KARPOV.COURSES >>>
КОНСПЕКТ

> Конспект > 3 урок > Продвинутые расчеты и визуализация

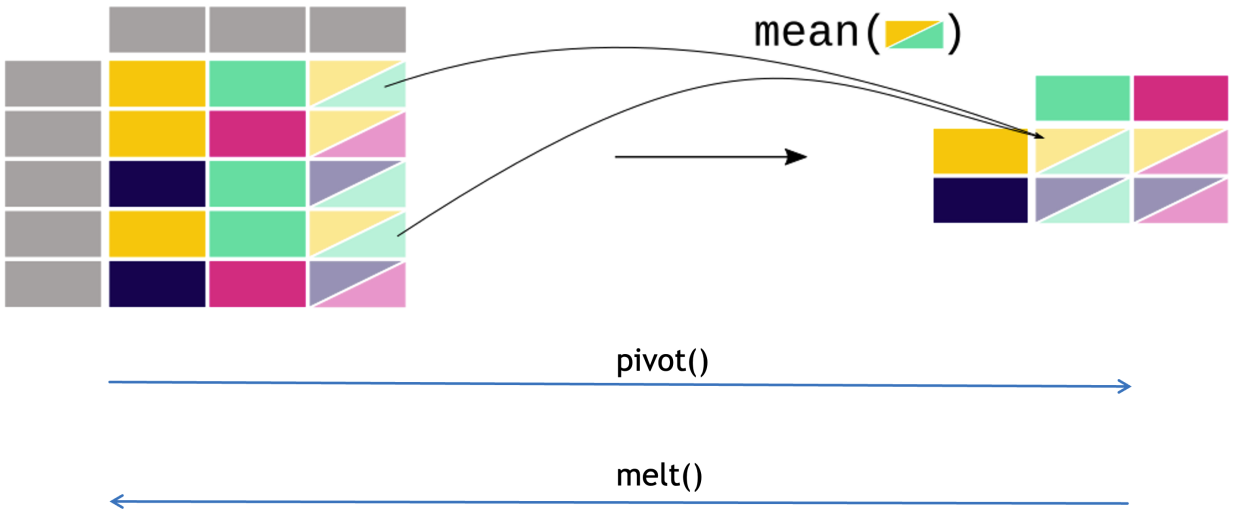
> Оглавление

> Сводные таблицы и объединение датафреймов

Иногда та структура, в которой к нам приходит набор данных, не очень удобна для наших целей. Решение проблемы — изменить форму датафрейма.

Метод `.pivot()`

С помощью `.pivot()` можно изменить структуру датафрейма, сделав его более широким. Обратная операция с «удлинением» датафрейма называется `.melt()`. Порой в этом также замешана агрегация данных, но об этом чуть позже.



Возьмём урезанный набор данных по содержанию NO_2:

date.utc	city	country	location	parameter	value	unit
2019-04-09 01:00:00+00:00	Antwerpen	BE	BETR801	no2	22.5	µg/m³
2019-04-09 01:00:00+00:00	Paris	FR	FR04014	no2	24.4	µg/m³
2019-04-09 02:00:00+00:00	London	GB	London Westminster	no2	67.0	µg/m³
2019-04-09 02:00:00+00:00	Antwerpen	BE	BETR801	no2	53.5	µg/m³
2019-04-09 02:00:00+00:00	Paris	FR	FR04014	no2	27.4	µg/m³
2019-04-09 03:00:00+00:00	London	GB	London Westminster	no2	67.0	µg/m³

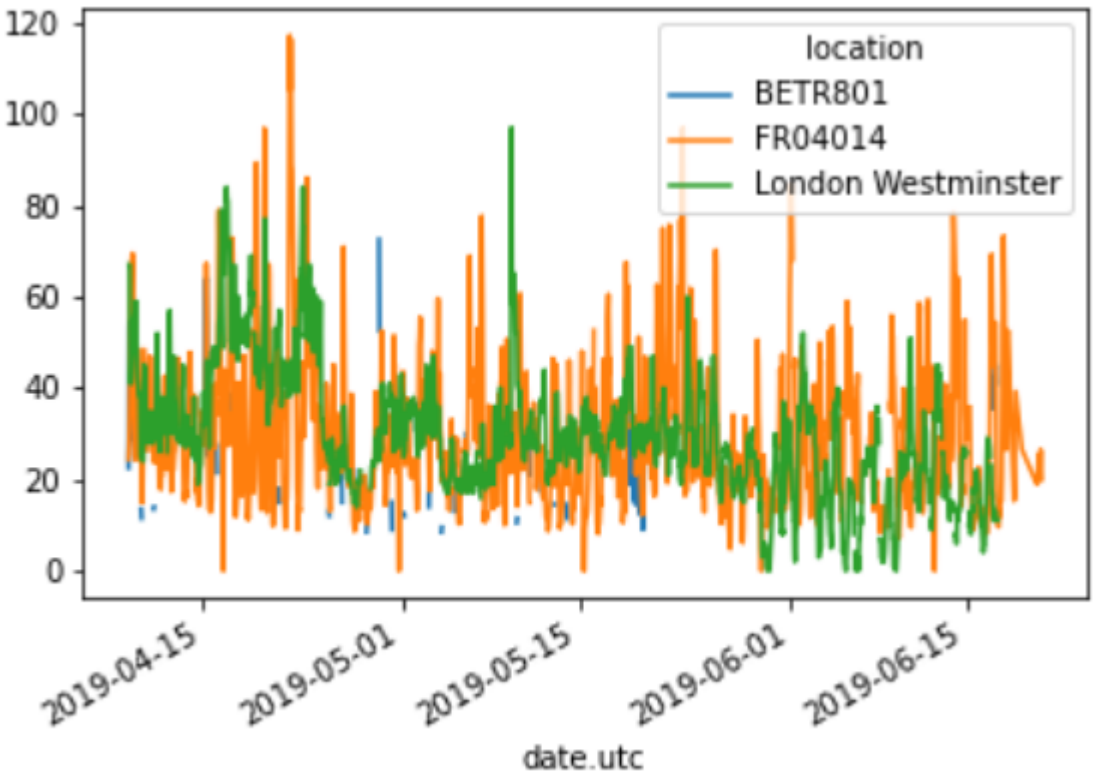
Вытянем его в ширину так, что на месте столбцов будут места замеров, а на месте значений — концентрации оксида азота:

```
no2_subset.pivot(columns="location", values="value")

#по этому можно посмотреть, в какое время и в каком месте была н
```

location	BETR801	FR04014	London Westminster
date.utc			
2019-04-09 01:00:00+00:00	22.5	24.4	NaN
2019-04-09 02:00:00+00:00	53.5	27.4	67.0
2019-04-09 03:00:00+00:00	NaN	NaN	67.0

Подобная форма позволит нам рисовать данные с помощью того же `pandas` — всего лишь добавьте `.plot()`:

```
#сделаем это на полных данных
no2.pivot(columns="location", values="value").plot()

#на графике видна динамика изменения концентраций
#цветом раскрашены каждое из трёх мест
#не очень аккуратно выглядит, но всё же
```



Родственным методом является `.pivot_table()` — пользоваться им немного сложнее, но он также позволяет совершать агрегацию данных наряду с изменением их формы. Испытаем это на данных airquality до фильтрации по NO_2:

date.utc	city	country	location	parameter	value	unit
2019-06-18 06:00:00+00:00	Antwerpen	BE	BETR801	pm25	18.0	µg/m³
2019-06-17 08:00:00+00:00	Antwerpen	BE	BETR801	pm25	6.5	µg/m³
2019-06-17 07:00:00+00:00	Antwerpen	BE	BETR801	pm25	18.5	µg/m³
2019-06-17 06:00:00+00:00	Antwerpen	BE	BETR801	pm25	16.0	µg/m³
2019-06-17 05:00:00+00:00	Antwerpen	BE	BETR801	pm25	7.5	µg/m³

```
air_quality.pivot_table(
    # числовое значение для агрегации
    values="value",

    # строки
    index="location",

    # столбцы
    columns="parameter",

    # тип агрегации
    aggfunc="mean")

#получается табличка со средними значениями по каждому месту зам
```

parameter	no2	pm25
location		
BETR801	26.950920	23.169492
FR04014	29.374284	NaN
London Westminster	29.740050	13.443568

Помимо среднего значения по ячейкам, мы можем подсчитать средние значения по столбцам и по строкам! Хотите узнать среднее значение NO_2 по всем местам замеров? Или среднюю концентрацию по обоим веществам в London Westminster? Или среднее значение по всем местам и всем веществам сразу? Добавьте аргумент `margins=True`:

parameter location	no2	pm25	All
BETR801	26.950920	23.169492	24.982353
FR04014	29.374284	NaN	29.374284
London Westminster	29.740050	13.443568	21.491708
All	29.430316	14.386849	24.222743

Конкатенация датафреймов

Иногда бывает так, что все нужные данные лежат не в одной табличке, а в разных. Мы можем захотеть объединить такие данные в общую табличку — есть много способов это сделать. Сейчас мы рассмотрим один из них.

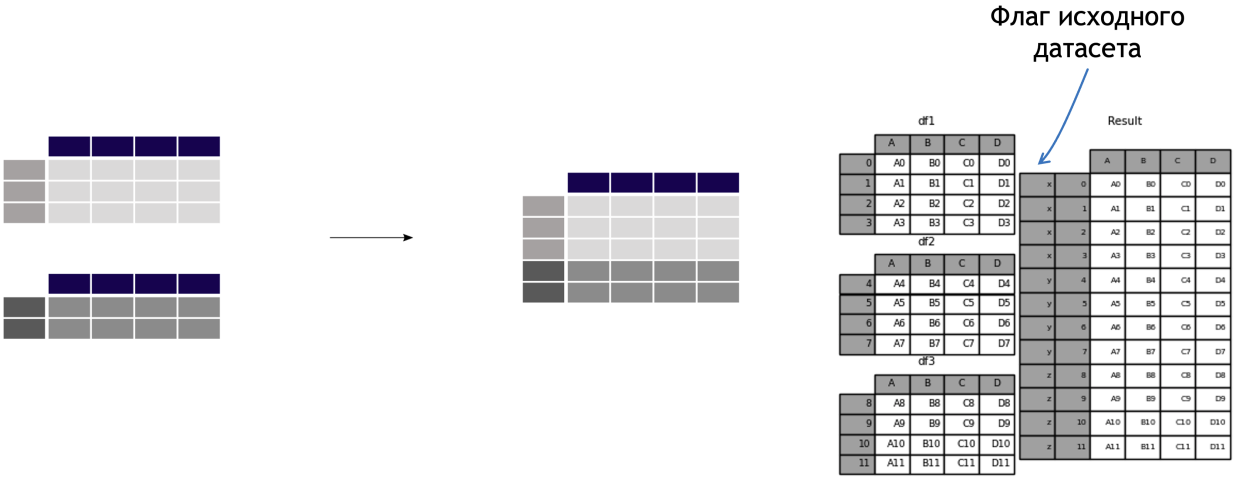
Допустим, у нас оказался не один датафрейм airquality, а два. В одном — данные только по NO_2:

	date.utc	location	parameter	value
0	2019-06-21 00:00:00+00:00	FR04014	no2	20.0
1	2019-06-20 23:00:00+00:00	FR04014	no2	21.8
2	2019-06-20 22:00:00+00:00	FR04014	no2	26.5
3	2019-06-20 21:00:00+00:00	FR04014	no2	24.9
4	2019-06-20 20:00:00+00:00	FR04014	no2	21.4

А в другом — только по $PM2.5$:

	date.utc	location	parameter	value
0	2019-06-18 06:00:00+00:00	BETR801	pm25	18.0
1	2019-06-17 08:00:00+00:00	BETR801	pm25	6.5
2	2019-06-17 07:00:00+00:00	BETR801	pm25	18.5
3	2019-06-17 06:00:00+00:00	BETR801	pm25	16.0
4	2019-06-17 05:00:00+00:00	BETR801	pm25	7.5

Как можно объединить их в один? С помощью конкатенации — просто «положить» один датафрейм сверху на другой:



Реализуется это через функцию `pd.concat()`:

```
#все датафреймы должны быть внутри списка
#axis задаёт ось, по которой соединяются датафреймы
#0 - сверху, 1 - сбоку

air_quality = pd.concat([air_quality_pm25, air_quality_no2], axi

#сравним размеры датфреймов
print(f'Shape "air_quality_pm25": {air_quality_pm25.shape}')
```

```
print(f'Shape "air_quality_no2": {air_quality_no2.shape}')
print(f'Shape результата "air_quality": {air_quality.shape}')

#число столбцов одинаковое
#число строк последнего - сумма первых двух
```

Output:

```
Shape "air_quality_pm25": (1110, 4)
Shape "air_quality_no2": (2068, 4)
Shape результата "air_quality": (3178, 4)
```

Можно также использовать аргумент `keys`, чтобы сгруппировать получившийся датафрейм:

```
air_quality_ = pd.concat([air_quality_pm25, air_quality_no2],
                         keys=["PM25", "NO2"])
air_quality_.head()
```

		date.utc	location	parameter	value
PM25	**0**	2019-06-18 06:00:00+00:00	BETR801	pm25	18.0
	1	2019-06-17 08:00:00+00:00	BETR801	pm25	6.5
	2	2019-06-17 07:00:00+00:00	BETR801	pm25	18.5
	3	2019-06-17 06:00:00+00:00	BETR801	pm25	16.0
	4	2019-06-17 05:00:00+00:00	BETR801	pm25	7.5

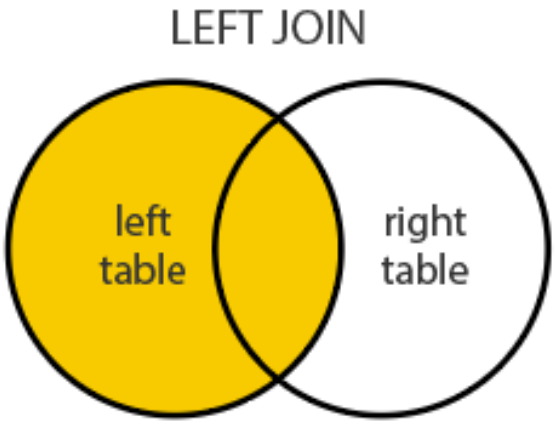
Объединение по ключу

Конкатенация — это довольно простой вариант объединения. Однако часто бывает так, что датафреймы могут быть разными по содержанию и даже размерам, но при этом могут иметь общий столбец или даже несколько. В

таких случаях датафреймы можно объединить именно по этому общему столбцу, по типу `join` в SQL.

