Практическая работа №1. Построение графиков в Python

Существует два основных варианта установки этой библиотеки: в первом случае вы устанавливаете пакет Anaconda, в состав которого входит большое количество различных инструментов для работы в области машинного обучения и анализа данных (и не только); во втором – установить Matplotlib самостоятельно, используя менеджер пакетов.

Проверка установки

Для проверки того, что все у вас установилось правильно, запустите интерпретатор *Python* и введите в нем следующее:

```
>>> import matplotlib
```

После этого можете проверить версию библиотеки (она скорее всего будет отличаться от приведенной ниже):

```
>>> matplotlib.__version__
'3.0.3'
```

Быстрый старт

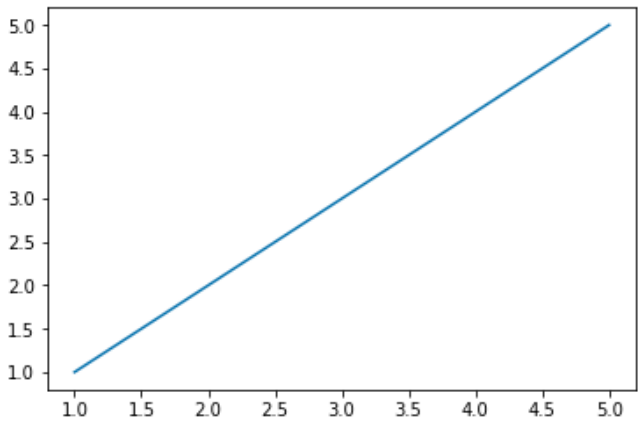
Перед тем как углубиться в дебри библиотеки *Matplotlib*, для того, чтобы появилось интуитивное понимание принципов работы с этим инструментом, рассмотрим несколько примеров, изучив которые вы уже сможете использовать библиотеку для решения своих задач.

Если вы работаете в *Jupyter Notebook* для того, чтобы получать графики рядом с ячейками с кодом необходимо выполнить специальную *magic* команду после того, как импортируете *matplotlib*:

```
import matplotlib.pyplot as plt
%matplotlib inline
```

Результат работы выглядеть будет так, как показано на рисунке ниже.

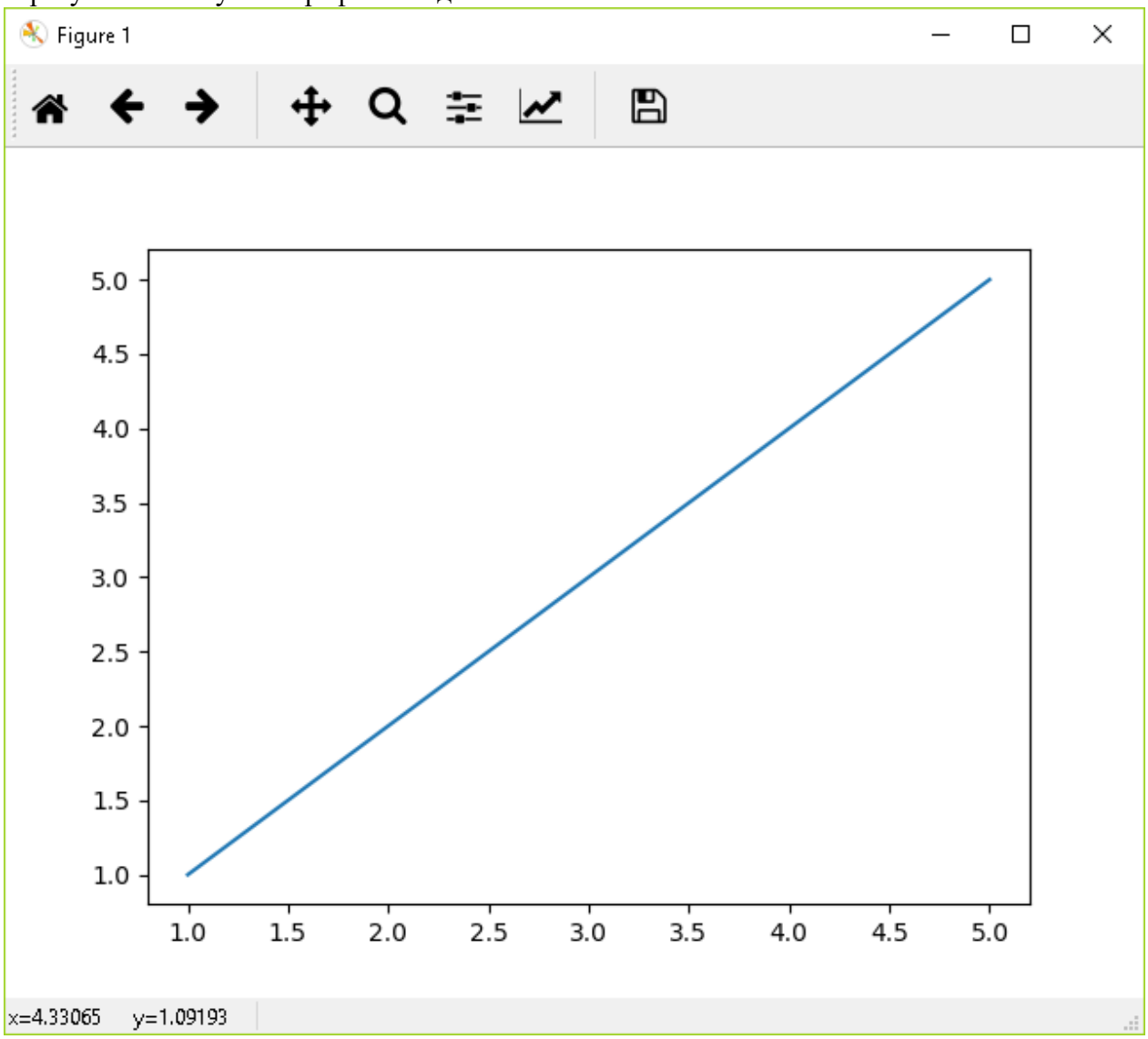
```
In [1]: import matplotlib.pyplot as plt
        %matplotlib inline

In [2]: plt.plot([1, 2, 3, 4, 5], [1, 2, 3, 4, 5])
Out[2]: [<matplotlib.lines.Line2D at 0x1a4f1dadd30>]
```



Если вы пишете код в *.py* файле, а потом запускаете его через вызов интерпретатора *Python*, то строка %matplotlib inline вам не нужна, используйте только импорт библиотеки.
Пример, аналогичный тому, что представлен на рисунке выше, для отдельного *Python* файла будет выглядеть так:

```
import matplotlib.pyplot as plt
plt.plot([1, 2, 3, 4, 5], [1, 2, 3, 4, 5])
plt.show()
```

В результате получите график в отдельном окне.



