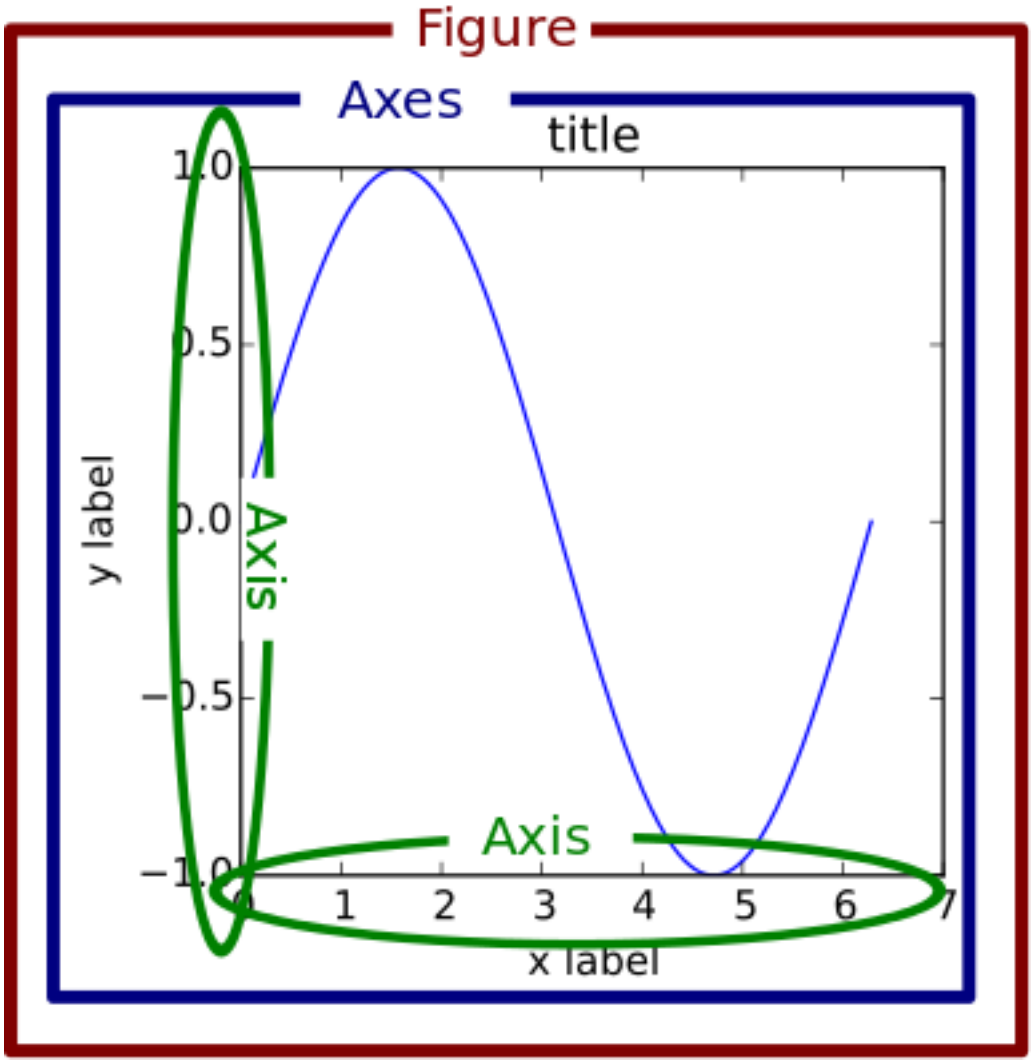
Построение графиков

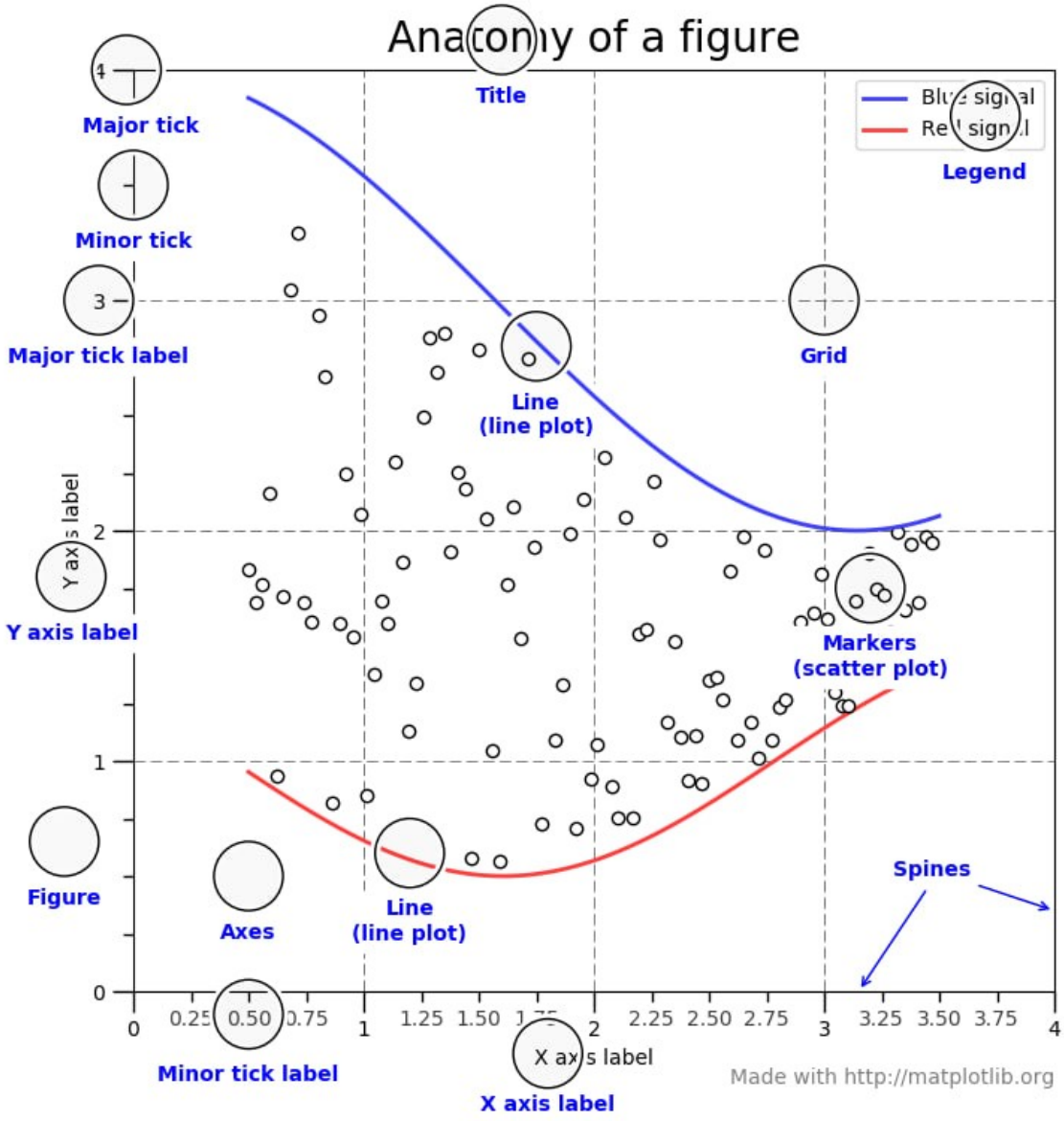
Matplotlib — одна из старейших научных библиотек визуализации и построения графиков, доступных на Python. С помощью Matplotlib можно создать практически любую двумерную визуализацию. В обширной галерее примеров показано множество изображений, которые можно создать с помощью Matplotlib.

График Matplotlib содержит

- Один или несколько объектов `Axes`, каждый из которых является отдельным подрисунком.
- Объект `Figure`, который представляет собой итоговое изображение, содержащее один или несколько `Axes`.



Картинка из Matplotlib FAQ чтобы понять (или хотя бы посмотреть), из чего состоит рисунок в matplotlib и какие его части можно кастомизировать.



Примеры

Перед решением заданий давайте разберем несколько примеров.

```
# импорт библиотек matplotlib, numpy, pandas (понадобится для
считывания данных)
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
```