Представим, что у нас опять airquality разделился на два! Но теперь всё чуть иначе: в первом датафрейме у нас данные по содержанию веществ в воздухе, а в другом — координаты мест измерения. При этом у них есть общий столбец с названием места.

Попробуем совершить так называемый left join: объединим данные так, чтобы первая таблица была полной, а все не соответствующие ей строки из второй таблицы были удалены. Поможет нам с этим функция `pd.merge()`:




```
#первый и второй аргументы - датафреймы
#how задаёт тип join-a (их много)
#on задаёт тот столбец, по которому датафреймы будут объединятьс

air_quality = pd.merge(air_quality, stations_coord,
                       how='left',
                       on='location')
air_quality.head()
```

	date.utc	location	parameter	value	coordinates.latitude	coordinates.longitude
0	2019-06-18 06:00:00+00:00	BETR801	pm25	18.0	51.20966	4.43182
1	2019-06-17 08:00:00+00:00	BETR801	pm25	6.5	51.20966	4.43182
2	2019-06-17 07:00:00+00:00	BETR801	pm25	18.5	51.20966	4.43182
3	2019-06-17 06:00:00+00:00	BETR801	pm25	16.0	51.20966	4.43182
4	2019-06-17 05:00:00+00:00	BETR801	pm25	7.5	51.20966	4.43182

Но это ещё не всё — у нас есть **третий датафрейм** с описанием каждого из измеряемых параметров!

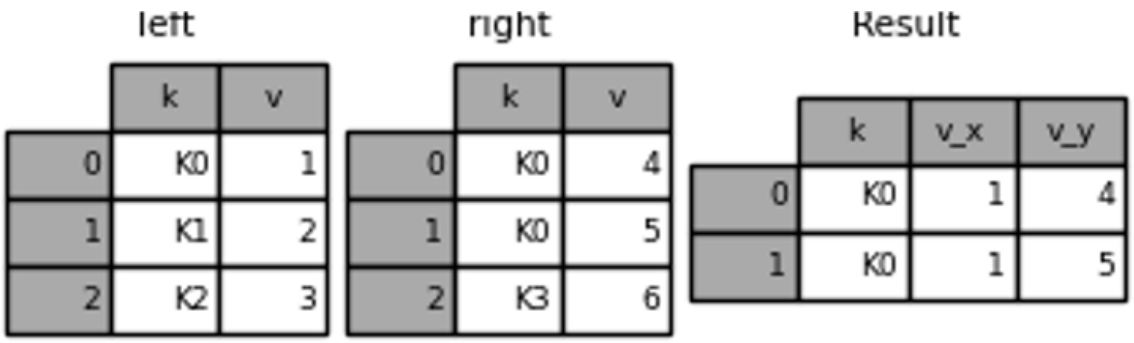
	id	description	name
0	bc	Black Carbon	BC
1	co	Carbon Monoxide	CO
2	no2	Nitrogen Dioxide	NO2
3	o3	Ozone	O3
4	pm10	Particulate matter less than 10 micrometers in...	PM10

Загвоздка: у этих датафреймов есть общий столбец, но называется он по-разному. В нашем объединённом датафрейме он называется `parameter`, а в третьем датафрейме он называется `id`. Здесь нас выручают аргументы `left_on` и `right_on`:

```
#left_on - как столбец называется в первом датафрейме
#right_on - как во втором

air_quality = pd.merge(air_quality, air_quality_parameters,
                       how='left',
                       left_on='parameter', right_on='id')
air_quality.head()
```

	date.utc	location	parameter	value	coordinates.latitude	coordinates.longitude	id	description	name
0	2019-06-18 06:00:00+00:00	BETR801	pm25	18.0	51.20966	4.43182	pm25	Particulate matter less than 2.5 micrometers i...	PM2.5
1	2019-06-17 08:00:00+00:00	BETR801	pm25	6.5	51.20966	4.43182	pm25	Particulate matter less than 2.5 micrometers i...	PM2.5
2	2019-06-17 07:00:00+00:00	BETR801	pm25	18.5	51.20966	4.43182	pm25	Particulate matter less than 2.5 micrometers i...	PM2.5
3	2019-06-17 06:00:00+00:00	BETR801	pm25	16.0	51.20966	4.43182	pm25	Particulate matter less than 2.5 micrometers i...	PM2.5
4	2019-06-17 05:00:00+00:00	BETR801	pm25	7.5	51.20966	4.43182	pm25	Particulate matter less than 2.5 micrometers i...	PM2.5

На закуску: что произойдёт, если одинаковые названия есть у тех столбцов, по которым мы не объединяем? Хитрый `pandas` просто добавит к их именам суффиксы: `_x` для первого, `_y` для второго:



> Matplotlib

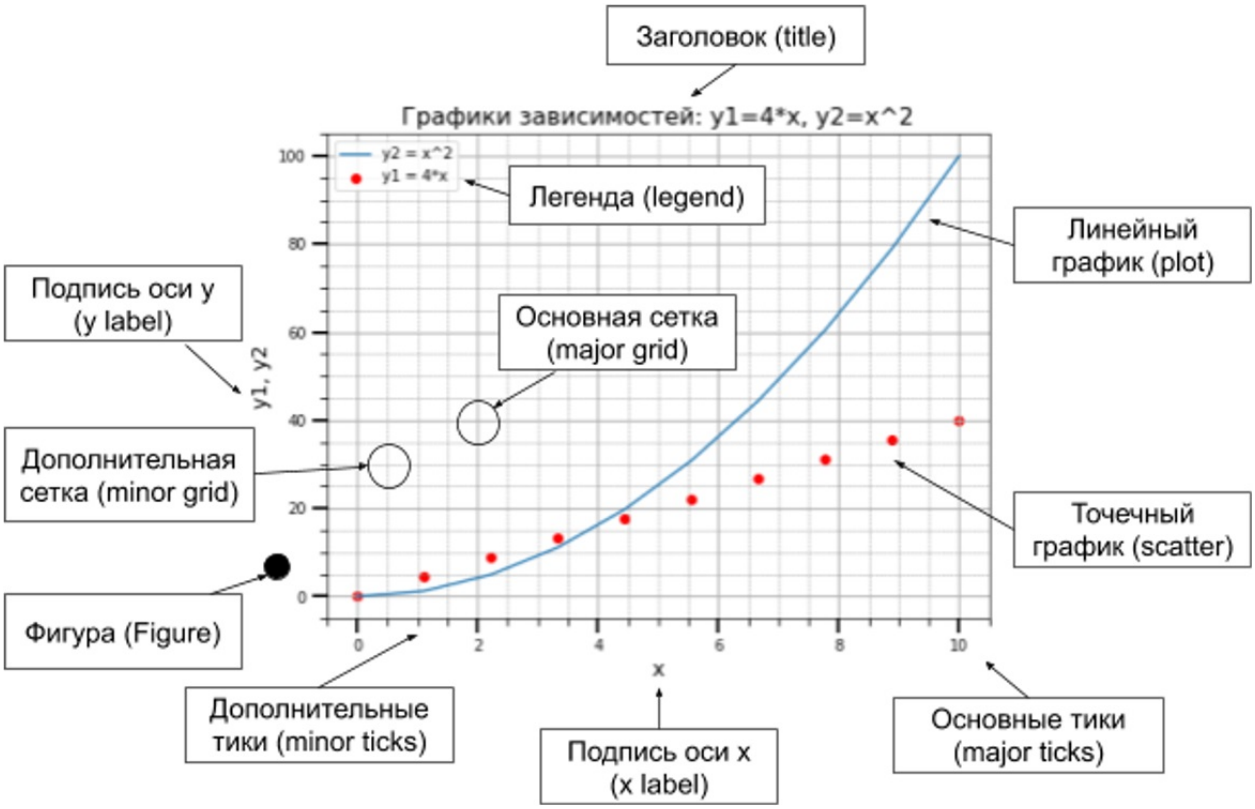
Мы успели совсем чуть-чуть порисовать, когда разбирались с `pandas` и использовали метод `.plot()`. Но в реальной работе вам придётся неоднократно рисовать графики: наиболее фундаментальная библиотека для визуализации данных называется `matplotlib`.

1. Базовый функционал очень лёгкий, но тонкая настройка элементов очень разветвлённая и сложная
2. Внутри `matplotlib` есть несколько модулей, но на практике чаще всего нужен `pyplot`
3. Множество типов графиков: гистограммы, точечные диаграммы, столбиковые и т.д
4. Множество других библиотек и пакетов с функциями визуализации имеют в своей основе именно `matplotlib` — в том числе уже упомянутый `pandas`

Документация: https://matplotlib.org/stable/index.html

```
import matplotlib.pyplot as plt #plt - конвенциональный алиас
```

Основные элементы графика



На самом деле элементы графика в контексте `matplotlib` можно поделить на два элемента:

1. `Figure` — фигура, контейнер верхнего уровня, практически «холст» для всех элементов изображения
2. `Axes` — оси, место, где и рисуется сам график

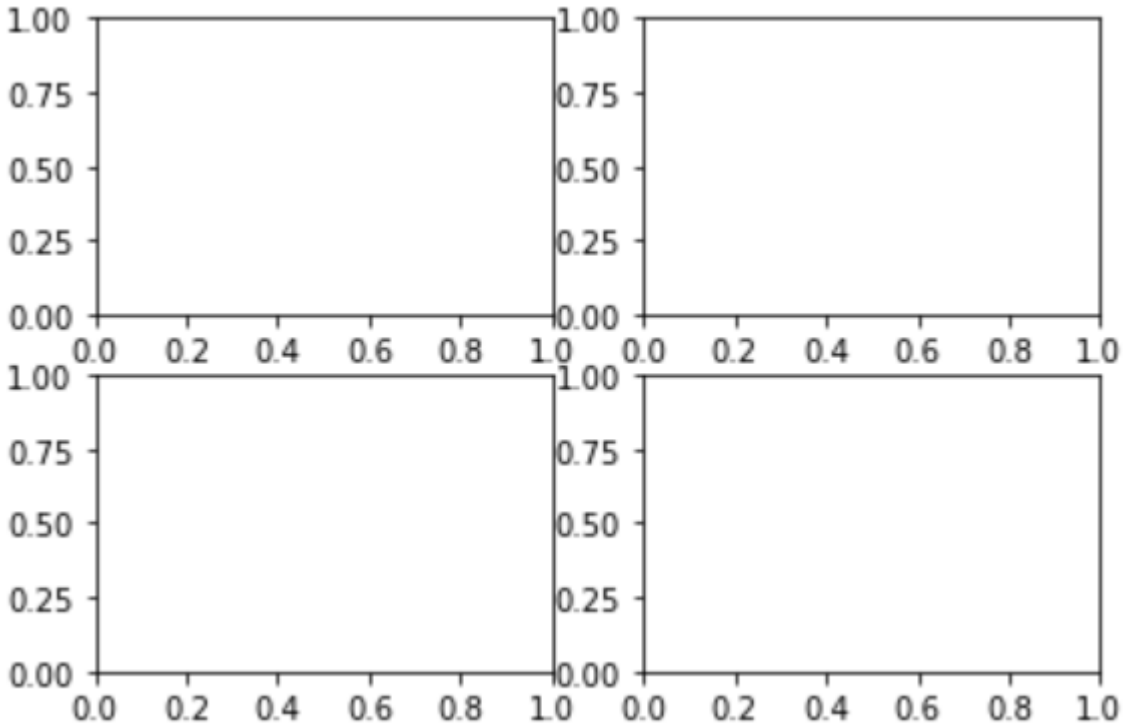
Можно создать фигуру без осей, но визуально в ней ничего не будет:

```
fig = plt.figure()

#можно увидеть, что объект создался
```

```
#он даже занимает какое-то место в памяти
#но из-за нуля осей графика никакого нет

#через аргумент figsize можно задавать размеры осей x и y
#принимает на вход кортеж: figsize=(10,5), например
```

А можно создать фигуру с несколькими осями — полезно в тех случаях, когда хочется нарисовать несколько разных графиков в одном пространстве:

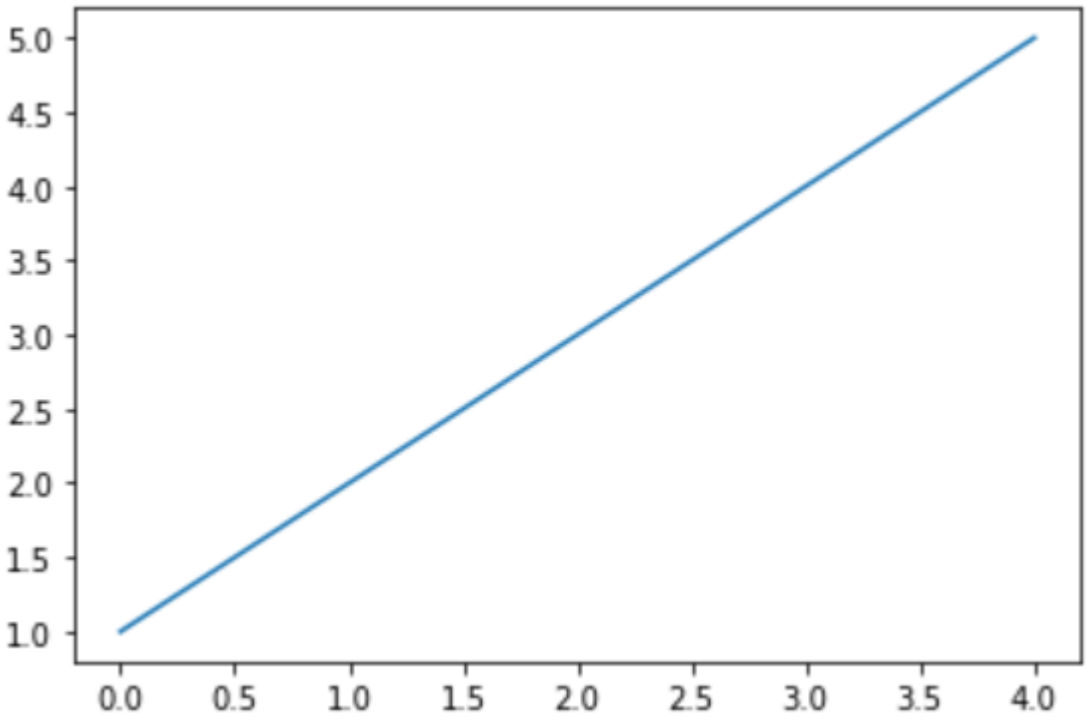
```
fig, ax_lst = plt.subplots(2, 2)

#так создадутся 4 объекта осей
#две строки, два столбца
#можно увеличивать или уменьшать число строк
#на выходе два объекта - фигура и список осей
```



