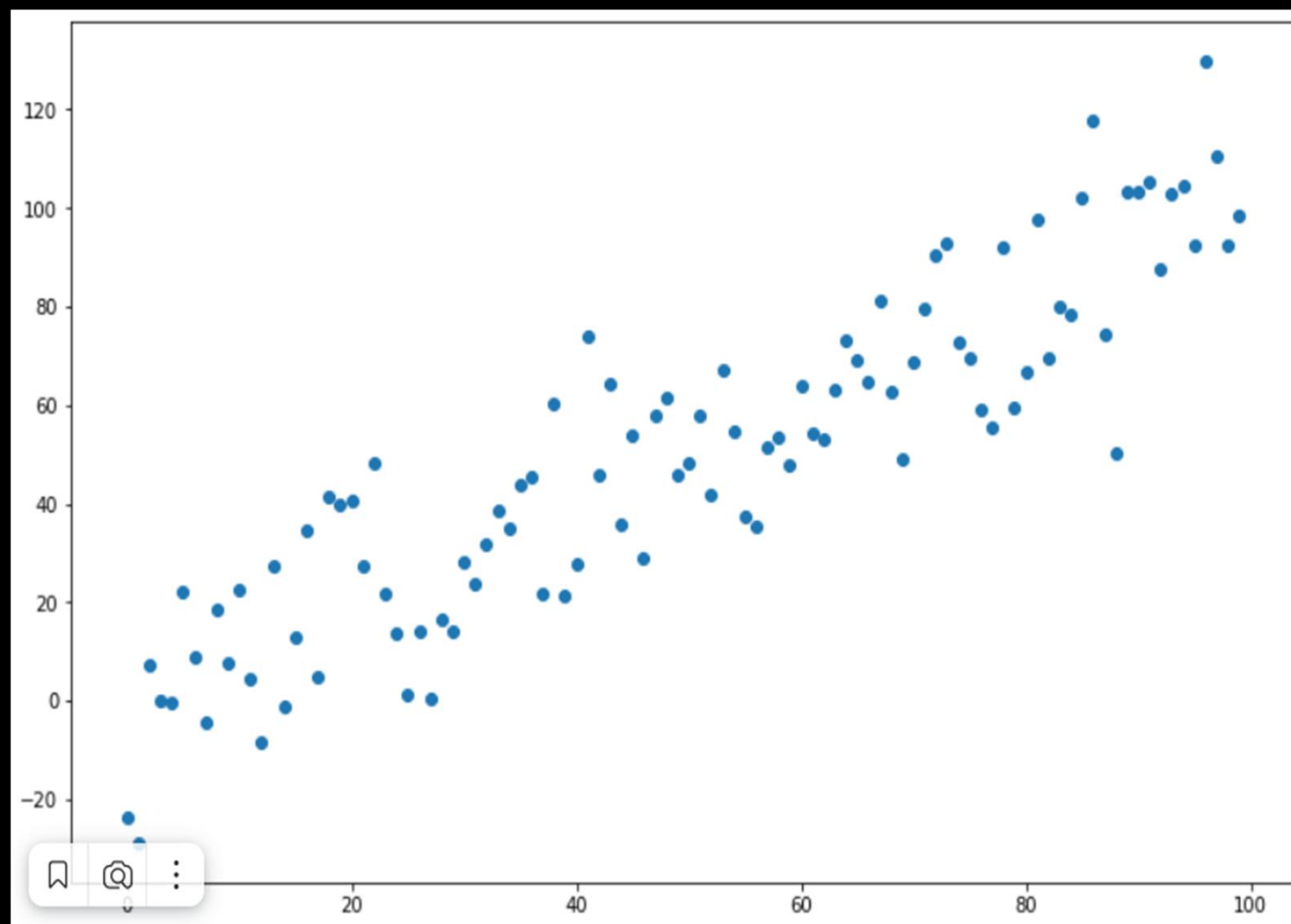
ТОЧЕЧНЫЙ ГРАФИК


Для отображения точечного графика предназначена функция `scatter()`. В простейшем виде точечный график можно получить, передав функции `scatter()` наборы точек для `x`, `y` координат:

```
>>> import numpy as np
>>> import matplotlib as mpl
>>> import matplotlib.pyplot as plt
>>> import pandas as pd
```

```
>>> fig = plt.figure(figsize = (12, 8))
>>> x = np.arange(100)
>>> y = x + np.random.normal(0, 15, size
= 100)
```

```
>>> plt.scatter(x,y)
```