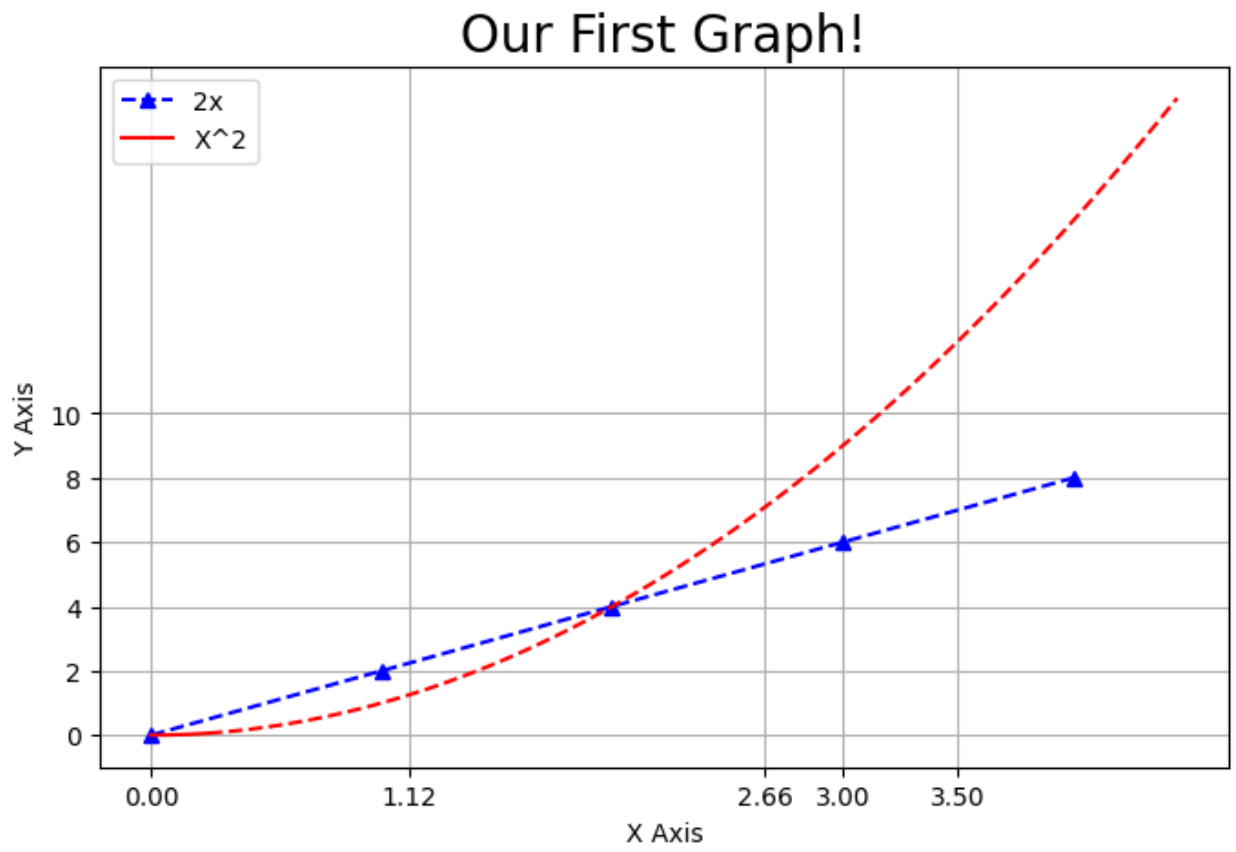
Простой график

Самым простым способом является интерактивный режим работы с графиком, в котором мы просим вызываем функции библиотеки для того, чтобы совершить какие-либо действия. В дальнейшем будет приведен пример работы с графиком как с объектом, что является более правильной методикой, когда нужно строить сложные графики в большом количестве.

Обратите внимание, что если вы работаете не в Jupyter Notebook, после выполнения функции `plt.show()` у вас появится новое окно с графиком. Если в запущенном файле вы пытались построить несколько графиков на разных `Figure`, то для отображения следующего графика вам надо закрыть предыдущий.