Рисовать и подписывать

`plt.plot()` — функция, которая позволяет рисовать линии и маркеры, при желании одновременно. Давайте тогда нарисуем самую простую линию:

```
#если дать plt.plot() только один список значений, то он помести

plt.plot([1, 2, 3, 4, 5])
```




```
#а теперь нарисуем как надо - с заданными x и y

plt.plot([1, 2, 3, 4, 5], [2, 3, 4, 6, 7])
```

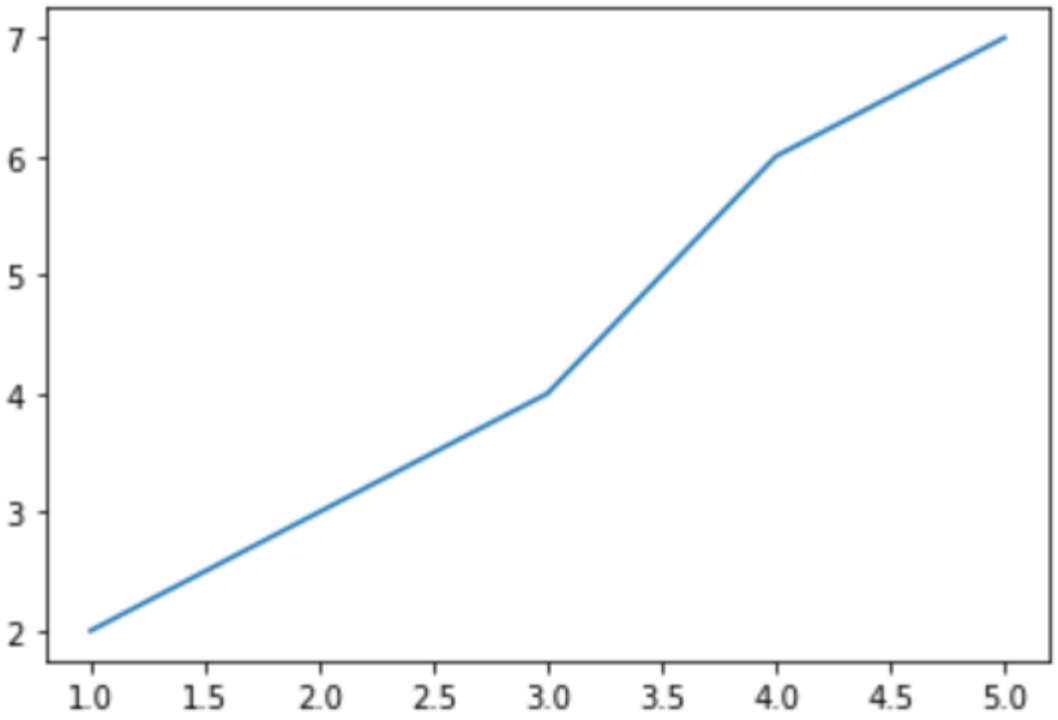
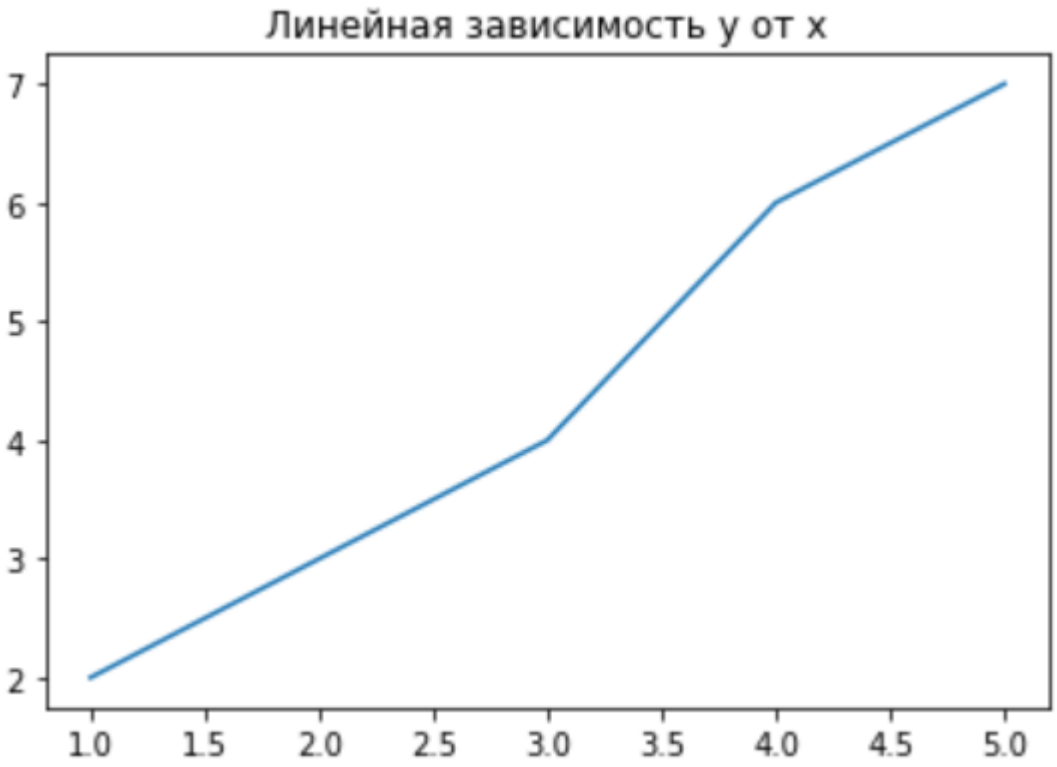


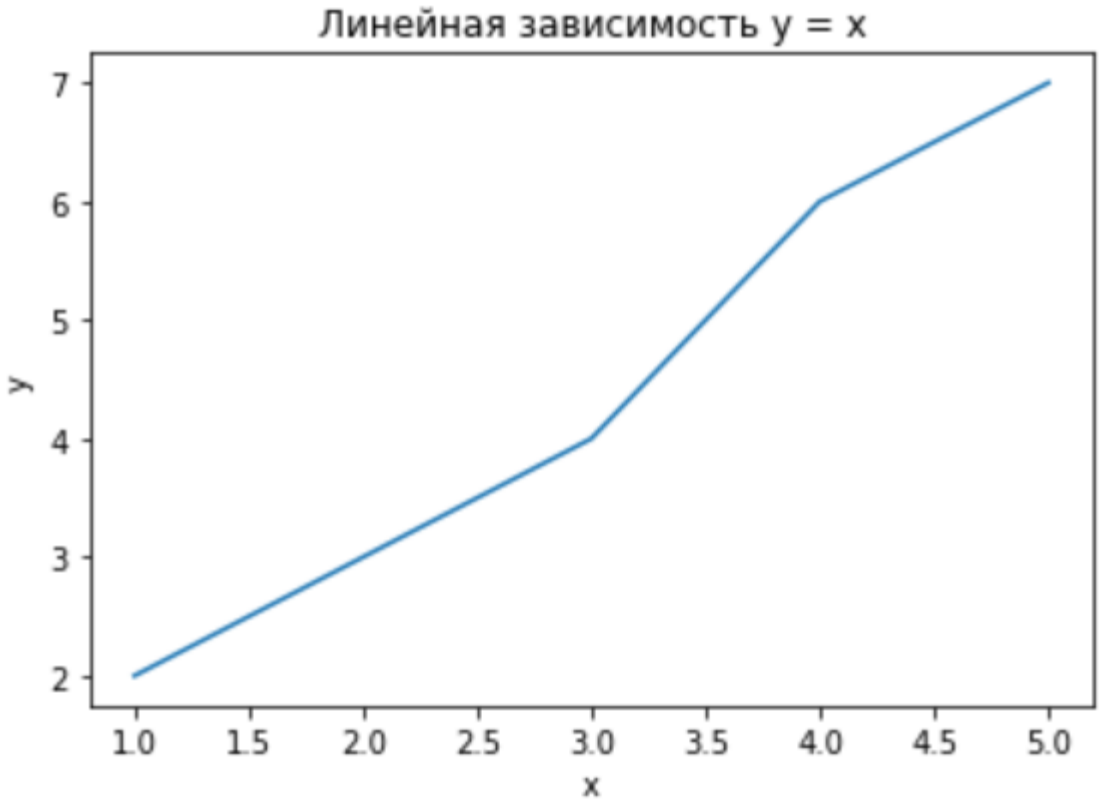
График — это хорошо, но без подписей мы никогда не поймём, что на нём происходит. Для начала добавим заголовок:

```
x = [1, 2, 3, 4, 5] #значения по оси x
y = [2, 3, 4, 6, 7] #значения по оси y#сначала рисуем график
plt.plot(x, y)
#затем функция для заголовка
plt.title("Линейная зависимость y от x")
```



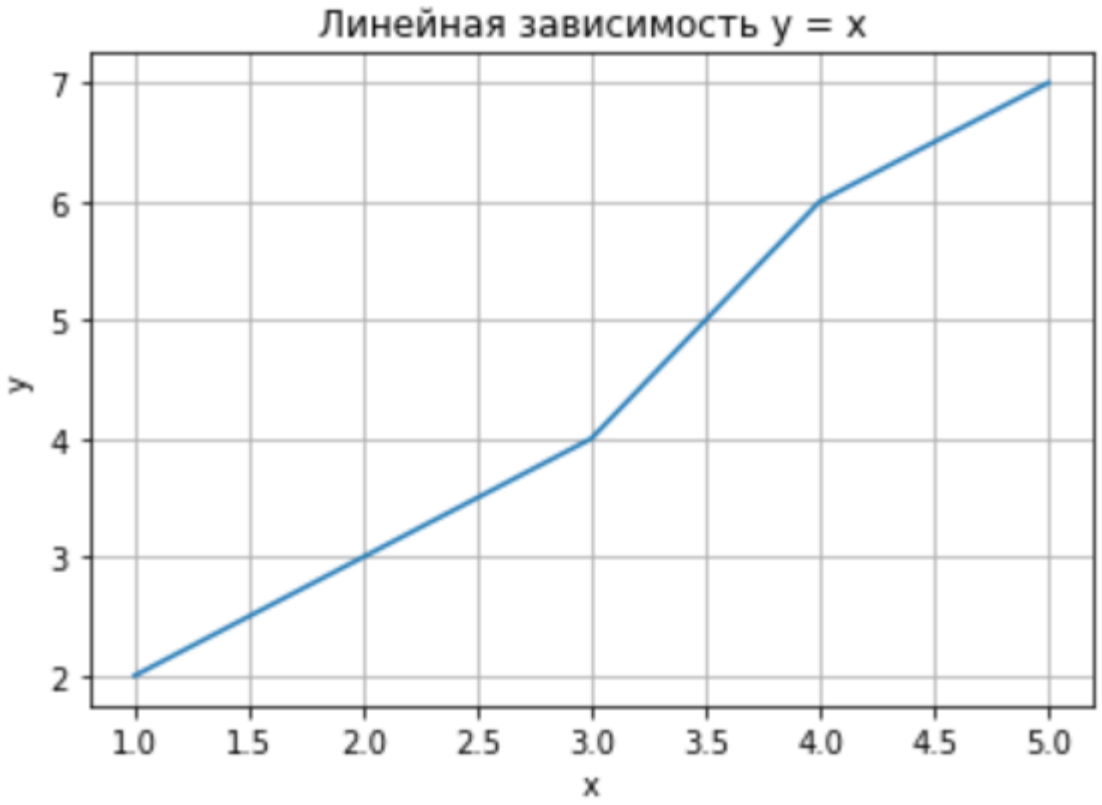
Теперь подпишем оси:

```
#построение графика
plt.plot(x, y)

#заголовок
plt.title("Линейная зависимость y = x")

#ось абсцисс
plt.xlabel("x")
#ось ординат
plt.ylabel("y")
```



Наконец, добавим к этому сетку, чтобы было удобнее смотреть на координаты графика:

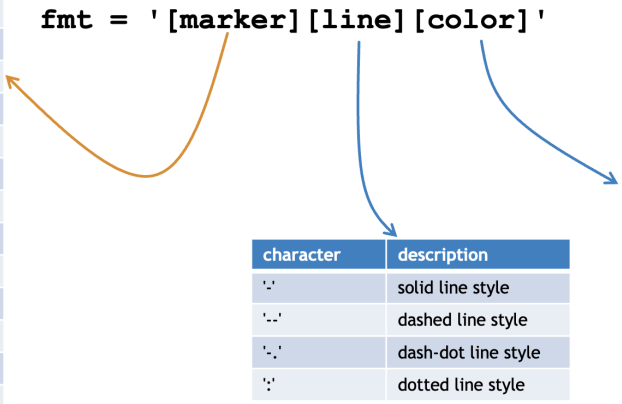
```
#построение графика
plt.plot(x, y)
#заголовок
plt.title("Линейная зависимость y = x")
#ось абсцисс
plt.xlabel("x")
#ось ординат
plt.ylabel("y")

#включение отображения сетки
plt.grid()
```



Форматирование линий

Перед нами синяя сплошная линия. Но что если мы хотим другой цвет? Или линию пунктиром? Или добавить маркеров? Или что-то другое? `plt.plot()` и это умеет: через аргумент `fmt` можно задать внешний вид маркеров, характер линий и цвет.