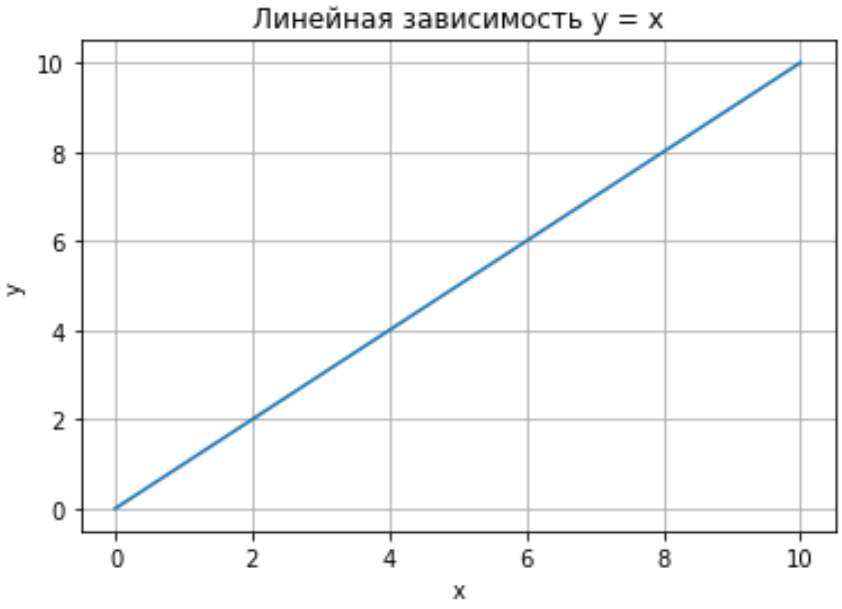
Далее мы не будем останавливаться на особенностях использования magic команды, просто запомните, если вы используете Jupyter notebook при работе с Matplotlib вам обязательно нужно включить %matplotlib inline.
Теперь перейдем непосредственно к Matplotlib. Задача урока –построить разные типы графиков, настроить их внешний вид и освоиться в работе с этим инструментом.

Построение графика

Для начал построим простую линейную зависимость, дадим нашему графику название, подпишем оси и отобразим сетку. Код программы:

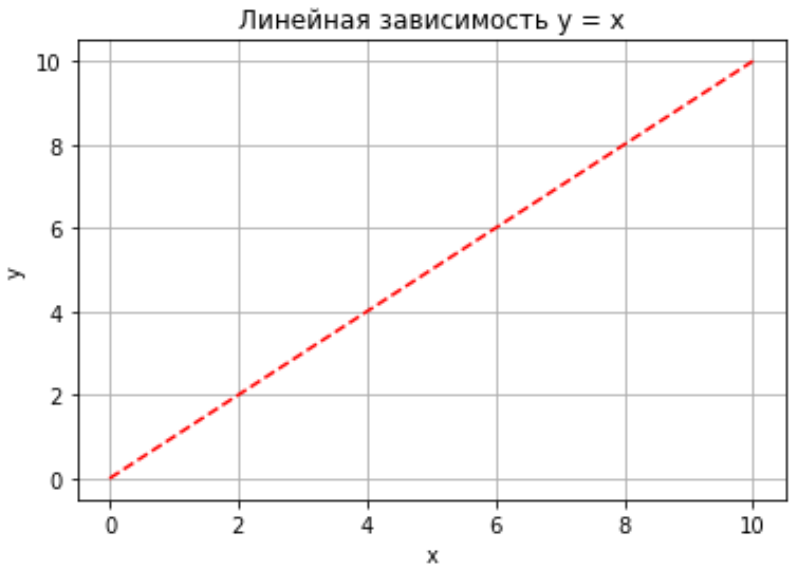
```
import numpy as np
# Независимая (x) и зависимая (y) переменные
x = np.linspace(0, 10, 50)
y = x

# Построение графика
plt.title("Линейная зависимость y = x") # заголовок
plt.xlabel("x") # ось абсцисс
plt.ylabel("y") # ось ординат
plt.grid() # включение отображение сетки
plt.plot(x, y) # построение графика
```

В результате получим следующий график:



Изменим тип линии и ее цвет, для этого в функцию plot(), в качестве третьего параметра передадим строку, сформированную определенным образом, в нашем случае это “r–”, где “r” означает красный цвет, а “–” – тип линии – пунктирная линия.

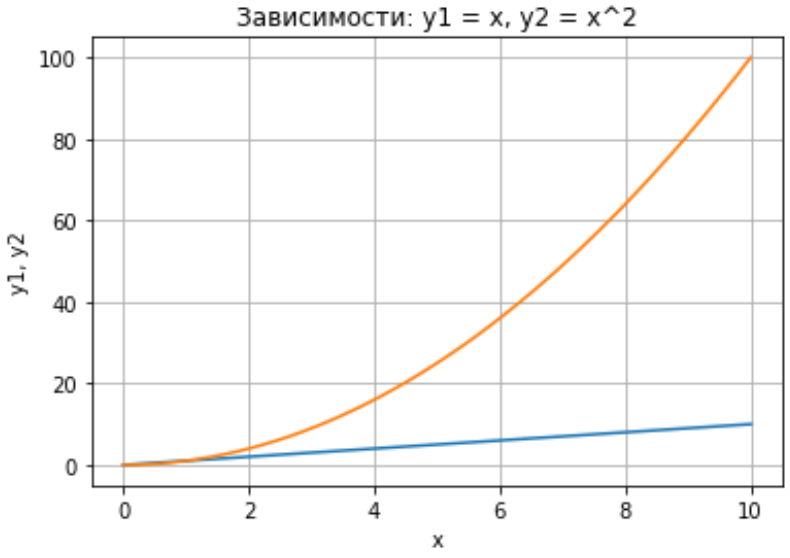
```
# Построение графика
plt.title("Линейная зависимость y = x") # заголовок
plt.xlabel("x") # ось абсцисс
plt.ylabel("y") # ось ординат
plt.grid() # включение отображение сетки
plt.plot(x, y, "r--") # построение графика
```