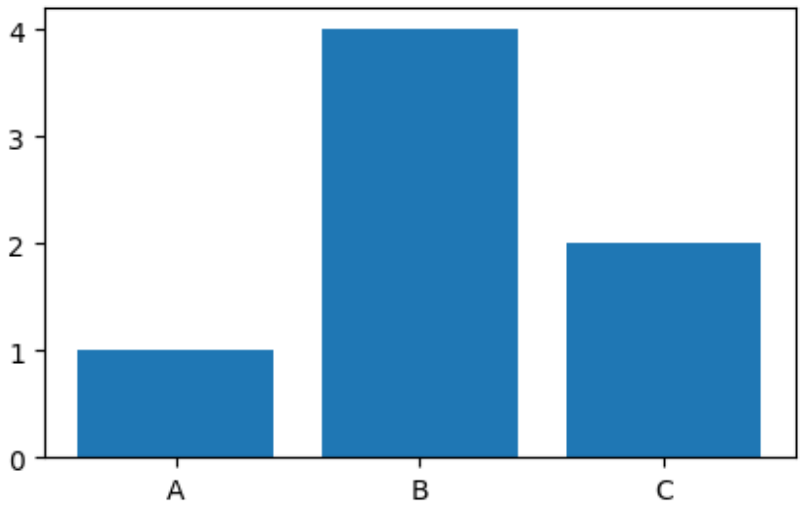
```
x = [0,1,2,3,4]
y = [0,2,4,6,8]

# Инициализировать рисунок/Figure
# dpi -- количество пикселей на дюйм в рисунке
# figsize -- пропорции "поля" рисунка
plt.figure(figsize=(8,5), dpi=100)

# Line 1

# Основные возможные аргументы функции plot. По умолчанию необходимы
только x и y
#plt.plot(x,y, label='2x', color='red', linewidth=2, marker='.',
linestyle='--', markersize=10, markeredgecolor='blue')

#нарисуем график первой функции -- 2x
plt.plot(x,y, 'b^--', label='2x')

## Line 2

# С помощью numpy мы можем создавать массив из чисел с определенным
интервалом функцией arange. np.linspace(..) делает то же самое, но с
целыми числами
x2 = np.arange(0,4.5,0.05)

# Нарисуем часть второго графика как сплошную кривую -- квадрат
значений x2
# Поскольку x2 -- массив numpy, мы можем это сделать просто "возведя в
квадрат" массив
plt.plot(x2[:6], x2[:6]**2, 'r', label='X^2')

# Нарисуем часть графика пунктиром. 'r' после x и y означает красный
цвет, '--' - рисовку пунктиром
plt.plot(x2[5:], x2[5:]**2, 'r--')

# Добавим заголовок (в fontdict нужен словарь, шрифт должен
поддерживаться matplotlib'ом)
```

```
plt.title('Our First Graph!', fontdict={'fontname': 'sans-serif',
'fontsize': 20})

# Подпишем оси
plt.xlabel('X Axis')
plt.ylabel('Y Axis')

# Зададим какие-нибудь корявые "штрихи"/ticks на осях. в эти фунции
можно передать любой список
plt.xticks([0,1.12,2.66,3,3.5])
plt.yticks([0,2,4,6,8,10])

# сделаем по этим штрихам сетку
plt.grid()

# функция легенды графика для отображения label'ов графиков
plt.legend()

# Можем сохранить график в высоком качестве
plt.savefig('mygraph.png', dpi=300)

# И вызвать эту функцию чтобы график сразу после отрисовки не пропал,
пока мы его не закроем
plt.show()
```



Столбчатая диаграмма (bar chart)

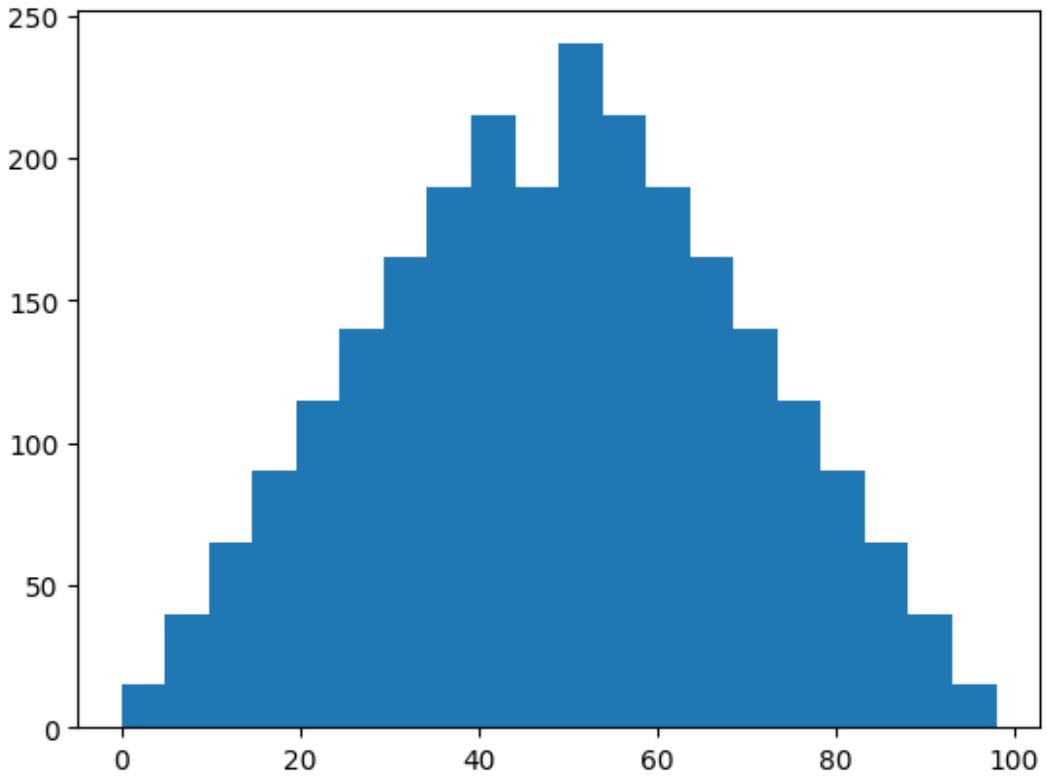
```
# принцип настройки схожий, но вместо plt.plot используем plt.bar
labels = ['A', 'B', 'C']
values = [1,4,2]

plt.figure(figsize=(5,3), dpi=100)

bars = plt.bar(labels, values)

plt.savefig('barchart.png', dpi=300)

plt.show()
```