Для более детальной настройки отображения необходимо воспользоваться дополнительными параметрами функции `scatter()`:

```
>>> scatter(x, y, s=None, c=None, marker=None, cmap=None, norm=None,  
vmin=None, vmax=None, alpha=None, linewidths=None, verts=None,  
edgecolors=None, *, plotnonfinite=False, data=None, **kwargs)
```


Параметр	Тип	Описание
x	array_like, shape (n,)	Набор данных для оси абсцисс
y	array_like, shape (n,)	Набор данных для оси ординат
s	scalar или array_like, shape (n,), optional	Масштаб точек
c	color, sequence, или sequence of color, optional	Цвет
marker	MarkerStyle, optional	Стиль точки объекта
cmap	Colormap, optional, default: None	Цветовая схема
norm	Normalize, optional, default: None	Нормализация данных
alpha	scalar, optional, default: None	Прозрачность
linewidths	scalar или array_like, optional, default: None	Ширина границы маркера
edgecolors	{'face', 'none', None} или color или sequence of color, optional.	Цвет границы

СТОЛБЧАТЫЕ ДИАГРАММЫ

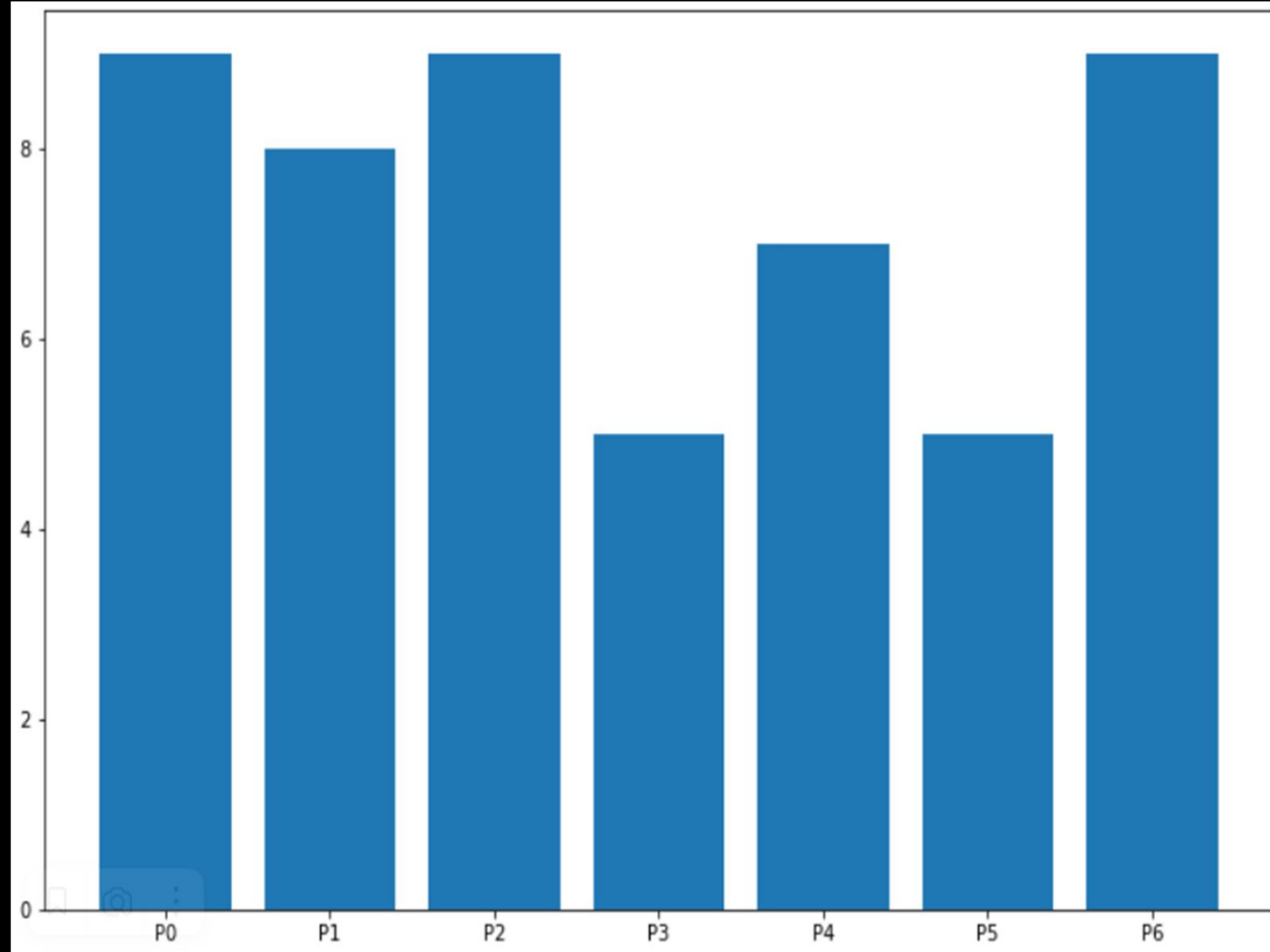
Для визуализации категориальных данных хорошо подходят столбчатые диаграммы. Для их построения используются функции:

`bar()` – для построения вертикальной диаграммы

`barh()` – для построения горизонтальной диаграммы.

Построим простую диаграмму:

```
>>> np.random.seed(123)
>>> groups = [f"P{i}" for i in range(7)]
>>> counts = np.random.randint(3, 10, len(groups))
>>> plt.bar(groups, counts)
```



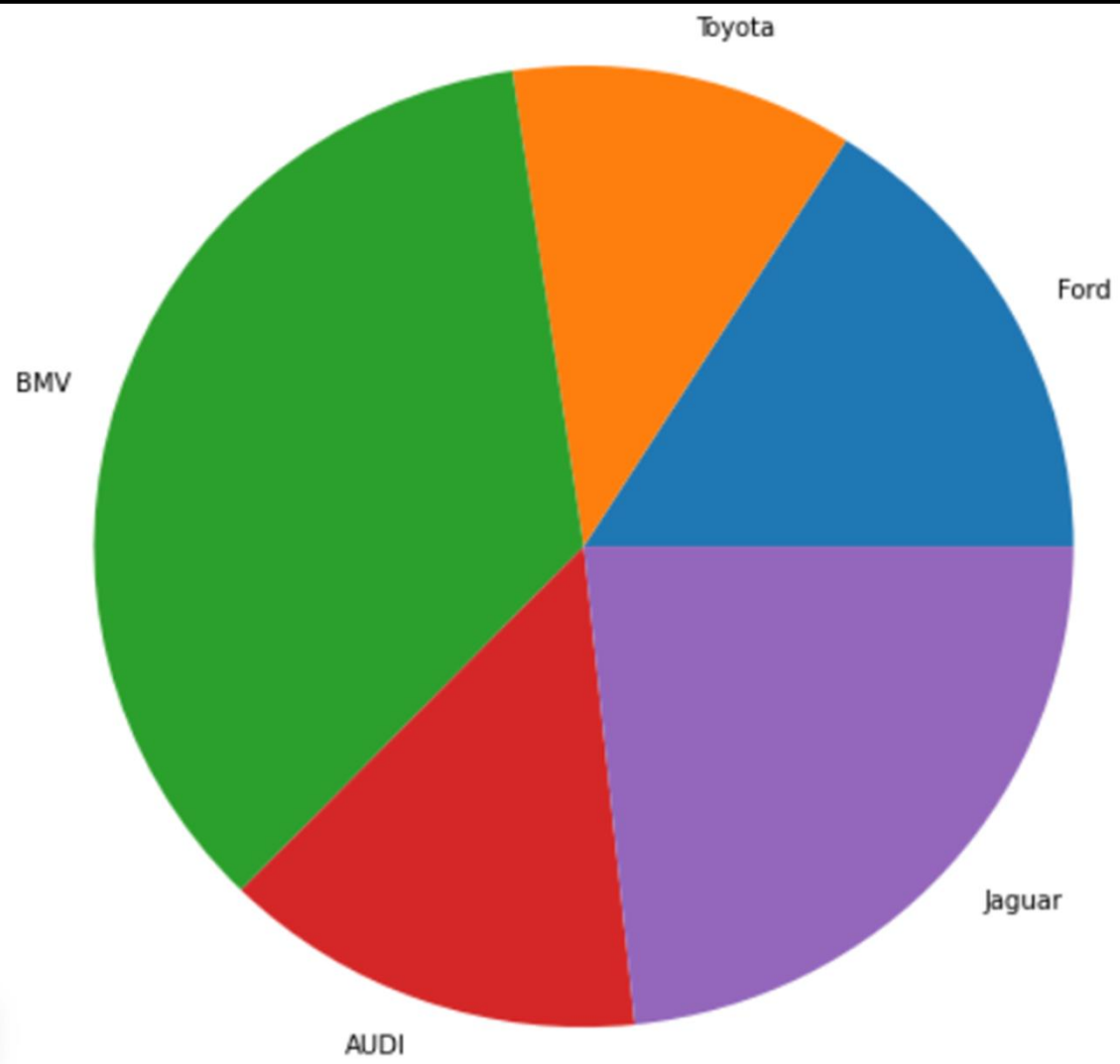
х: набор величин	х координаты столбцов
height : скалярная величина или набор величин	Высоты столбцов
width: скалярная величина, массив или optional	Ширина столбцов
bottom: скалярная величина, массив или optional	у координата базы
align : {'center', 'edge'}, optional, значение по умолчанию: 'center'	Выравнивание по координате х.