Несколько графиков на одном поле

Построим несколько графиков на одном поле, для этого добавим квадратичную зависимость:

```
# Линейная зависимость
x = np.linspace(0, 10, 50)
y1 = x
# Квадратичная зависимость
y2 = [i**2 for i in x]
# Построение графика
plt.title("Зависимости: y1 = x, y2 = x^2") # заголовок
plt.xlabel("x") # ось абсцисс
plt.ylabel("y1, y2") # ось ординат
plt.grid() # включение отображение сетки
plt.plot(x, y1, x, y2) # построение графика
```



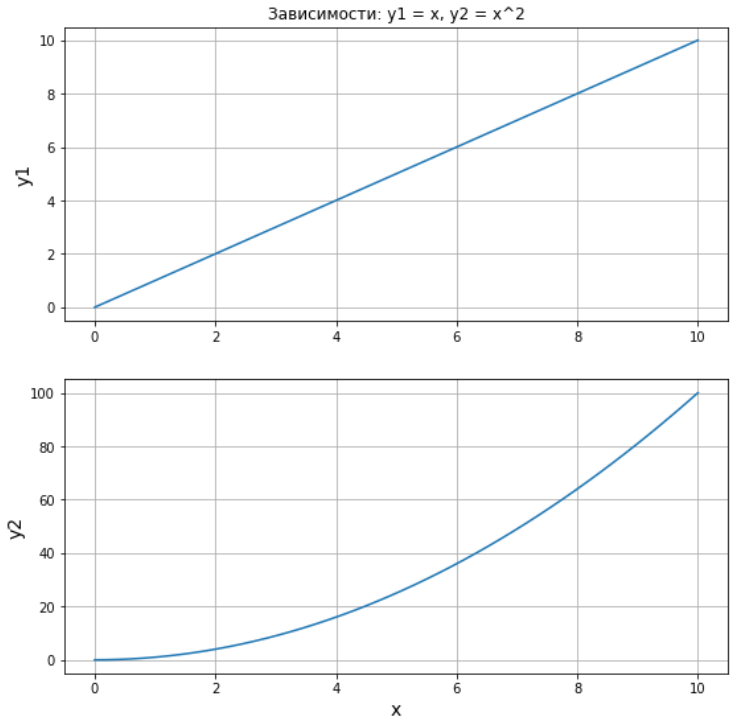
В приведенном примере в функцию **plot()** последовательно передаются два массива для построения первого графика и два массива для построения второго, при этом, как вы можете заметить, для обоих графиков массив значений независимой переменной x один и то же.

Несколько разделенных полей с графиками

Третья, довольно часто встречающаяся задача – это отобразить два или более различных поля, на которых будет отображено по одному или более графику.

Построим уже известные нам две зависимость на разных полях.

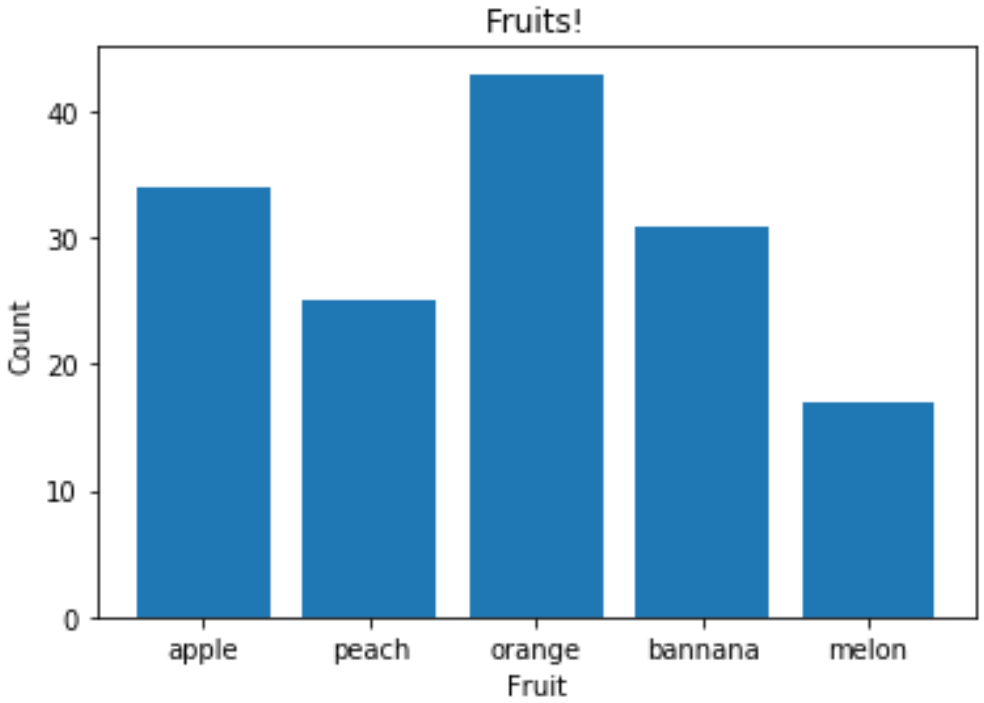
```
# Линейная зависимость
x = np.linspace(0, 10, 50)
y1 = x
# Квадратичная зависимость
y2 = [i**2 for i in x]
# Построение графиков
plt.figure(figsize=(9, 9))
plt.subplot(2, 1, 1)
plt.plot(x, y1) # построение графика
plt.title("Зависимости: y1 = x, y2 = x^2") # заголовок
plt.ylabel("y1", fontsize=14) # ось ординат
plt.grid(True) # включение отображение сетки
plt.subplot(2, 1, 2)
plt.plot(x, y2) # построение графика
plt.xlabel("x", fontsize=14) # ось абсцисс
plt.ylabel("y2", fontsize=14) # ось ординат
plt.grid(True) # включение отображение сетки
```



Здесь мы воспользовались новыми функциями:
figure() – функция для задания глобальных параметров отображения графиков. В нее, в качестве аргумента, мы передаем кортеж, определяющий размер общего поля.
subplot() – функция для задания местоположения поля с графиком. Существует несколько способов задания областей для вывода через функцию subplot() мы воспользовались следующим: первый аргумент – количество строк, второй – столбцов в формируемом поле, третий – индекс (номер поля, считаем сверху вниз, слева направо).
Остальные функции уже вам знакомы, дополнительно мы использовали параметр *fontsize* для функций *xlabel()* и *ylabel()*, для задания размера шрифта.