character	description
'.'	point marker
','	pixel marker
'o'	circle marker
'v'	triangle_down marker
'^'	triangle_up marker
'<'	triangle_left marker
'>'	triangle_right marker
'1'	tri_down marker
'2'	tri_up marker
'3'	tri_left marker
'4'	tri_right marker
's'	square marker
'p'	pentagon marker
'*'	star marker
'h'	hexagon1 marker
'H'	hexagon2 marker
'+'	plus marker
'x'	x marker
'D'	diamond marker
'd'	thin_diamond marker
'|'	vline marker
'_'	hline marker

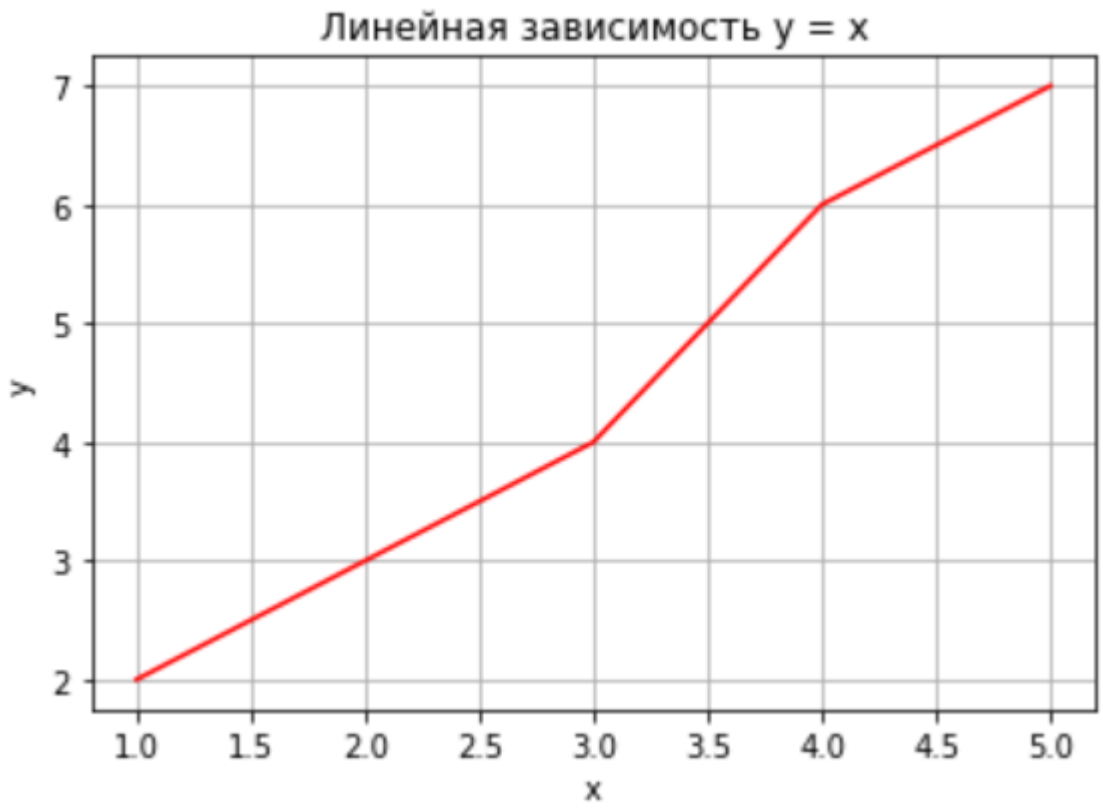


character	description
'-'	solid line style
'--'	dashed line style
'-.'	dash-dot line style
':'	dotted line style

character	color
'b'	blue
'g'	green
'r'	red
'c'	cyan
'm'	magenta
'y'	yellow
'k'	black
'w'	white

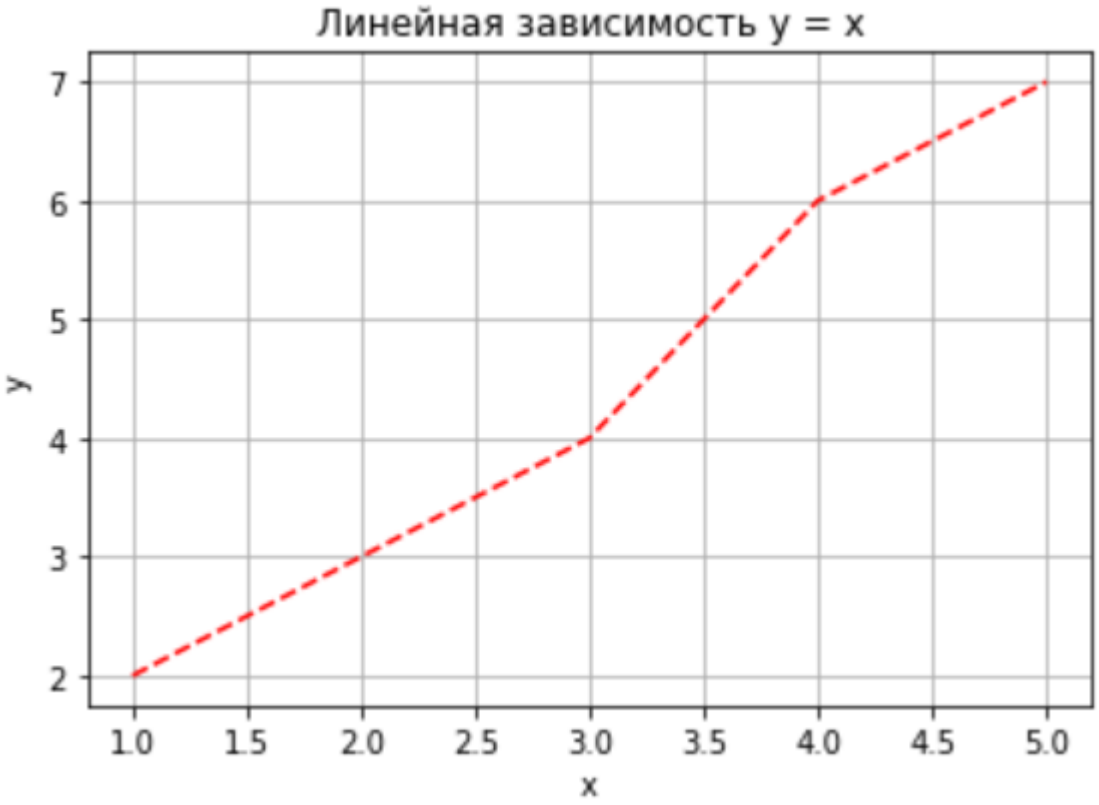
Например, давайте сделаем линию красной — укажем букву `'r'` (red):

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



А теперь добавим пунктир:

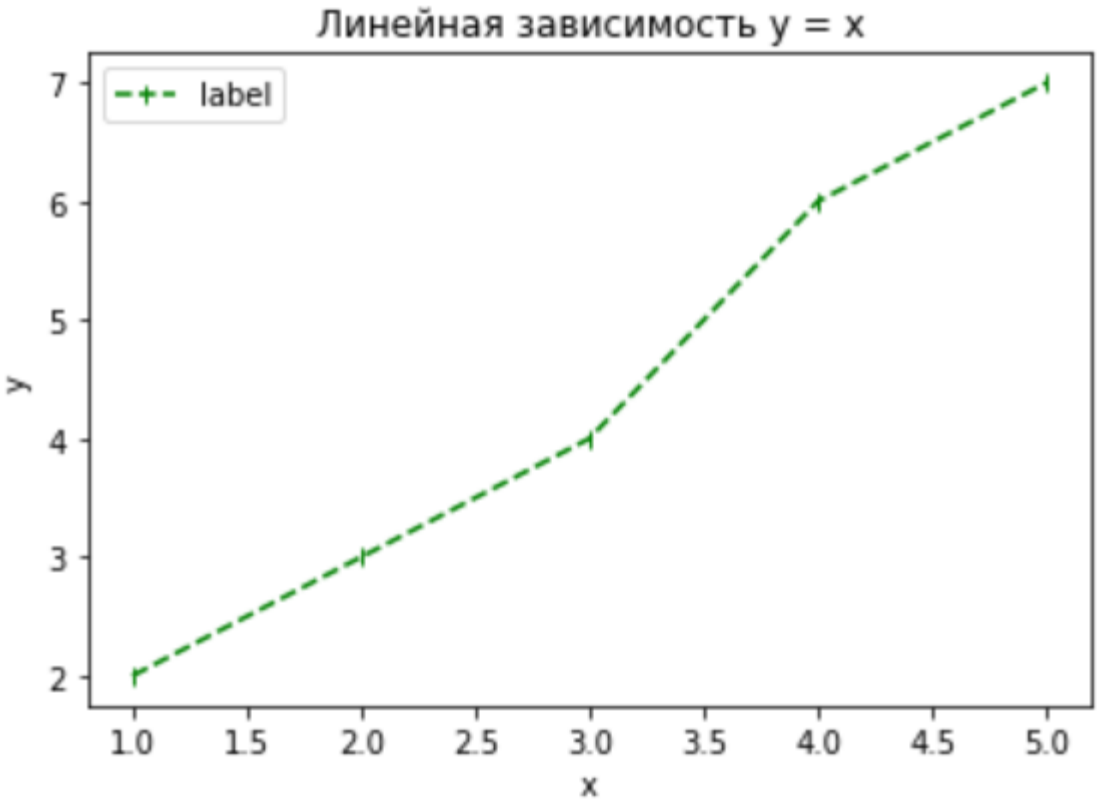
```
#построение графика
plt.plot(x, y, 'r--')
#заголовок
plt.title("Линейная зависимость y = x")
#ось абсцисс
plt.xlabel("x")
#ось ординат
plt.ylabel("y")
#включение отображения сетки
plt.grid()
```



А теперь давайте сделаем три вещи:

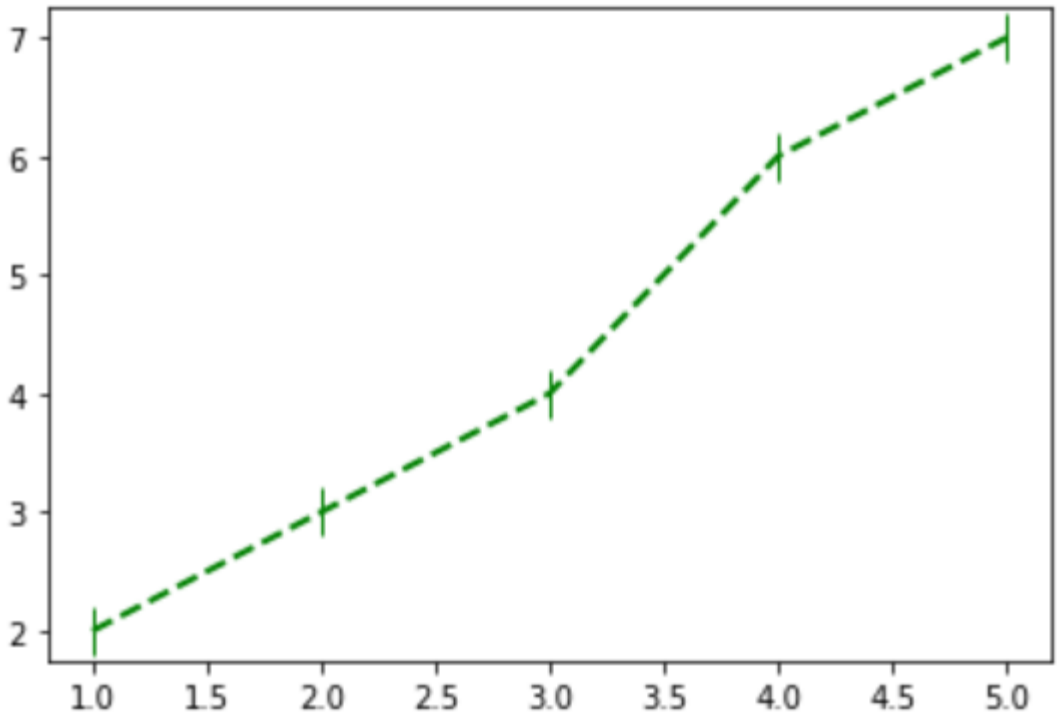
1. Сделаем цвет зелёным
2. Добавим маркеры в виде вертикальной линии `'|'`
3. Дадим этой линии имя и выведем легенду

```
# построение графика
plt.plot(x, y, '|--g', label='label') #label присваивает имя эт
plt.title("Линейная зависимость y = x")
#ось абсцисс
plt.xlabel("x")
#ось ординат
plt.ylabel("y")

#добавляем легенду
plt.legend()
```



Что ещё мы можем изменить на графике? Например, увеличить длину каждого пунктира через аргумент `linewidth` и увеличить размер каждого маркера через аргумент `markersize`:

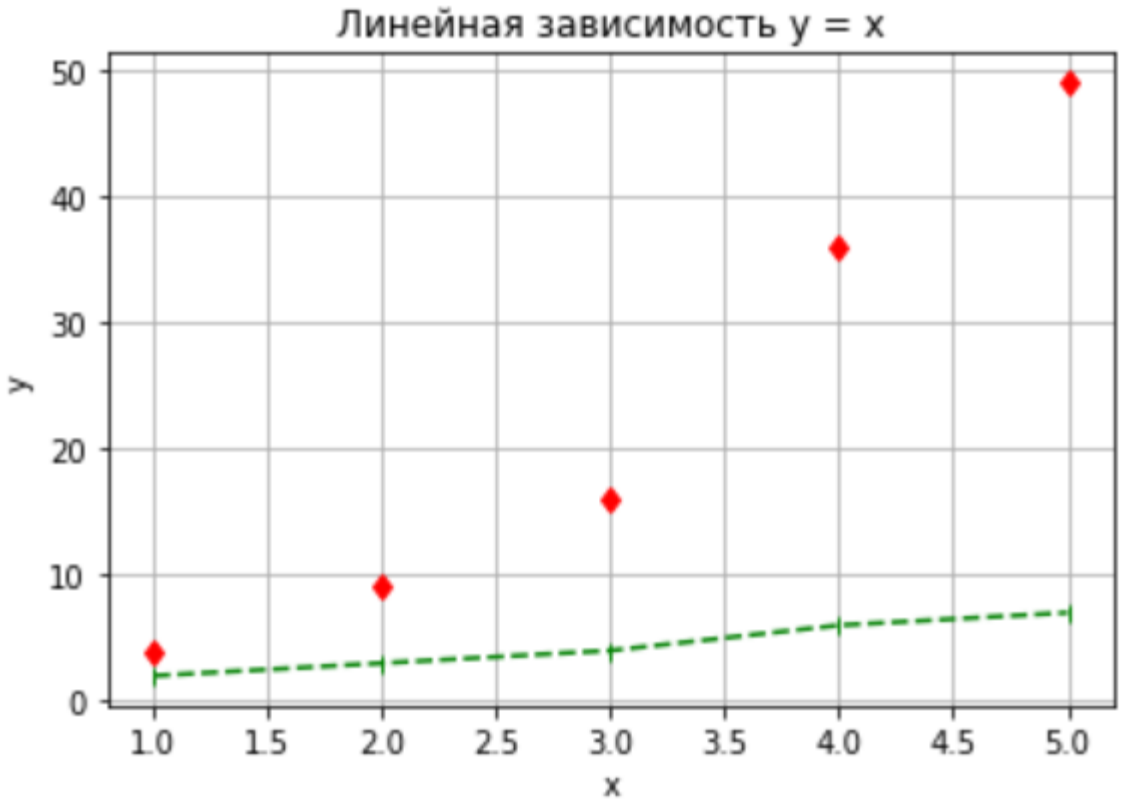
```
plt.plot(x, y, 'g|--', linewidth=2, markersize=16)
```



Попробуем нарисовать ещё один график в нашей системе координат!