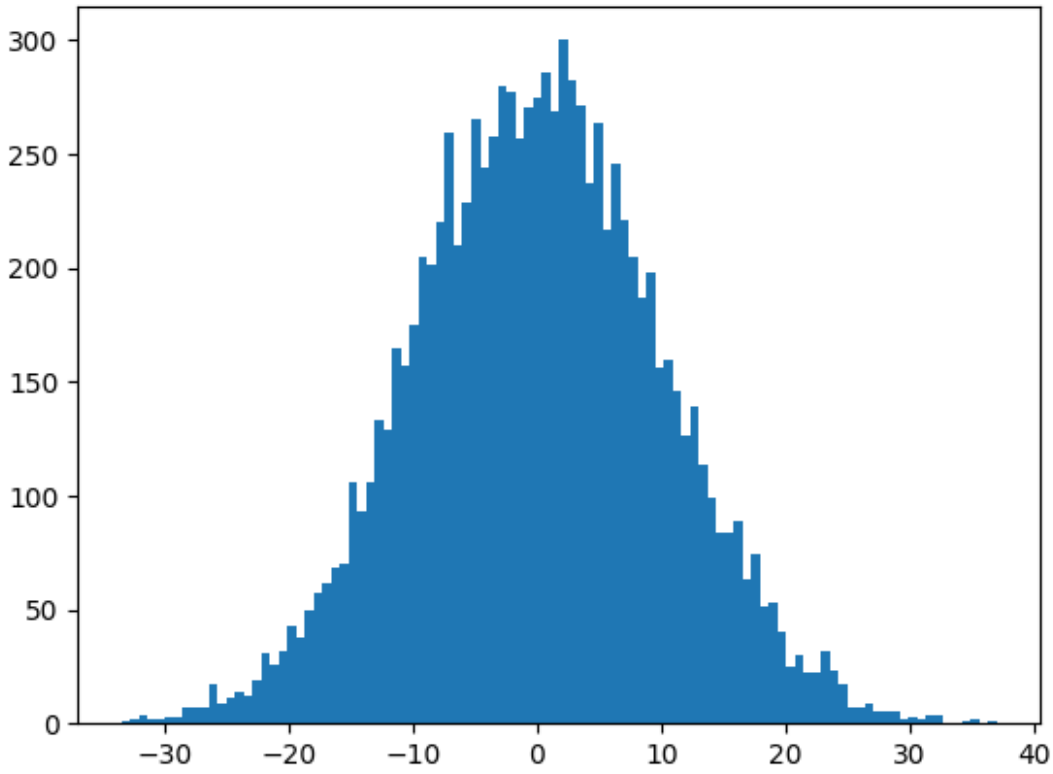
Гистограмма

```
# создадим два списка чисел от 0 до 49 и составим список S попарных
сумм элементов двух списков
# посмотрим распределение получившихся чисел построив гистограмму

x = [i for i in range(50)]
y = [j for j in range(50)]
S = [i + j for i in x for j in y]
print(len(S))
#bins задает количество столбцов гистограммы. если не задать,
подберутся автоматически
plt.hist(S, bins = 20)

plt.show()
# получился треугольник с центром около 50
# сложить два равномерных распределения -- самый простой способ
получить треугольное распределение
```

```
2500
```