Построение диаграммы для категориальных данных
До этого мы строили графики по численным данным, т.е. зависимая и независимая переменные имели числовой тип. На практике довольно часто приходится работать с данными нечисловой природы – имена людей, название фруктов, и т.п.
Построим диаграмму на которой будет отображаться количество фруктов в магазине:

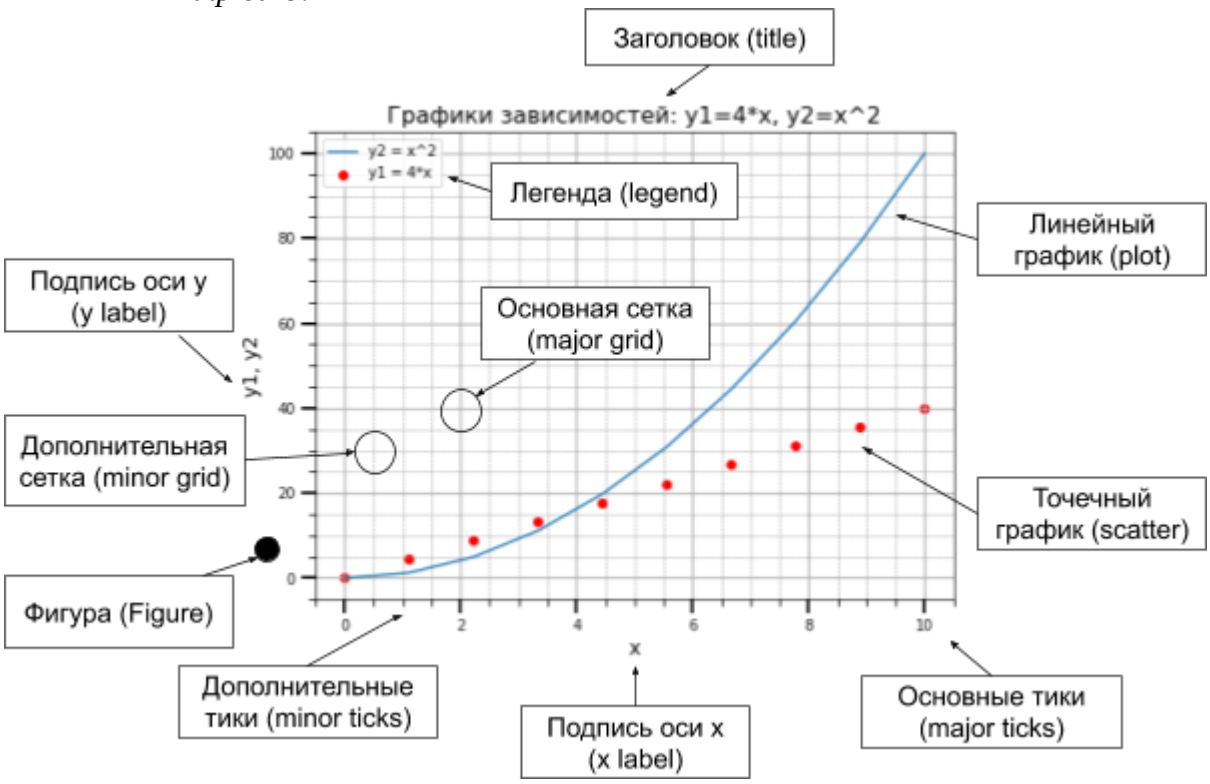
```
fruits = ["apple", "peach", "orange", "bannana", "melon"]
counts = [34, 25, 43, 31, 17]
plt.bar(fruits, counts)
plt.title("Fruits!")
plt.xlabel("Fruit")
plt.ylabel("Count")
```



Для вывода диаграммы мы использовали функцию bar().
К этому моменту, если вы самостоятельно попробовали запустить приведенные выше примеры, у вас уже должно сформировать некоторое понимание того, как осуществляется работа с этой библиотекой.

Основные элементы графика

Рассмотрим основные термины и понятия, касающиеся изображения графика, с которыми вам необходимо будет познакомиться, для того, чтобы в дальнейшем у вас не было трудностей при прочтении материалов из этого цикла статей и документации по библиотеке *matplotlib*.



Корневым элементом при построения графиков в системе *Matplotlib* является Фигура (*Figure*). Все, что нарисовано на рисунке выше является элементами фигуры. Рассмотрим ее составляющие более подробно.

График

На рисунке представлены два графика – линейный и точечный. *Matplotlib* предоставляет огромное количество различных настроек, которые можно использовать для того, чтобы придать графику вид, который вам нужен: цвет, толщина и тип линии, стиль линии и многое другое, все это мы рассмотрим в ближайших статьях.

Оси

Вторым, после непосредственно самого графика, по важности элементом фигуры являются оси. Для каждой оси можно задать метку (подпись), основные (*major*) и дополнительные (*minor*) тики, их подписи, размер и толщину, также можно задать диапазоны по каждой из осей.

Сетка и легенда

Следующими элементами фигуры, которые значительно повышают информативность графика являются сетка и легенда. Сетка также может быть основной (*major*) и дополнительной (*minor*). Каждому типу сетки можно задавать цвет, толщину линии и тип. Для отображения сетки и легенды используются соответствующие команды.

Ниже представлен код, с помощью которого была построена фигура, изображенная на рисунке:

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

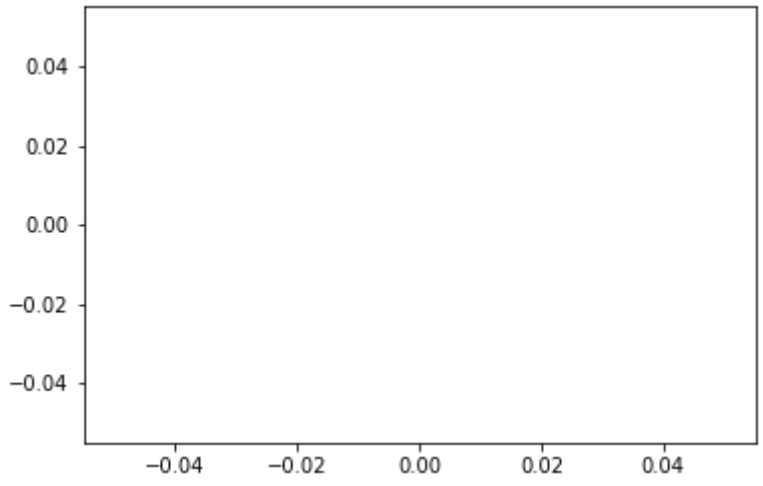
```
from matplotlib.ticker import (MultipleLocator, FormatStrFormatter,
AutoMinorLocator)
import numpy as np
x = np.linspace(0, 10, 10)
y1 = 4*x
y2 = [i**2 for i in x]
fig, ax = plt.subplots(figsize=(8, 6))
ax.set_title("Графики зависимостей: y1=4*x, y2=x^2", fontsize=16)
ax.set_xlabel("x", fontsize=14)
ax.set_ylabel("y1, y2", fontsize=14)
ax.grid(which="major", linewidth=1.2)
ax.grid(which="minor", linestyle="--", color="gray", linewidth=0.5)
ax.scatter(x, y1, c="red", label="y1 = 4*x")
ax.plot(x, y2, label="y2 = x^2")
ax.legend()
ax.xaxis.set_minor_locator(AutoMinorLocator())
ax.yaxis.set_minor_locator(AutoMinorLocator())
ax.tick_params(which='major', length=10, width=2)
ax.tick_params(which='minor', length=5, width=1)
plt.show()
```