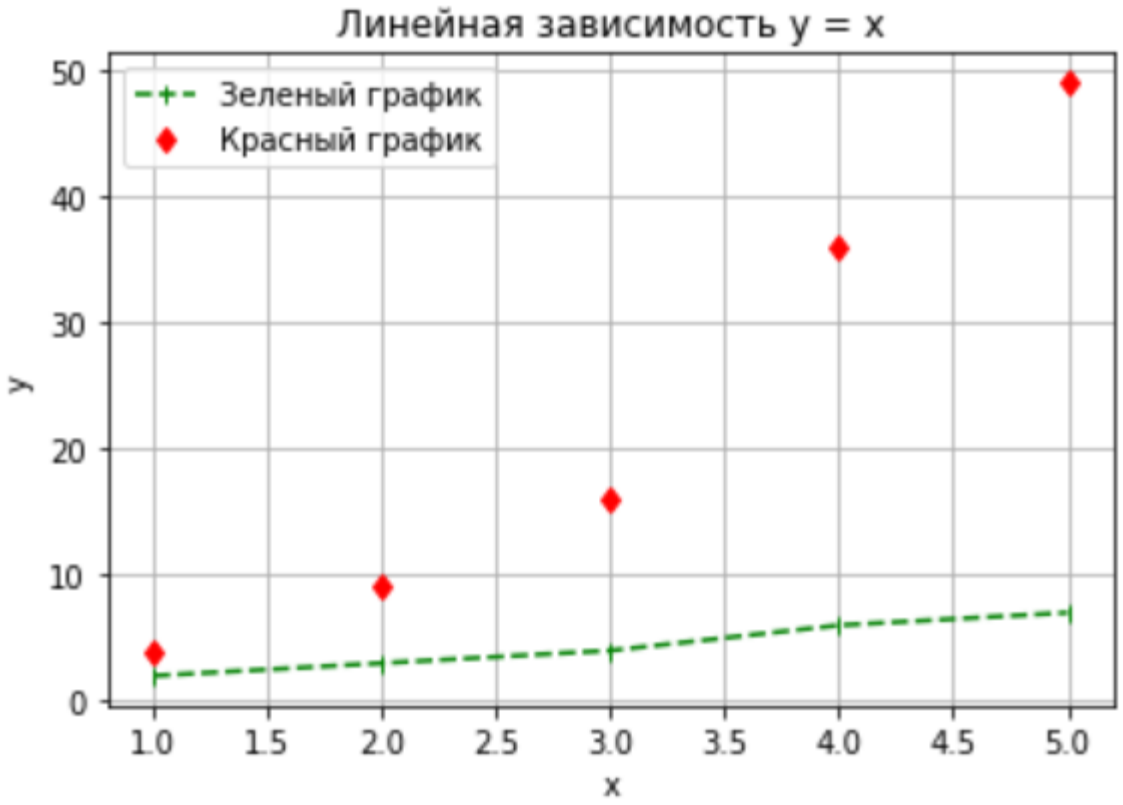
- x оставим таким же
- y возведём в квадрат
- нарисуем результат круглыми красными маркерами без линии

```
#функция np.power() возводит числа в указанную степень
y2 = np.power(y, 2)

# построение графика
plt.plot(x, y, '|--g', x, y2, 'rd')  #два x, два y, свои элемент
plt.title("Линейная зависимость y = x")
# ось абсцисс
plt.xlabel("x")
# ось ординат
plt.ylabel("y")
# включение отображения сетки
plt.grid()
```



Можно сделать это не одной функцией `plt.plot()`, а двумя подряд. Более того, это позволит нам дать имя обоим графикам и вывести легенду:

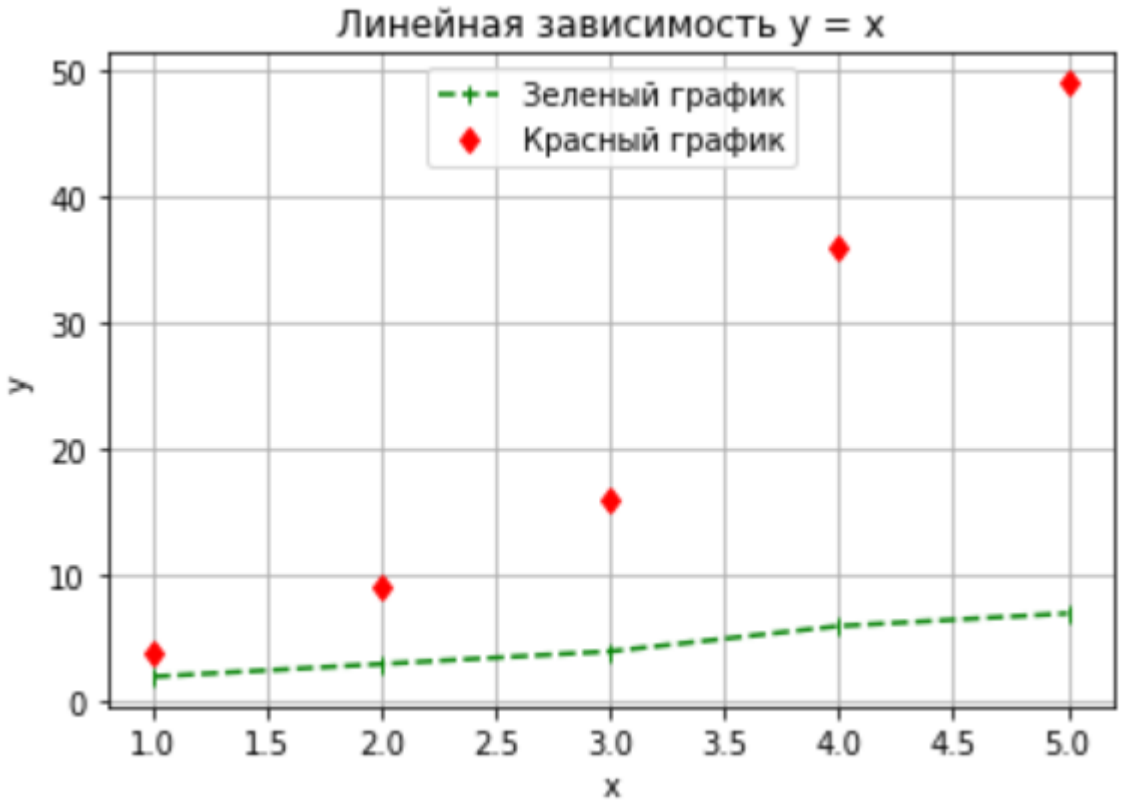
```
#построение графика
plt.plot(x, y, '|--g', label='Зеленый график')
plt.plot(x, y2, 'rd', label='Красный график')
#заголовок
plt.title("Линейная зависимость y = x")
#ось абсцисс
plt.xlabel("x")
#ось ординат
plt.ylabel("y")
#включение отображения сетки
plt.grid()
#добавляем легенду
plt.legend()
```



Сейчас легенда находится у нас в левом верхнем углу. Но мы можем изменить её позицию через аргумент `loc`. Полный список вариантов можно увидеть в документации, а сейчас поместим её в центр вверху:

```
#построение графика
plt.plot(x, y, '|--g', label='Зеленый график')
plt.plot(x, y2, 'rd', label='Красный график')
#заголовок
plt.title("Линейная зависимость y = x")
#ось абсцисс
plt.xlabel("x")
#ось ординат
plt.ylabel("y")
#включение отображения сетки
plt.grid()
```

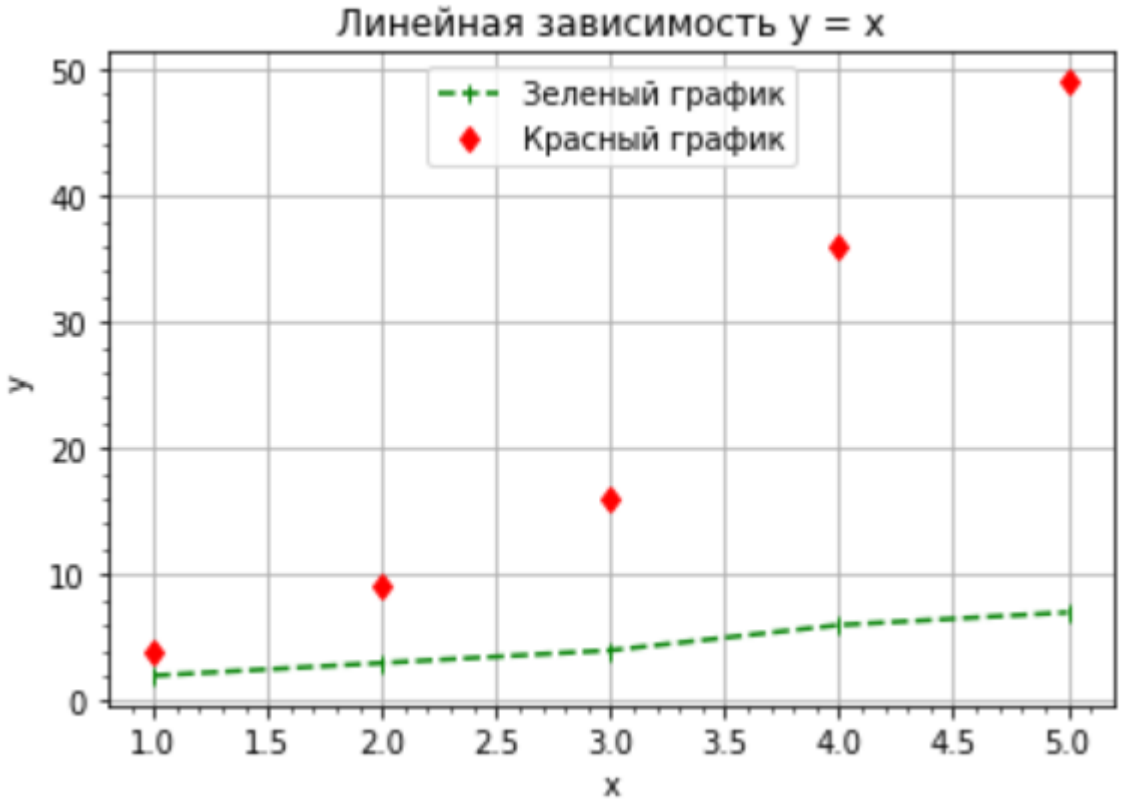
```
#добавляем легенду
plt.legend(loc='upper center')
```



Помимо сетки, можно добавить мелкие отсечки на сами оси:

```
#построение графика
plt.plot(x, y, '|--g', label='Зеленый график')
plt.plot(x, y2, 'rd', label='Красный график')
#заголовок
plt.title("Линейная зависимость y = x")
#ось абсцисс
plt.xlabel("x")
#ось ординат
plt.ylabel("y")
#включение отображения сетки
plt.grid()
#добавляем легенду
```

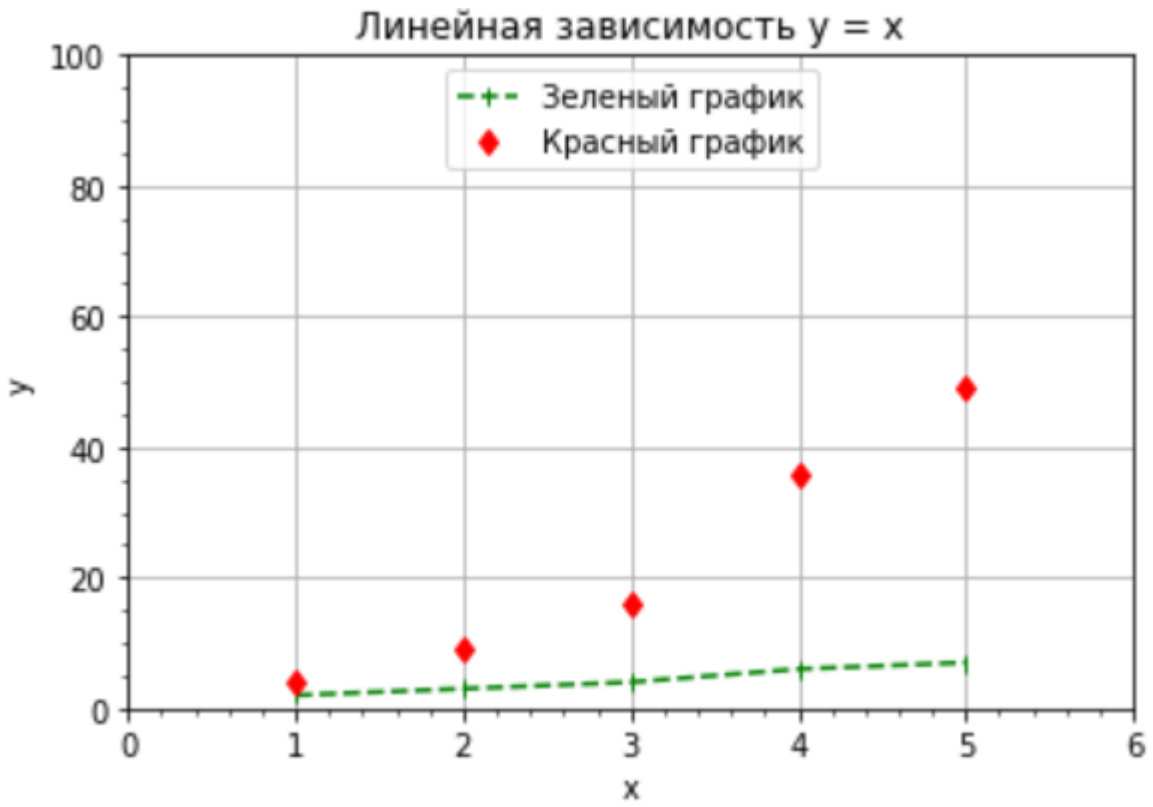
```
plt.legend(loc='upper center')

#добавляем мелкие отсечки
plt.minorticks_on()
```