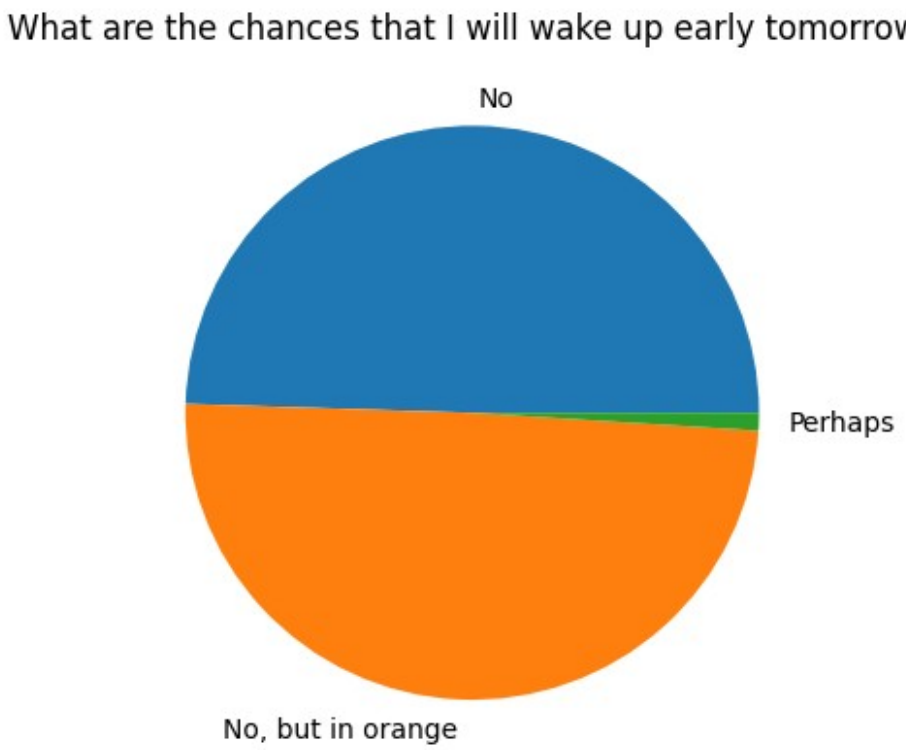



```
# попробуем сгенерировать случайные числа из нормального распределения
и посмотреть, как оно выглядит

# среднее
pos = 0

# параметр отвечающий за разброс
scale = 10

# размер массива случайных чисел (сколько их сгенерируем)
size = 10000

# используем функцию из подраздела random библиотеки numpy и передадим
наши параметры
values = np.random.normal(pos, 10, size)

# строим гистограмму с 100 блоков
plt.hist(values, 100)

plt.show()
```



Круговая диаграмма (pie chart)

```
plt.pie([0.5, 0.5, 0.01], labels = ['No','No, but in
orange','Perhaps'])

plt.title('What are the chances that I will wake up early tomorrow?')

plt.show()
```