Построение графиков

Основным элементом изображения, которое строит *pyplot* является Фигура (Figure), на нее накладываются один или более графиков, осей, надписей и т.п. Для построения графика используется команда *plot()*. В самом минимальном варианте можно ее использовать без параметров:

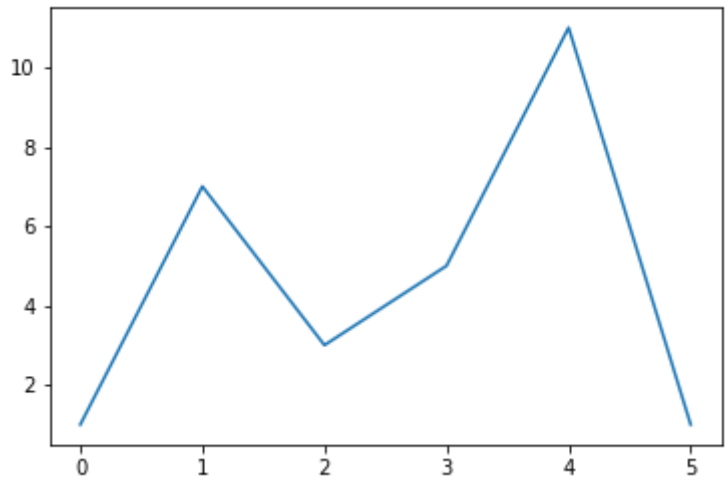
```
import matplotlib.pyplot as plt
%matplotlib inline
plt.plot()
```

В результате будет выведено пустое поле:



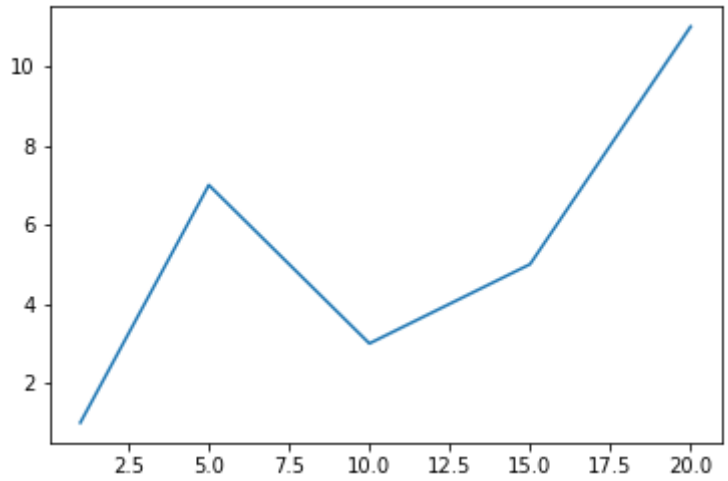
Далее команду импорта и *magic*-команду для *Jupyter* (первая и вторая строки приведенной выше программы) мы использовать не будем.
Если в качестве параметра функции *plot()* передать список, то значения из этого списка будут отложены по оси ординат (ось *y*), а по оси абсцисс (ось *x*) будут отложены индексы элементов массива:

```
plt.plot([1, 7, 3, 5, 11, 1])
```



Для того, чтобы задать значения по осям *x* и *y* необходимо в *plot()* передать два списка:

```
plt.plot([1, 5, 10, 15, 20], [1, 7, 3, 5, 11])
```



Текстовые надписи на графике

Наиболее часто используемые текстовые надписи на графике это:

- наименование осей;
- наименование самого графика;
- текстовое примечание на поле с графиком;
- легенда.

Рассмотрим кратко данные элементы, более подробный рассказ о них будет в одном из ближайших уроков.

Наименование осей

Для задания подписи оси *x* используется функция *xlabel()*, оси *y* – *ylabel()*. Разберемся с аргументами данных функций. Здесь и далее аргументы будем описывать следующим образом:

- **Имя_аргумента: тип(ы)**
  - **Описание**

Для функций *xlabel()/ylabel()* основными являются следующие аргументы:

- *xlabel* (или *ylabel*):str
  - Текст подписи.
- *labelpad :* численное значение либо *None*; значение по умолчанию: *None*
  - Расстояние между областью графика, включающую оси, и меткой.

Функции xlabel()/ylabel() принимают в качестве аргументов параметры конструктора класса *matplotlib.text.Text*, некоторые из них нам могут пригодиться:

- *fontsize* или *size:* число либо значение из списка: {*'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', 'xx-large'*}.
  - Размер шрифта.
- *fontstyle*: значение из списка: {*'normal', 'italic', 'oblique'*}.
  - Стиль шрифта.
- *fontweight*: число в диапазоне от 0 до 1000 либо значение из списка: {*'ultralight', 'light', 'normal', 'regular', 'book', 'medium', 'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy', 'extra bold', 'black'*}.
  - Толщина шрифта.
- *color:* одни из доступных способов определения цвета см. Цвет линии.
  - Цвет шрифта.

Пример использования:

```
plt.xlabel('Day', fontsize=15, color='blue')
```

Аргументов у этих функций довольно много и они позволяют достаточно тонко настроить внешний вид надписей. В рамках этого урока мы только начинаем знакомиться с инструментом *pyplot* поэтому не будем приводить весь список.

Заголовок графика

Для задания заголовка графика используется функция *title()*:

```
plt.title('Chart price', fontsize=17)
```

Из параметров отметим следующие:

- *label*: *str*
  - Текст заголовка.
- *loc:* значение из набора {'*center*', '*left*', '*right*'}
  - Выравнивание заголовка.