color: скалярная величина, массив или optional	Цвет столбцов диаграммы
edgecolor: скалярная величина, массив или optional	Цвет границы столбцов
linewidth: скалярная величина, массив или optional	Ширина границы
tick_label: str, массив или optional	Метки для столбца
xerr, yerr: скалярная величина, массив размера shape(N,) или shape(2,N) или optional	Величина ошибки для графика. Выставленное значение удаляется/прибавляется к верхней (правой – для горизонтального графика) границе. Может принимать следующие значения:
скаляр: симметрично +/- для всех баров	shape(N,): симметрично +/- для каждого бара
shape(2,N): выборочного – и + для каждого бара. Первая строка содержит нижние значения ошибок, вторая строка – верхние.	None: не отображать значения ошибок. Это значение используется по умолчанию.
ecolor: скалярная величина, массив или optional, значение по умолчанию: 'black'	Цвет линии ошибки.
log: bool, optional, значение по умолчанию: False	Включение логарифмического масштаба для оси y
orientation : {'vertical', 'horizontal'}, optional	Ориентация: вертикальная или горизонтальная.

КРУГОВЫЕ ДИАГРАММЫ

Круговые диаграммы – это наглядный способ показать доли компонент в наборе. Они идеально подходят для отчетов, презентаций и т.п. Для построения круговых диаграмм в Matplotlib используется функция `pie()`:

```
>>> vals = [24, 17, 53, 21, 35]
>>> labels = ["Ford", "Toyota", "BMV", "AUDI", "Jaguar"]
>>> fig, ax = plt.subplots()
>>> ax.pie(vals, labels=labels)
>>> ax.axis("equal")
```



х: массив	Массив с размерами долей.
explode: массив, optional, значение по умолчанию:None	Если параметр не равен None, то часть долей, который перечислены в передаваемом значении будут вынесены из диаграммы на заданное расстояние
labels: list, optional, значение по умолчанию:None	Текстовые метки долей.
colors: массив, optional, значение по умолчанию: None	Цвета долей.
autorct: str, функция, optional, значение по умолчанию: None	Формат текстовой метки внутри доли, текст – это численное значение показателя, связанного с конкретной долей.
pctdistance: float, optional, значение по умолчанию: 0.6	Расстояние между центром каждой доли и началом текстовой метки, которая определяется параметром autorct.
shadow: bool, optional, значение по умолчанию:False	Отображение тени для диаграммы.
labeldistance: float, None, optional, значение по умолчанию: 1.1	Расстояние, на котором будут отображены текстовые метки долей. Если параметр равен None, то метки не будут отображены.
startangle: float, optional, значение по умолчанию:None	Задаёт угол, на который нужно повернуть диаграмму против часовой стрелке относительно оси х.
radius: float, optional, значение по умолчанию:None	Величина радиуса диаграммы.
counterclock: bool, optional, значение по умолчанию:True	Определяет направление вращения – по часовой или против часовой стрелки.
wedgeprops: dict, optional, значение по умолчанию:None	Словарь параметров, определяющих внешний вид долей.
textprops: dict, optional, значение по умолчанию:None	Словарь параметров определяющих внешний вид текстовых меток.
center: list значений float, optional, значение по умолчанию: (0, 0)	Центр диаграммы.
frame: bool, optional, значение по умолчанию:False	Если параметр равен True, то вокруг диаграммы будет отображена рамка.
rotatelabels: bool, optional, значение по умолчанию:False	Если параметр равен True, то текстовые метки будут повернуты на угол.

ГИСТОГРАММЫ

Гистограммы – незаменимый инструмент в статистических исследованиях.
В наиболее простом случае для создания гистограммы понадобится такой код:

```
>>> import numpy as np
>>> import matplotlib.mlab as mlab
>>> import matplotlib.pyplot as plt

>>> fig = plt.figure(figsize=(12,8))
>>> x = np.random.normal(15, 3, size = 100)
>>> num_bins = 5
>>> plt.hist(x, num_bins)
```