Наконец, мы можем самостоятельно задать числовые границы осей:

```
plt.axis([0, 6, 0, 100])
#ось x - от 0 до 6#ось y - от 0 до 100# построение графика
plt.plot(x, y, '|--g', label='Зеленый график')
plt.plot(x, y2, 'rd', label='Красный график')
# заголовок
plt.title("Линейная зависимость y = x")
# ось абсцисс
plt.xlabel("x")
# ось ординат
plt.ylabel("y")
```

```
# включение отображение сетки
plt.grid()
# добавляем легенду
plt.legend(loc='upper center')
# добавляем мелкие отсечки
plt.minorticks_on()
```



Рисование через subplots

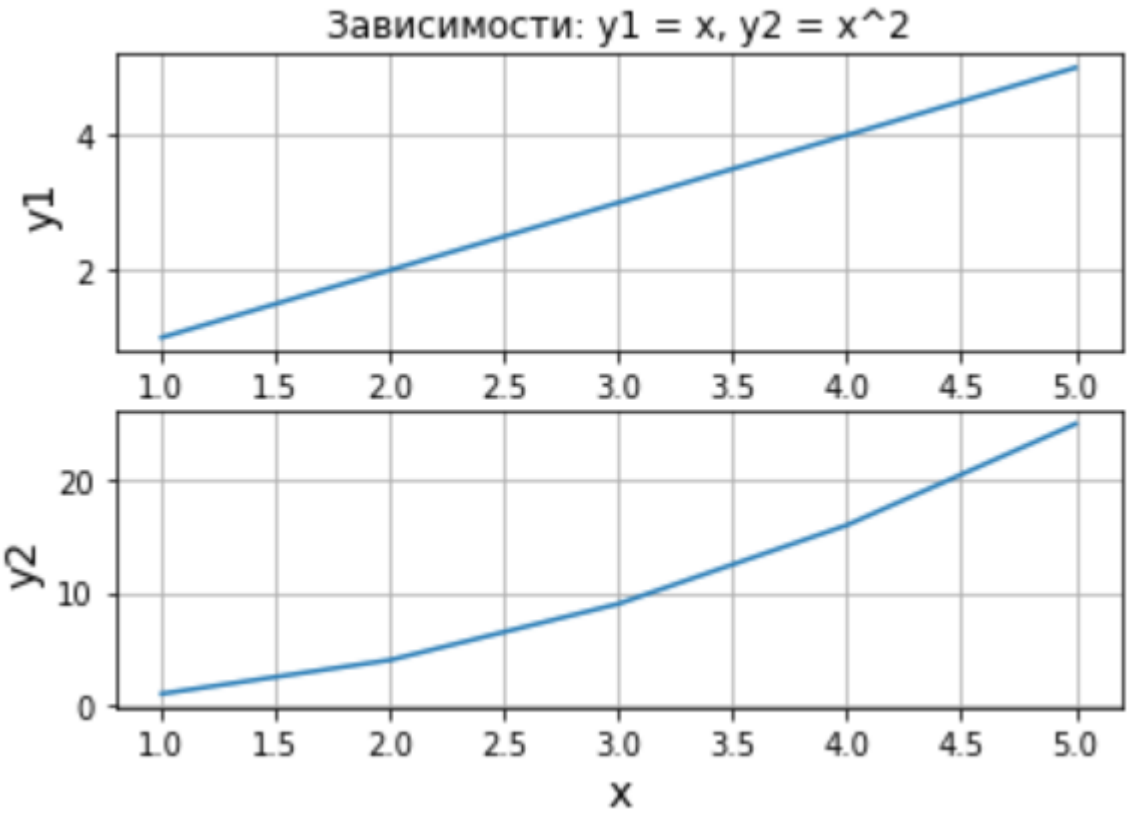
Всё это время мы помещали два графика в одну систему осей. Однако, как мы видели в начале, таких систем осей можно создать несколько!

Для начала опробуем функцию `plt.subplot()`. Она принимает на вход три числовых аргумента:

1. Количество рядов
2. Количество столбцов

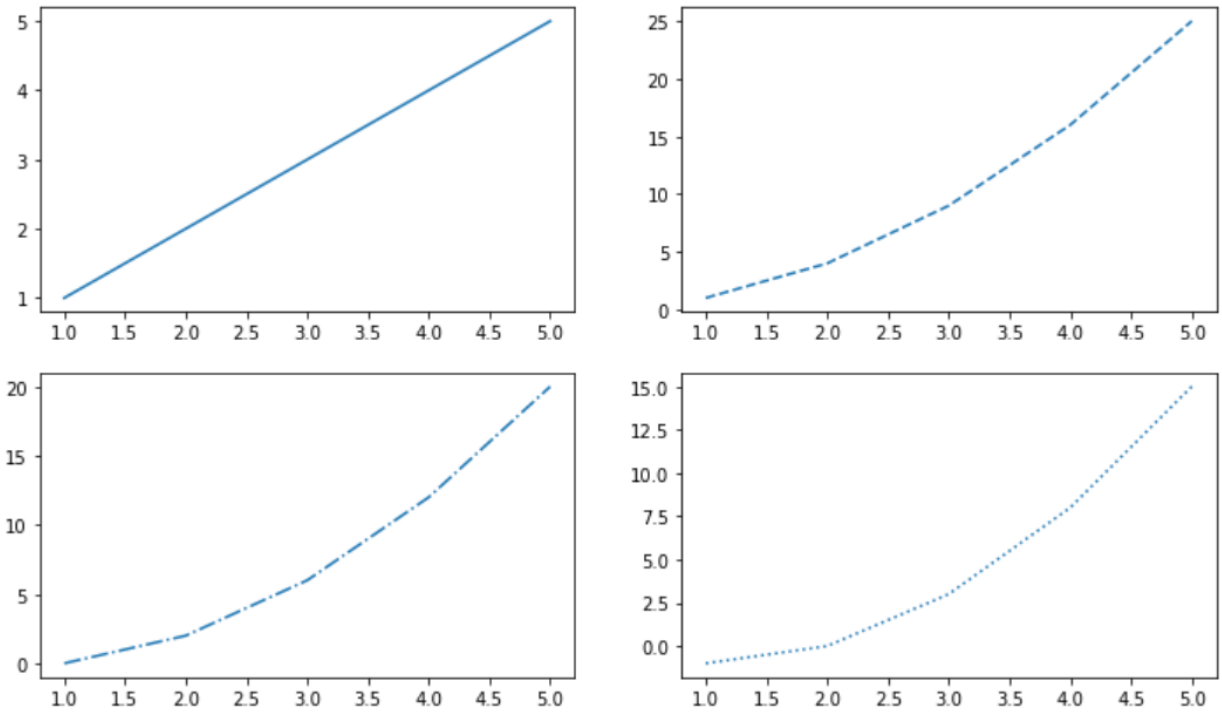
3. Индекс этого графика — начиная с 1 в верхнем левом углу и увеличиваясь при движении направо

```
y1 = np.array(x) #делаем копию значений по x
y2 = np.square(x) #возводим в квадрат
#у нас будет два ряда и один столбец
#рисуем первый график

plt.subplot(2, 1, 1)
#построение графика
plt.plot(x, y1)
#заголовок
plt.title("Зависимости: y1 = x, y2 = x^2")
#ось ординат
plt.ylabel("y1", fontsize=14) #fontsize задаёт размер шрифта
#для экономии местам ось абсцисс подпишем только на втором графи
#включение отображения сетки
plt.grid(True)

#а теперь второй график
plt.subplot(2, 1, 2)
#построение графика
plt.plot(x, y2)
#ось абсцисс
plt.xlabel("x", fontsize=14)
#ось ординат
plt.ylabel("y2", fontsize=14)
# включение отображение сетки
plt.grid(True)
```



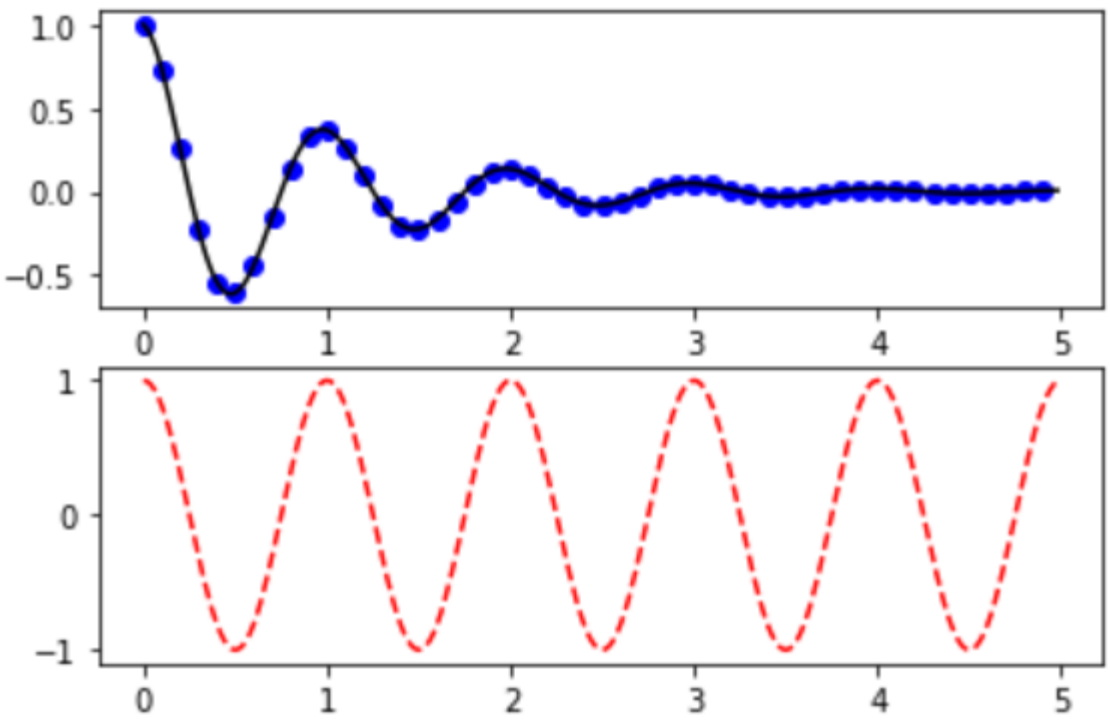
Теперь опробуем `plt.subplots()` схожим образом, как мы это делали в начале:

```
fig, axs = plt.subplots(2, 2, figsize=(12, 7)) #два ряда, два ст

#работаем со списком осей в переменной axs
#обратите внимание на индексы
axs[0, 0].plot(x, y1, '-') #график слева вверху
axs[0, 1].plot(x, y2, '--') #график справа вверху
axs[1, 0].plot(x, y2 - y1, '-.') #график слева внизу
axs[1, 1].plot(x, y2 - 2 * y1, ':') #график справа внизу
```