Для функции *title()* также доступны параметры конструктора класса *matplotlib.text.Text*, часть из них представлена в описании аргументов функций xlabel() / ylabel().

Текстовое примечание

За размещение текста на поле графика отвечает функция *text()*, которой вначале передаются координаты позиции надписи, после этого – текст самой надписи.

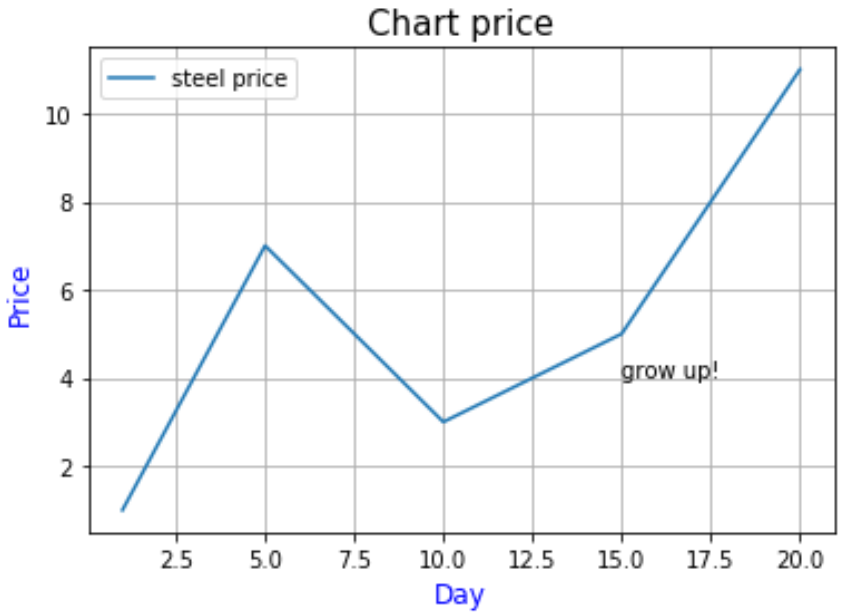
```
plt.text(1, 1, 'type: Steel')
```

Легенда

Легенда будет размещена на графике, если вызвать функцию *legend()*, в рамках данного урока мы не будем рассматривать аргументы этой функции.

Разместим на уже знакомом нам графике необходимый набор подписей.

```
x = [1, 5, 10, 15, 20]
y = [1, 7, 3, 5, 11]
plt.plot(x, y, label='steel price')
plt.title('Chart price', fontsize=15)
plt.xlabel('Day', fontsize=12, color='blue')
plt.ylabel('Price', fontsize=12, color='blue')
plt.legend()
plt.grid(True)
plt.text(15, 4, 'grow up!')
```



К перечисленным опциям мы добавили сетку, которая включается с помощью функции grid(True).

Работа с линейным графиком

Matplotlib предоставляет огромное количество инструментов для построения различных видов графиков. Так как наиболее часто встречающийся вид графика – это линейный, ему и уделим внимание. Необходимо помнить, что настройка графиков других видов, будет осуществляться сходным образом.

Параметры, которые отвечают за отображение графика можно задать непосредственно в самой функции plot():

```
plt.plot(x, y, color='red')
```

Либо воспользоваться функцией setp(), через которую можно модифицировать нужные параметры:

```
plt.setp( color='red', linewidth=1)
```

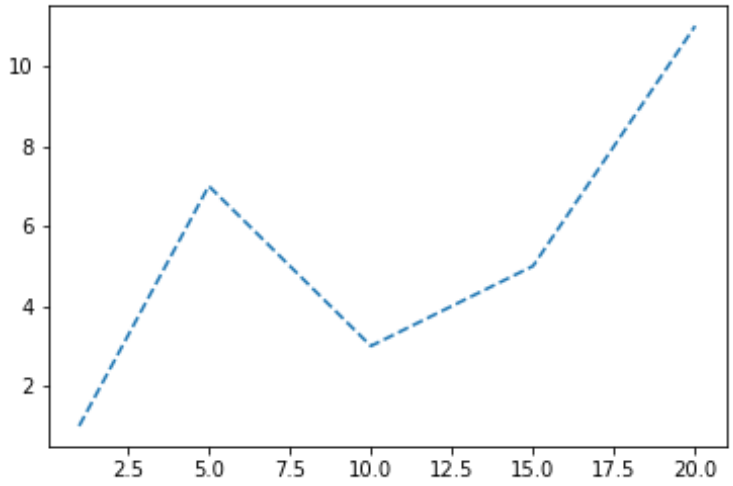
Стиль линии графика

Стиль линии графика задается через параметр *linestyle*, который может принимать значения из приведенной ниже таблицы.

Значение параметра	Описание
'-' или *'solid'*	Непрерывная линия
'–' или *'dashed'*	Штриховая линия
'-.' или *'dashdot'*	Штрихпунктирная линия
':' или *'dotted'*	Пунктирная линия
'None' или ' ' или ''	Не отображать линию

Стиль линии можно передать сразу после указания списков с координатами без указания, что это параметр *linewidth*.

```
x = [1, 5, 10, 15, 20]
y = [1, 7, 3, 5, 11]
plt.plot(x, y, '--')
```



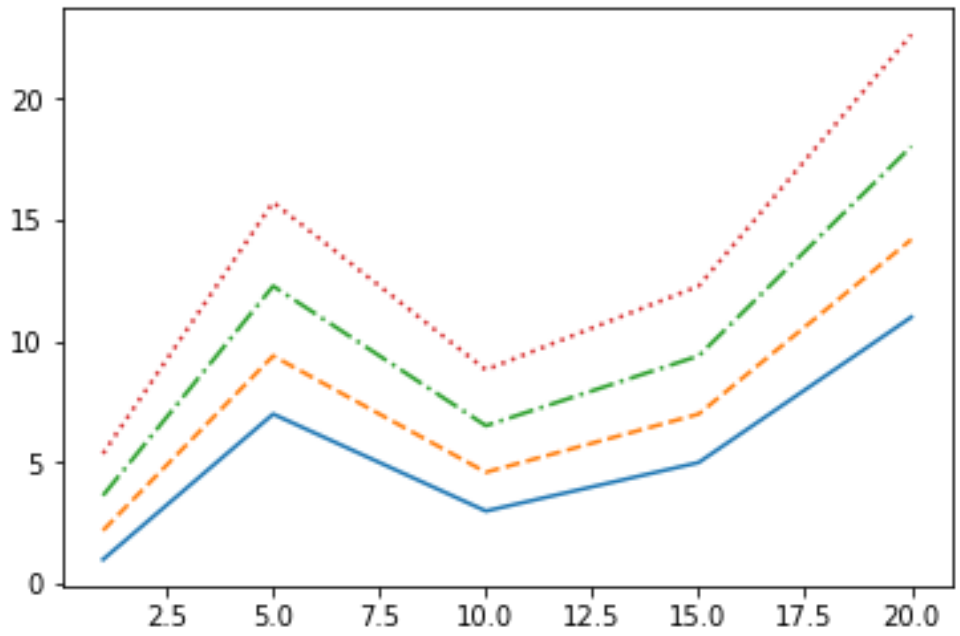
Либо можно воспользоваться функцией setp():

```
x = [1, 5, 10, 15, 20]
y = [1, 7, 3, 5, 11]
line = plt.plot(x, y)
plt.setp(line, linestyle='--')
```