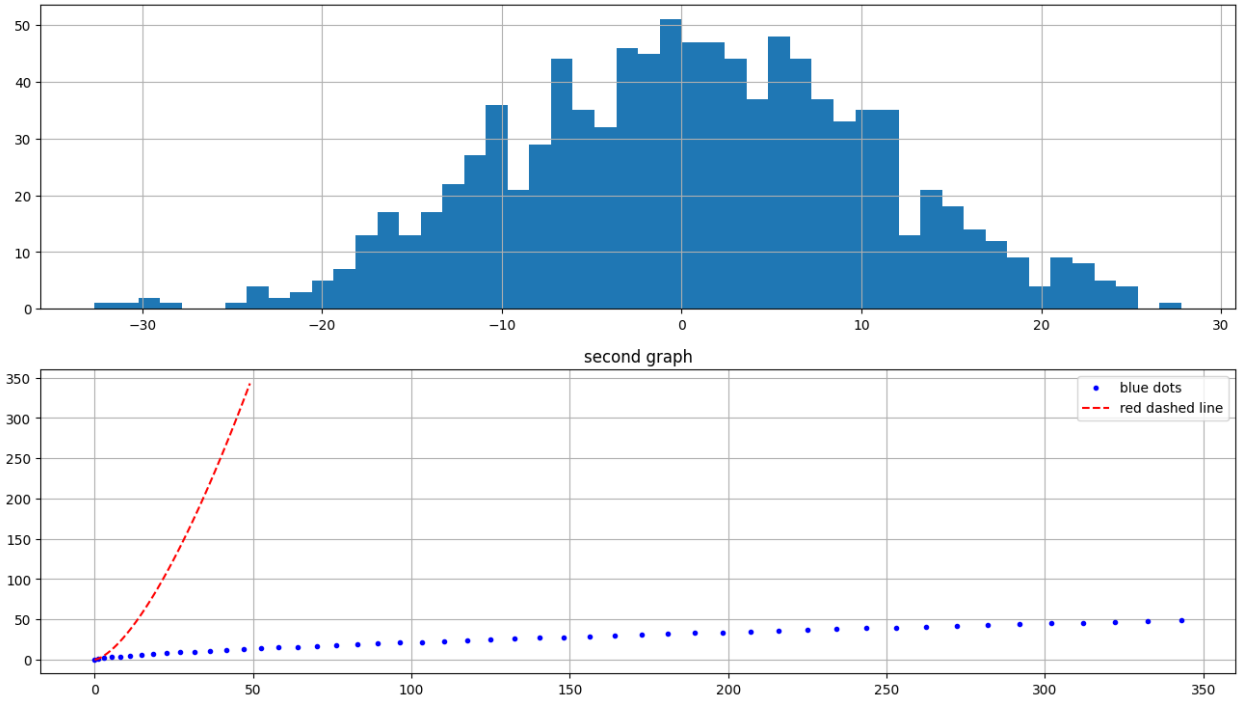
Несколько графиков

В случае, если нам необходимо построить несколько графиков, таким простым подходом не ограничишься. В этом случае будет правильнее создавать отдельные **объекты** типов `Figure` и `Axes` и настраивать их посредством **методов** этих объектов.

```
fig = plt.figure(figsize = (16,9)) # создали рисунок/Figure Fig пропорциями 16:9
ax1 = fig.add_subplot(211) # создали Axes (подграфик) ax1 в серии из 2 графиков, поставили на позицию [1,1] -- левый верхний угол
ax2 = fig.add_subplot(212) # создали Axes ax2 в серии из 2 графиков, поставили на позицию [1,2] -- первый график во второй "строке" графиков

# сгенерируем данные для какой-нибудь гистограммы
values = np.random.normal(0, 10, 1000)

# строим гистограмму с 50 блоками
ax1.hist(values, 50)
ax1.grid() # делаем сетку на графике ax1

x = [i for i in range(50)]
y = [j**1.5 for j in x]


ax2.plot(y,x,'b.', label = 'blue dots')
```

```
ax2.plot(x,y,'r--', label = 'red dashed line')
ax2.set_title('second graph') #здесь название функции немного
отличается от случая, когда мы вызывали напрямую из plt!


ax2.grid() # делаем сетку на графике ax2
ax2.legend() # делаем легенду на графике ax2

plt.show()
```

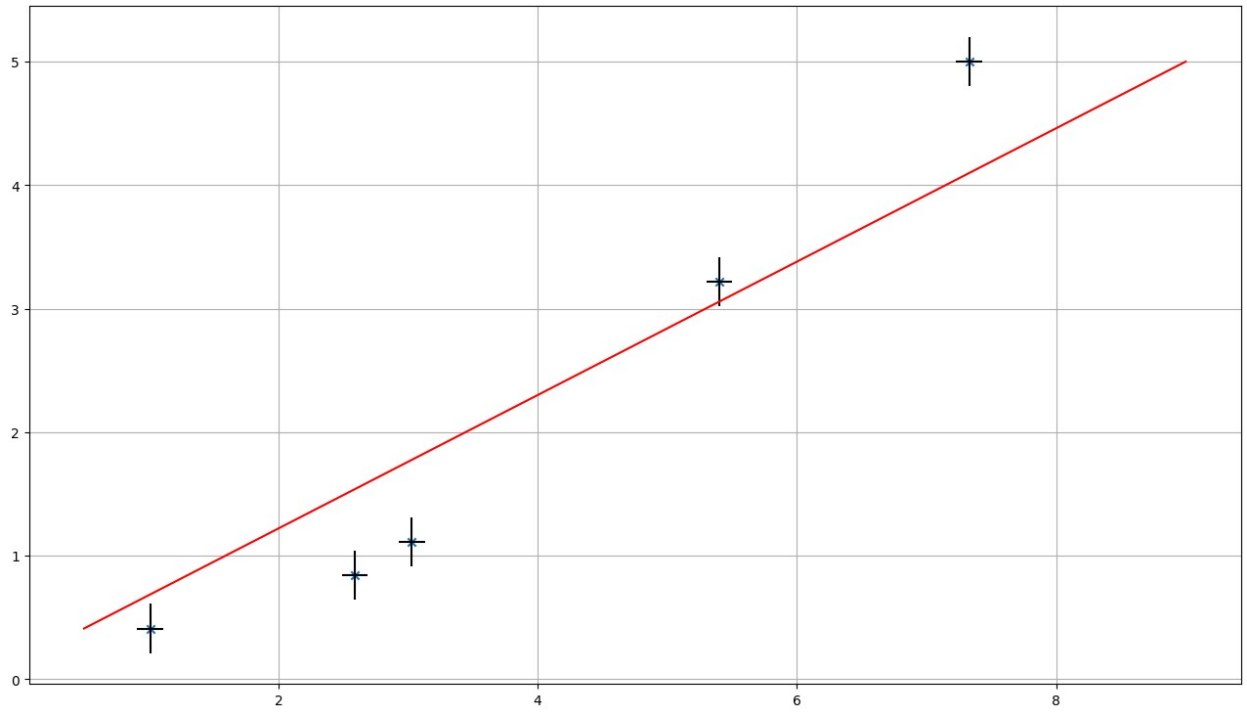
```
C:\Users\Klimkou\AppData\Local\Temp\ipykernel_12632\1725427724.py:24:
UserWarning: Matplotlib is currently using
module://matplotlib_inline.backend_inline, which is a non-GUI backend,
so cannot show the figure.
  fig.show()
```