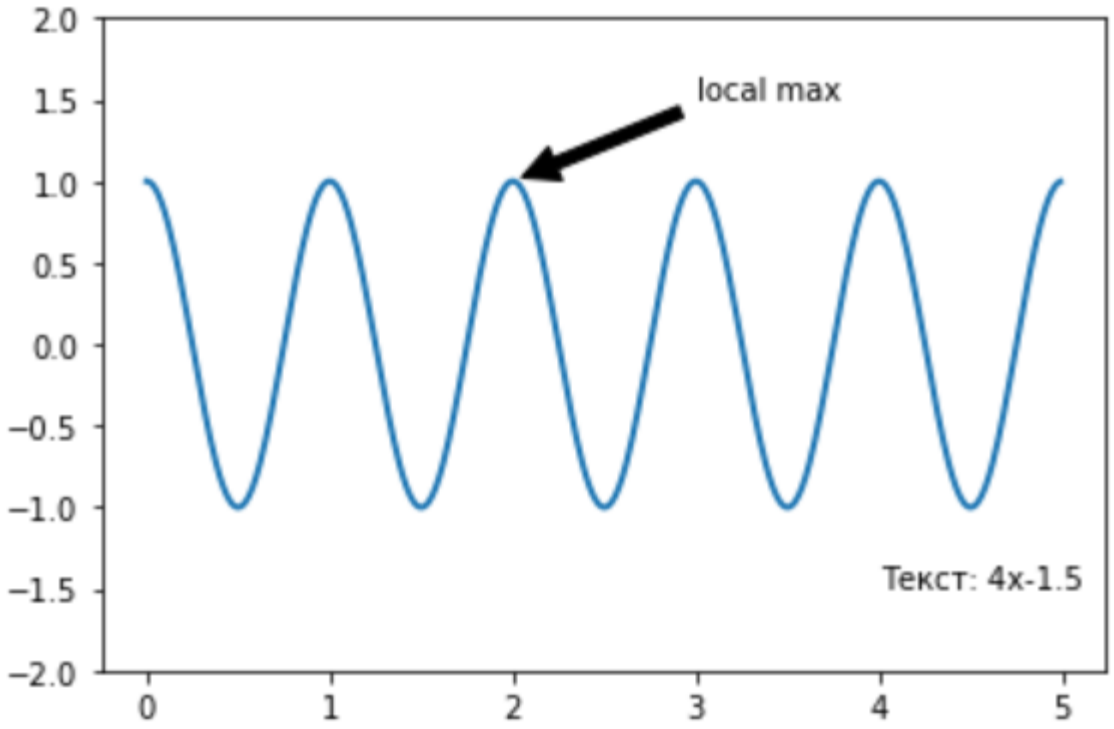
В обычном `plt.subplot()` можно писать аргументы не через запятую, а подряд:

```
#делаем функцию угасающего колебания
def f(t):
    return np.exp(-t) * np.cos(2 * np.pi * t)

#создаём последовательности чисел, которые будут лежать на осях
t1 = np.arange(0.0, 5.0, 0.1)
t2 = np.arange(0.0, 5.0, 0.02)

plt.figure()
plt.subplot(211)
plt.plot(t1, f(t1), 'bo', t2, f(t2), 'k') #синие круглые маркеры

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

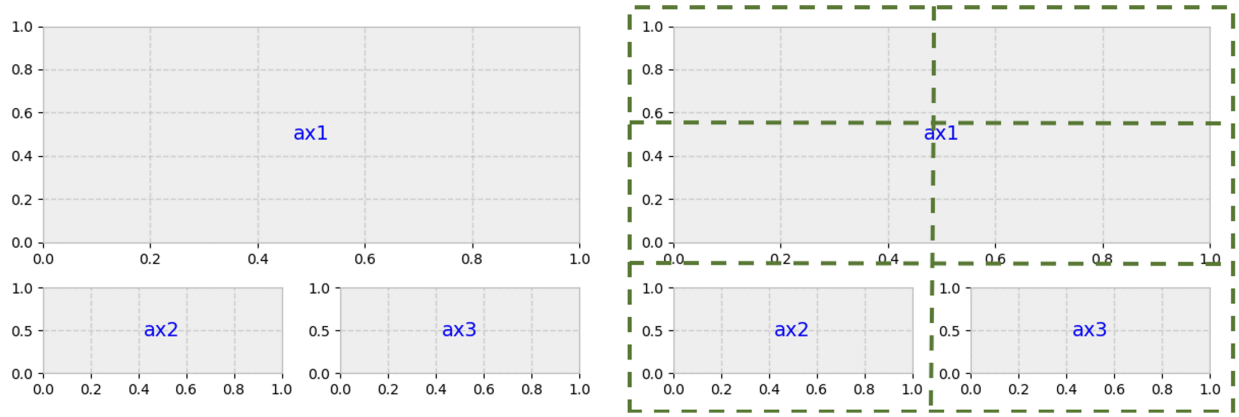


В общем-то с помощью `plt.subplot()` можно создать и один график. Сделаем это, а заодно продемонстрируем пару дополнительных функций:

```
#один ряд, один столбец, первый график
ax = plt.subplot(111)

t = np.arange(0.0, 5.0, 0.01)
s = np.cos(2 * np.pi * t)
line, = plt.plot(t, s, lw=2) #lw - толщина линии

#эта функция помогает делать аннотации
#по порядку: текст аннотации, координаты аннотируемой точки, коо
plt.annotate('local max', xy=(2, 1), xytext=(3, 1.5),
             arrowprops=dict(facecolor='black', shrink=0.05))

plt.ylim(-2, 2) #границы значений по y
plt.text(4, -1.5, 'Текст: 4x-1.5') #дополнительный текст - коорд
```

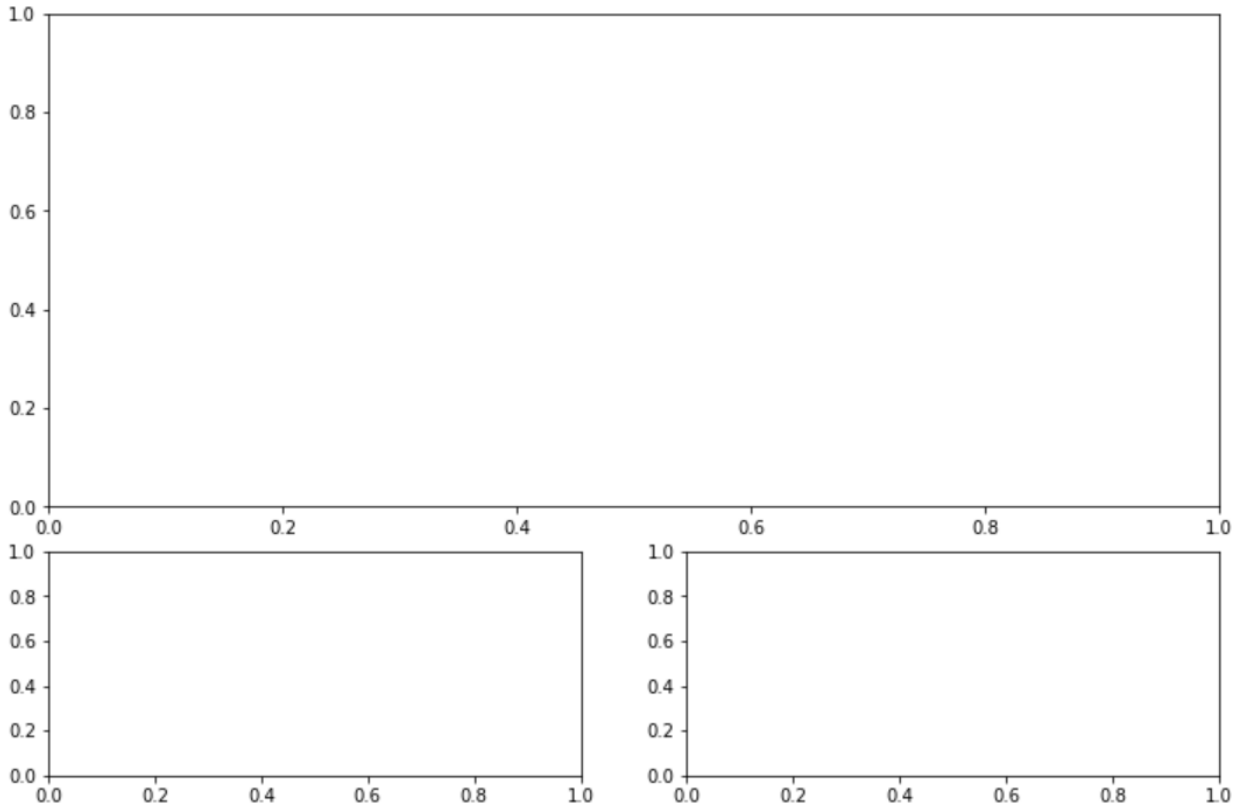


Также можно объединять несколько элементов по `subplots` в один большой элемент. Это полезно, если какой-то график мы хотим акцентировать путём увеличения его размера относительно остальных:




```
gridsize = (3, 2) #три ряда, два столбца
fig = plt.figure(figsize=(12, 8)) #задаём размер фигуры
#дальше используем plt.subplot2grid()
#colspan и rowspan - сколько рядов и столбцов занимает этот граф
```

```
ax1 = plt.subplot2grid(gridsize, (0, 0), colspan=2, rowspan=2) #
ax2 = plt.subplot2grid(gridsize, (2, 0)) #левый нижний график
ax3 = plt.subplot2grid(gridsize, (2, 1)) #правый нижний график
```



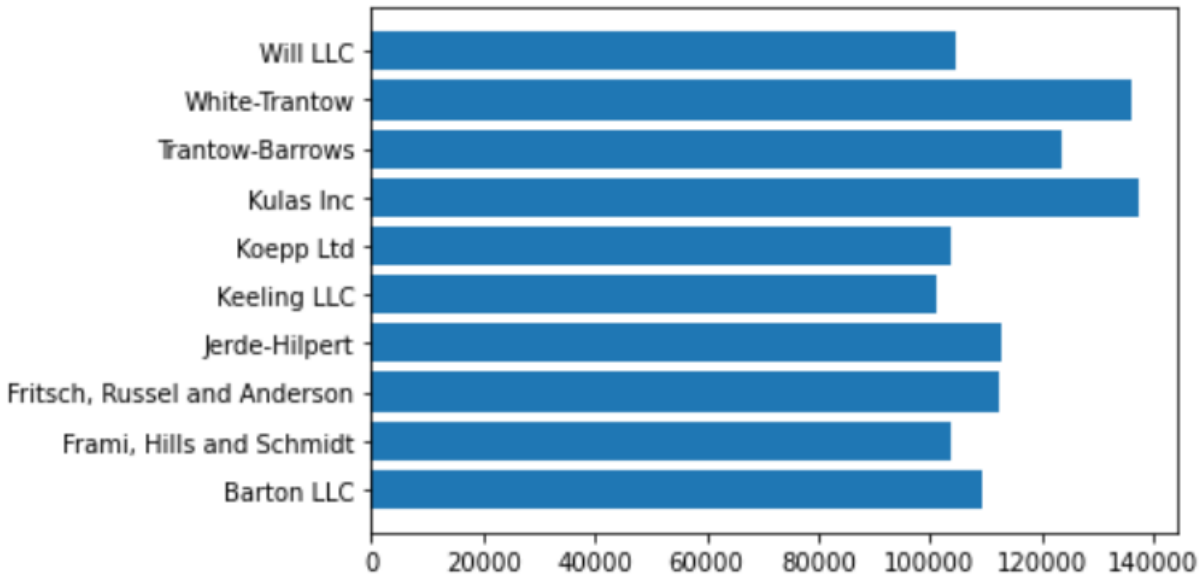
Чтобы отрегулировать отступы между графиками используйте функцию `plt.tight_layout()`. Её документация

Другие типы графиков

Один из наиболее простых и часто встречающихся типов графиков — столбиковая диаграмма. Нарисовать её можно, например, так:

```
#сделаем словарь
#ключи - названия компаний
#значения - их заработок

data = {'Barton LLC': 109438.50,
        'Frami, Hills and Schmidt': 103569.59,
```

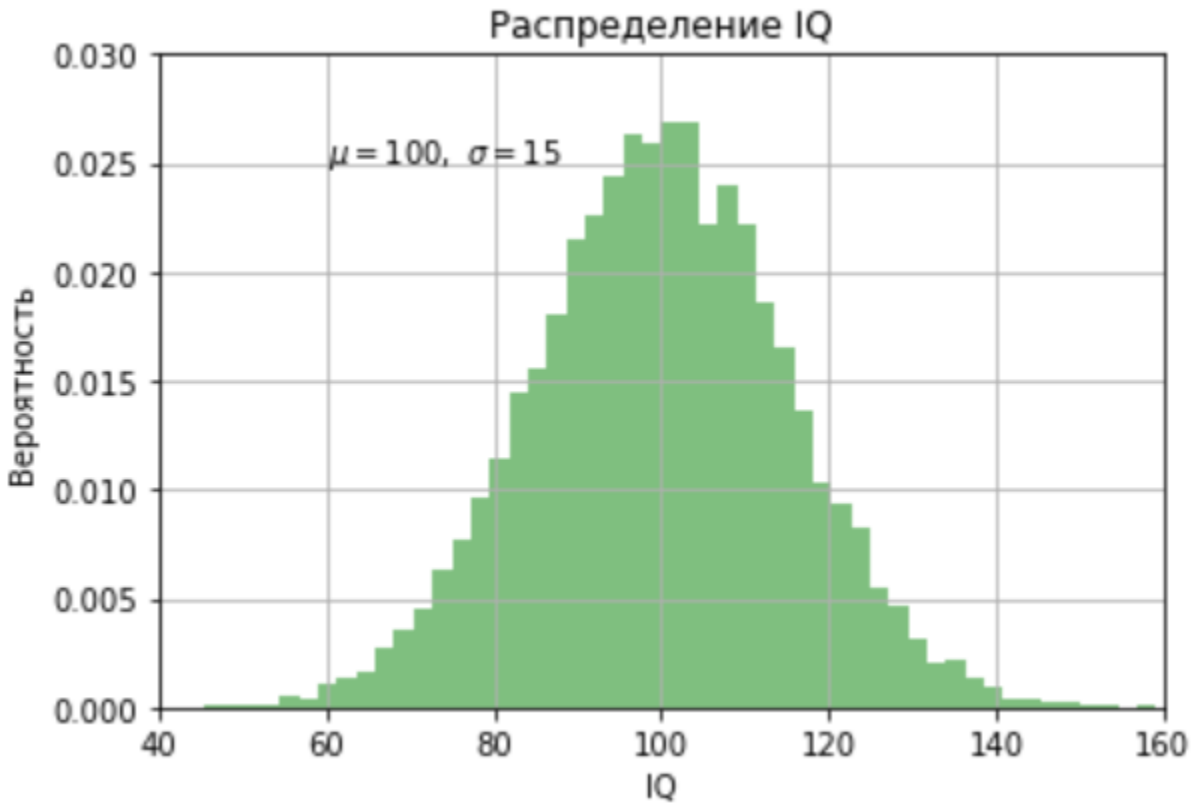
```
        'Fritsch, Russel and Anderson': 112214.71,
        'Jerde-Hilpert': 112591.43,
        'Keeling LLC': 100934.30,
        'Koepp Ltd': 103660.54,
        'Kulas Inc': 137351.96,
        'Trantow-Barrows': 123381.38,
        'White-Trantow': 135841.99,
        'Will LLC': 104437.60}

group_data = list(data.values()) #отдельно сохраним значения
group_names = list(data.keys()) #отдельно ключи

fig, ax = plt.subplots() #создаём один сабплот, нам нужен объект
ax.barh(group_names, group_data) #метод .barh() рисует горизонта
```



А теперь нарисуем гистограмму нормального распределения! Допустим, это будет распределение IQ в популяции.