Результат будет тот же, что на рисунке выше.

Для того, чтобы вывести несколько графиков на одном поле необходимо передать соответствующие наборы значений в функцию plot(). Построим несколько наборов данных и выведем их с использованием различных стилей линии:

```
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.plot(x, y1, '-', x, y2, '--', x, y3, '-.', x, y4, ':')
```



Тот же результат можно получить, вызвав plot() для построения каждого графика по отдельности. Если вы хотите представить каждый график отдельно на своем поле, то используйте для этого subplot()

```
plt.plot(x, y1, '-')
plt.plot(x, y2, '--')
plt.plot(x, y3, '-.')
plt.plot(x, y4, ':')
```

Цвет линии

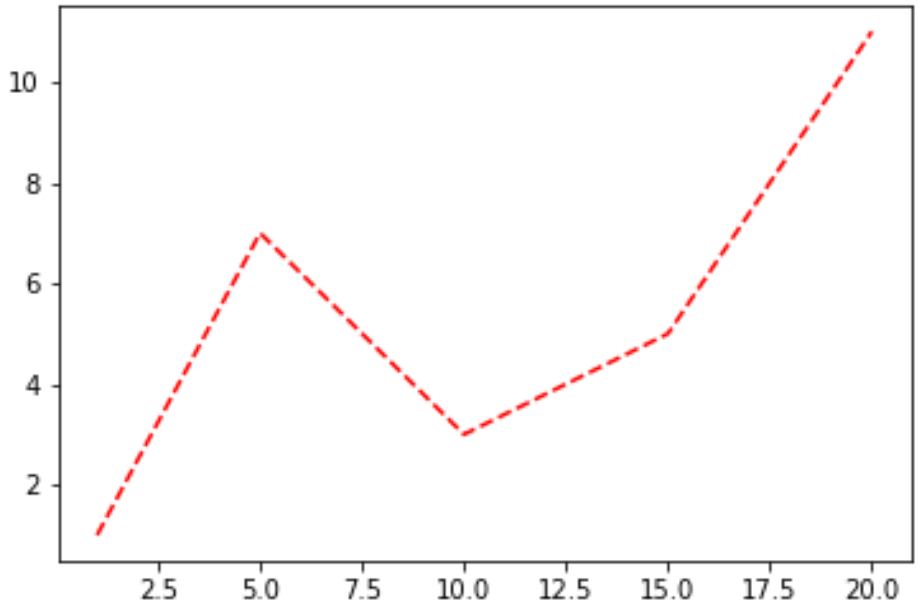
Задание цвета линии графика производится через параметр color (или c, если использовать сокращенный вариант). Значение может быть представлено в одном из следующих форматов:

- *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’)
- символ из набора {*‘b’, ‘g’, ‘r’, ‘c’, ‘m’, ‘y’, ‘k’, ‘w’*}
- 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*:’

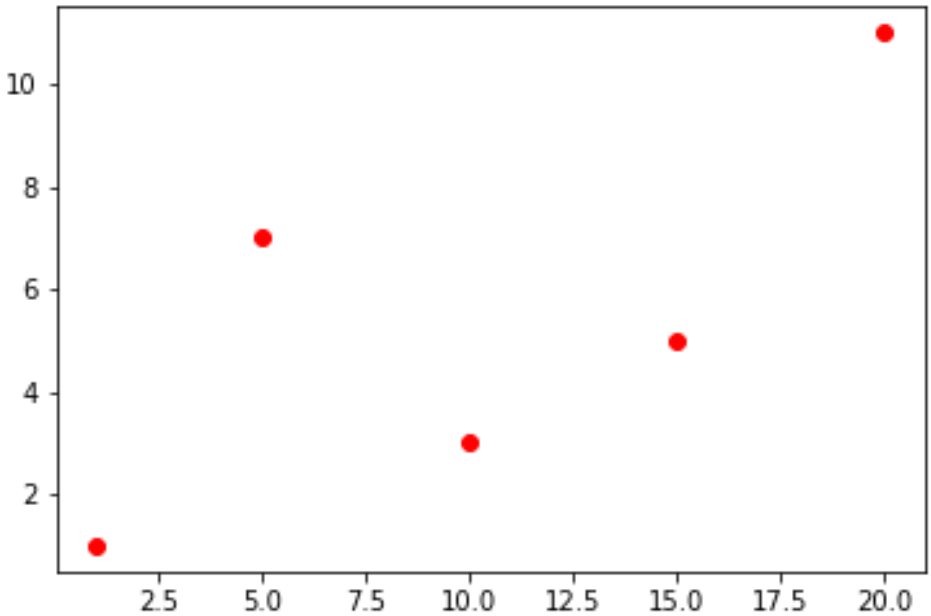
Если цвет задается с помощью символа из набора {*‘b’, ‘g’, ‘r’, ‘c’, ‘m’, ‘y’, ‘k’, ‘w’*}, то он может быть совмещен со стилем линии в рамках параметра *fmt* функции *plot()*. Например штриховая красная линия будет задаваться так: ‘–r’, а штрих пунктирная зеленая так ‘-.g’

```
x = [1, 5, 10, 15, 20]
y = [1, 7, 3, 5, 11]
```