В реальности данные, которые мы измеряем или получаем, не всегда ложатся в ровную закономерность, поэтому помимо линии аппроксимации (например прямой, полученной из МНК), имеет смысл также дорисовывать изначальные измеренные точки. С крестами погрешностей. Давайте это сделаем, чтобы было совсем красиво.

```
fig = plt.figure(figsize = (16,9)) # создали рисунок/Figure Fig
пропорциями 16:9
ax1 = fig.add_subplot(111) # допустим, больше 1 графика нам не надо

x_measured = [1.01, 2.59, 3.03, 5.40, 7.33]
y_measured = [0.41, 0.84, 1.11, 3.22, 5.00]
```

```
#используем встроенный линейный интерполятор чтобы посчитать значения прямой МНК в точках, на которых будем строить нашу прямую
#Поскольку мы хотим прямую, нам достаточно двух точек -- начало и конец прямой
x = [0.5, 9.0]
y = np.interp(x, x_measured, y_measured)

# ставим точки функцией scatter, точки будем ставить крестиком
ax1.scatter(x_measured, y_measured, marker='x')

# поставим кресты погрешностей, linestyle = None, чтобы кресты не соединялись прямыми
ax1.errorbar(x_measured, y_measured, yerr=0.2, xerr = 0.1, color = 'k', linestyle = 'None')

#построим красную прямую МНК
ax1.plot(x,y, 'r')

ax1.grid() # делаем сетку

plt.show()
# для готового графика для лабы по общефизу не хватает только названия, подписанных осей и легенды, но это вы уже умеете
# успехов!
```



В папке Seminars лежат файлы iris_data.csv и BTC_data.csv. **CSV** -- comma-separated-values -- значения, разделенные запятой -- формат файла, в котором хранится серия значений, каждое на новой строке, где каждая строка содержит некоторое количество величин,

разделенных запятой. Формат CSV поддерживается Excel, можете посмотреть их содержимое. Эти файлы понадобятся для выполнения заданий 3-6, скачайте их себе в директорию, где располагаются ваши .py файлы. Для чтения CSV файлов вам может пригодиться функция `read_csv` библиотеки `pandas`, однако считывать данные из CSV можно и без нее при помощи обычных `readlines` и `split`.

Упражнение 1 (Эксперименты)

Составьте произвольный набор данных. Постройте на них график, подходящий для отчета по общей физике.

Упражнение 2 (Стремление к нормальности)

Постройте на одном рисунке (но разных подграфиках) несколько гистограмм, которые покажут, что с увеличением количества точек полученных из распределения, гистограмма значений этих точек стремится к этому распределению.

Упражнение 3 (Пироги с цветами)

Постройте из файла iris_data -- стандартного датасета цветков ириса -- две круговые диаграммы: доля видов (Species) ирисов в датасете, Доли ирисов, у которых длина лепестка (PetalLengthCm) больше 1.2см, больше 1.2см и меньше 1.5см, больше 1.5см.

Упражнение 4 (Подгониан)

Постройте из файла iris_data все возможные комбинации длин и ширин лепестков и чашелистников (Petal и Sepal). Можно ли заметить какие-либо закономерности? Если да, попробуйте построить прямую МНК и вывести на экран ее коэффициенты.

Упражнение 5 (Ты родился слишком поздно, чтобы ...)

В файле BTC_data лежат ежедневные значения цены биткоина на бирже с 2018 по 2023 год. Не будем вдаваться в подробности, чем отличаются четыре указанные цены, нас интересует только последнее число в каждой строке -- цена закрытия (close) и дата, первое значение в каждой строке. Постройте исторический график зависимости цены биткоина от времени. В качестве зарубок по оси X используйте даты в формате DD-MM-YY (например 06-03-2005 -- означает 6 марта 2005 года).

Кроме функции `interp` в `numpy` также есть функция `polyfit`. Попробуйте аппроксимировать цену биткоина каким-нибудь полиномом и постройте его на том же графике, что график цены.

```
df = pd.read_csv('iris_data.csv')
A = list(df['Species'])

list(df['close'])

x = np.array([0.0, 1.0, 2.0, 3.0,  4.0,  5.0])
y = np.array([0.0, 0.8, 0.9, 0.1, -0.8, -1.0])
z = np.polyfit(x, y, 3) # аппроксимировать функцию y(x) полиномом
```

```
третьей степени
print(z)
# чтобы не приходилось каждый раз строить полином с полученными
коэффициентами, также существует функция poly1d
p = np.poly1d(z)
# теперь чтобы получить значение полинома с коэффициентами z в точке,
например, 5, нам надо вызвать p(5)
print(p(5))
```

```
[ 0.08703704 -0.81349206  1.69312169 -0.03968254]
-1.0317460317460356
```