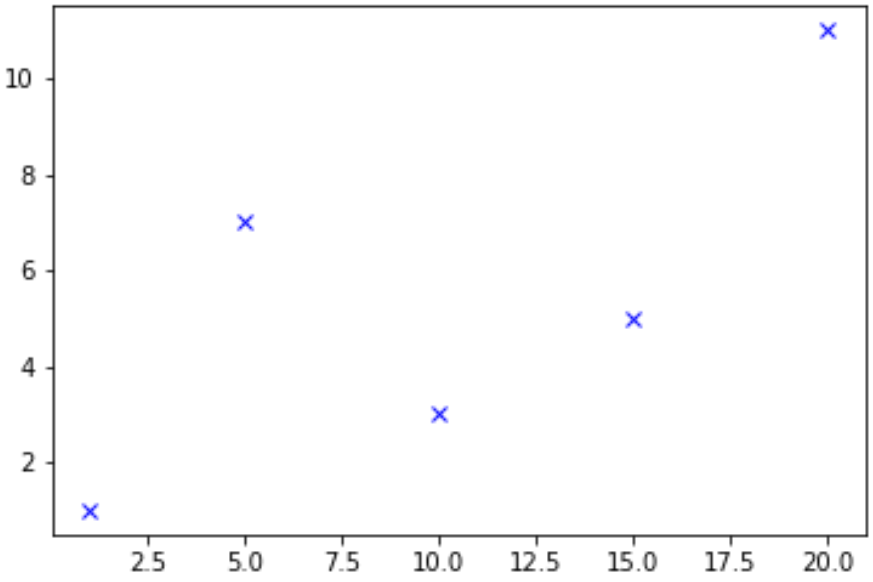
plt.plot(x, y, '--r')



Тип графика
До этого момента мы работали только с линейными графиками,
функция *plot()* позволяет задать тип графика: линейный либо точечный, при этом для точечного графика можно указать соответствующий маркер. Приведем пару примеров:
plt.plot(x, y, 'ro')



plt.plot(x, y, 'bx')



Размер маркера можно менять, об этом более подробно будет рассмотрено в уроке, посвященном точечным графикам.

Размещение графиков на разных полях

Существуют три основных подхода к размещению нескольких графиков 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.

В этом уроке будут рассмотрены первые два подхода.

Работа с функцией subplot()

Самый простой способ представить графики в отдельных полях – это использовать функцию *supplot()* для задания их мест размещения. До этого момента мы не работали с Фигурой (*Figure*) напрямую, значения ее параметров, задаваемые по умолчанию, нас устраивали. Для решения текущей задачи придется один из параметров – размер подложки, задать вручную. За это отвечает аргумент *figsize* функции *figure()*, которому присваивается кортеж из двух *float* элементов, определяющих высоту и ширину подложки.

После задания размера, указывается местоположение, куда будет установлено поле с графиком с помощью функции subplot(). Чаще всего используют следующие варианты вызова subplot:

subplot(nrows, ncols, index)

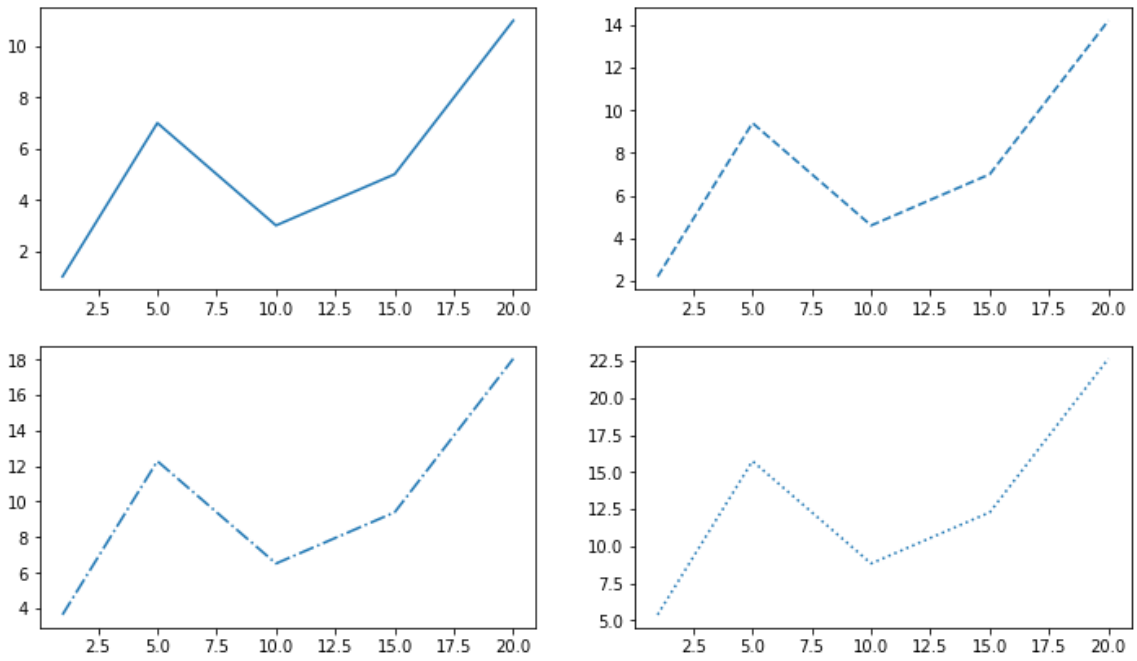
- *nrows: int*
  - Количество строк.
- *ncols: int*
  - Количество столбцов.
- *index: int*
  - Местоположение элемента.

subplot(pos)

- *pos:int*
  - 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, в ином случае лучше обратиться к первому варианту.

Рассмотрим на примере работу с данными функциями:

```
# Исходный набор данных
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]
# Настройка размеров подложки
plt.figure(figsize=(12, 7))
# Вывод графиков
plt.subplot(2, 2, 1)
plt.plot(x, y1, '-')
plt.subplot(2, 2, 2)
plt.plot(x, y2, '--')
plt.subplot(2, 2, 3)
plt.plot(x, y3, '-.')
plt.subplot(2, 2, 4)
plt.plot(x, y4, ':')
```



Второй вариант использования *subplot()*:

```
# Вывод графиков
plt.subplot(221)
plt.plot(x, y1, '-')
plt.subplot(222)
plt.plot(x, y2, '--')
plt.subplot(223)
plt.plot(x, y3, '-.')
plt.subplot(224)
plt.plot(x, y4, ':')
```

Работа с функцией subplots()

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 полных доступ к настройке внешнего вида отображаемых элементов.

Решим задачу вывода четырех графиков с помощью функции *subplots()*:

```
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, ':')
```

Результат будет аналогичный тому, что приведен в разделе “Работа с функцией subplot()”.