```
mu, sigma = 100, 15 #среднее и стандартное отклонение шкалы IQ
x = mu + sigma * np.random.randn(10000) #создаём разброс данных,
```

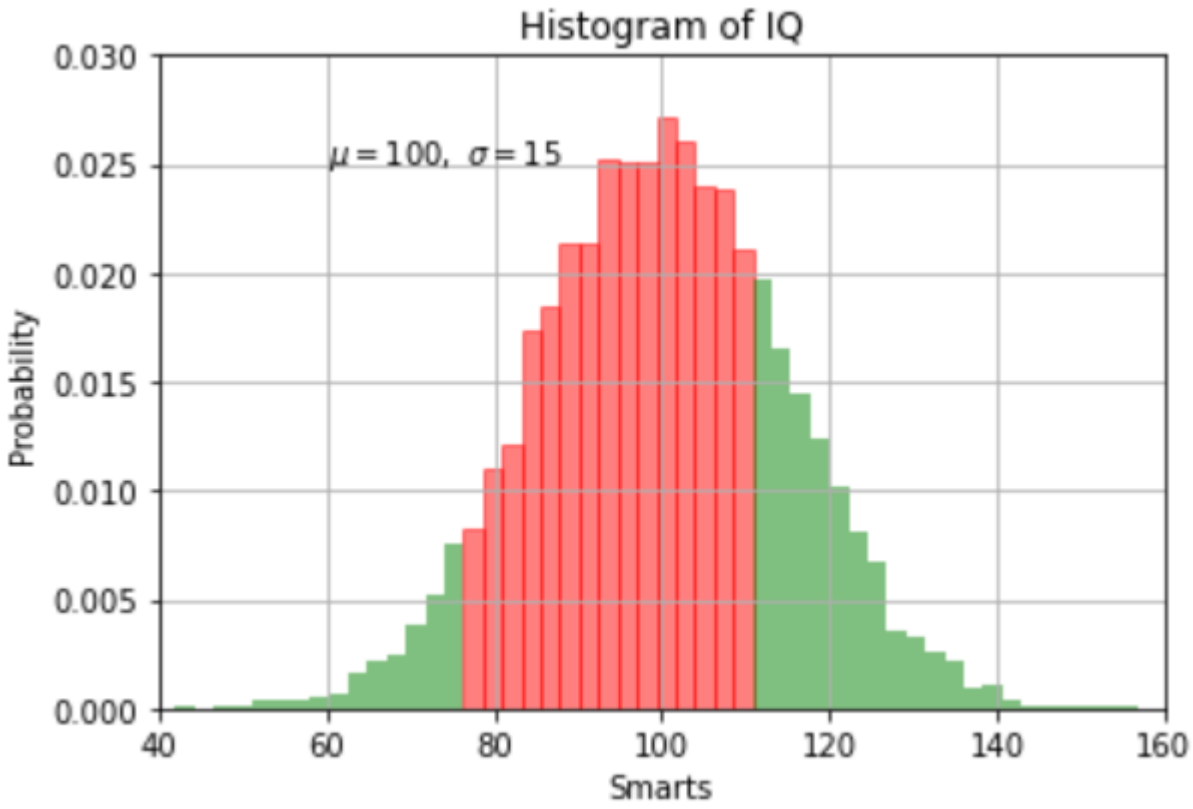
```
#гистограмма
#на вход: данные, число столбиков гистограммы, нормализация расп
#на выход: значения столбиков, их границы, полигоны для рисовани
n, bins, patches = plt.hist(x, 50, density=1, facecolor='g', alp

plt.xlabel('IQ')
plt.ylabel('Вероятность')
plt.title('Распределение IQ')
plt.text(60, .025, r'$\mu=100,\ \sigma=15$') #здесь используется
plt.axis([40, 160, 0, 0.03])
plt.grid()
```



Мы также можем раскрасить кусок гистограммы, выбрав несколько полигонов и задав им другой цвет:

```
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', alp
for patch in patches[15:30]: #для каждого полигона с 14 по 30
    patch.set_color('r') #раскрасить его в красный цвет

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()
```



Что есть ещё?

1. `plt.scatter()` — точечные диаграммы или диаграммы рассеяния
2. `plt.bar()` — столбиковые, но уже вертикальные

3. `plt.boxplot()` — ящики с усами
4. `plt.violinplot()` — скрипичные графики

> Seaborn

Возможно, вас не особо впечатлили возможности `matplotlib` для рисования статистических графиков. Хочется сделать красиво, быстро и интересно? Для такого была создана библиотека `seaborn` с уже готовым набором статистических графиков на любой вкус.

Особенности:

1. Эта библиотека основана на matplotlib, поэтому функции обеих можно сочетать в одном коде.
2. seaborn позволяет визуализировать данные, которые уже лежат в датафрейме pandas. Построить две линии на графике с синусами, создав два обычных массива, уже не получится.

Документация: https://seaborn.pydata.org/index.html

```
import seaborn as sns #sns - конвенциональный алиас
```

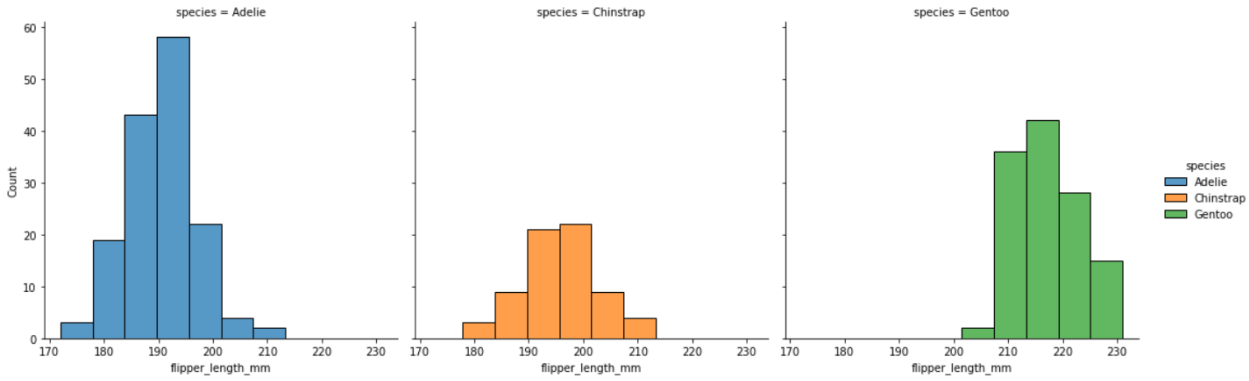
Поэкспериментируем с данными! В `seaborn` есть несколько наборов данных, вложенных в саму библиотеку — возьмём данные с пингвинами с помощью функции `sns.load_dataset()`:

```
penguins = sns.load_dataset("penguins")
```

Для начала нарисуем гистограмму длин ласт пингвинов, причём для каждого вида (species) пингвинов будет своя гистограмма. Можете подумать, как это сделать в `matplotlib`, а здесь это делается в одну строчку кода с использованием функции `sns.displot()`:

```
#data - наш датафрейм
#x - столбец, для которого мы рисуем гистограмму
```

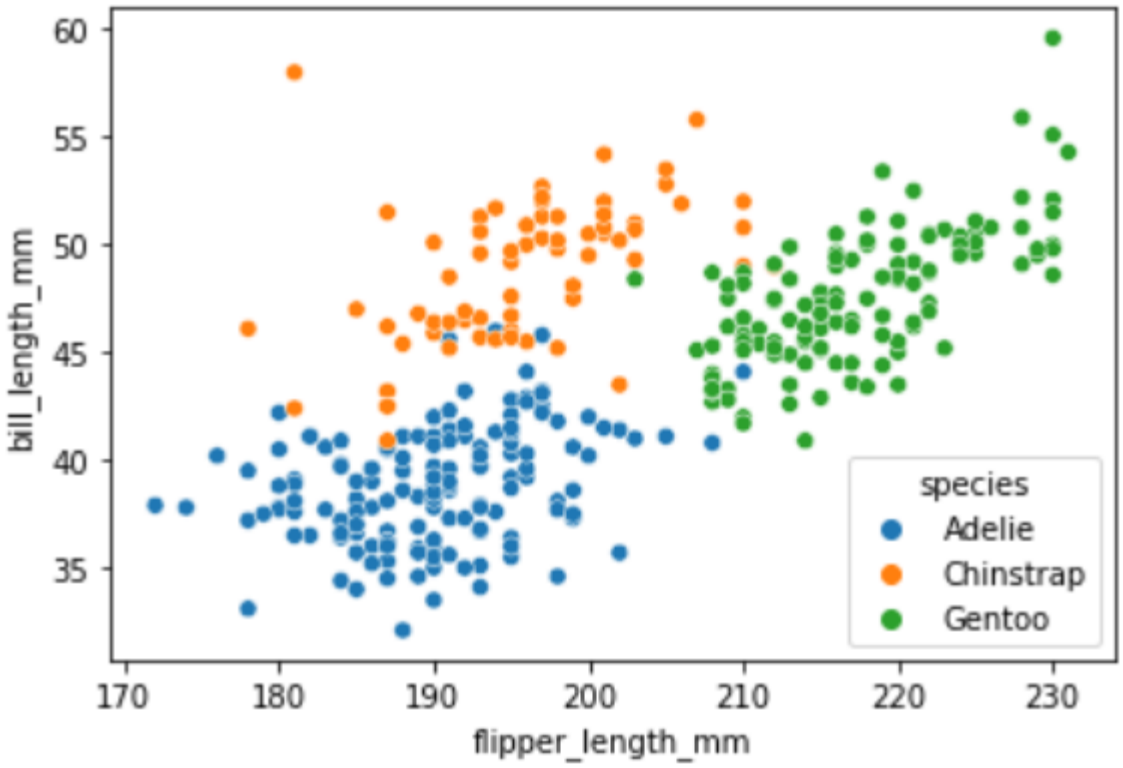
```
#hue - по какому столбцу раскрасить эти гистограммы
#col - по какому столбцу разделить эти гистограммы на три систем

sns.displot(data=penguins, x="flipper_length_mm", hue="species",

#получились три гистограммы разных цветов для каждого вида
```



Существует похожая функция `sns.dis t plot()`, разработчики seaborn пометили её как устаревшую, но она ещё работает и имеет несколько другой синтаксис и результат.

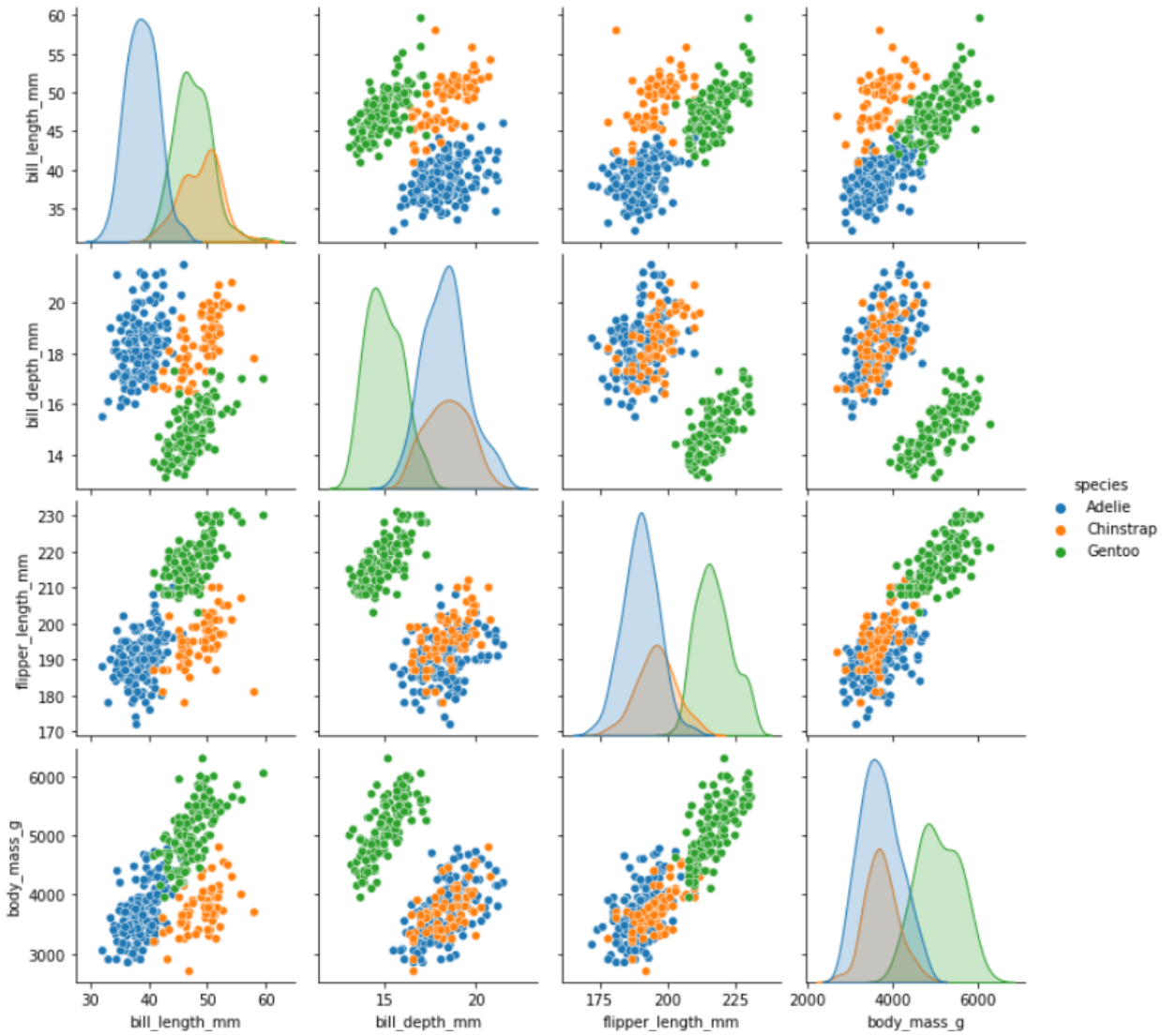
А теперь построим точечную диаграмму связи между длиной плавника и длиной клюва, раскрасив точки по видам пингвинов, применим `sns.scatterplot()`:

```
sns.scatterplot(data=penguins,
                x="flipper_length_mm", y="bill_length_mm",
                hue="species")
```

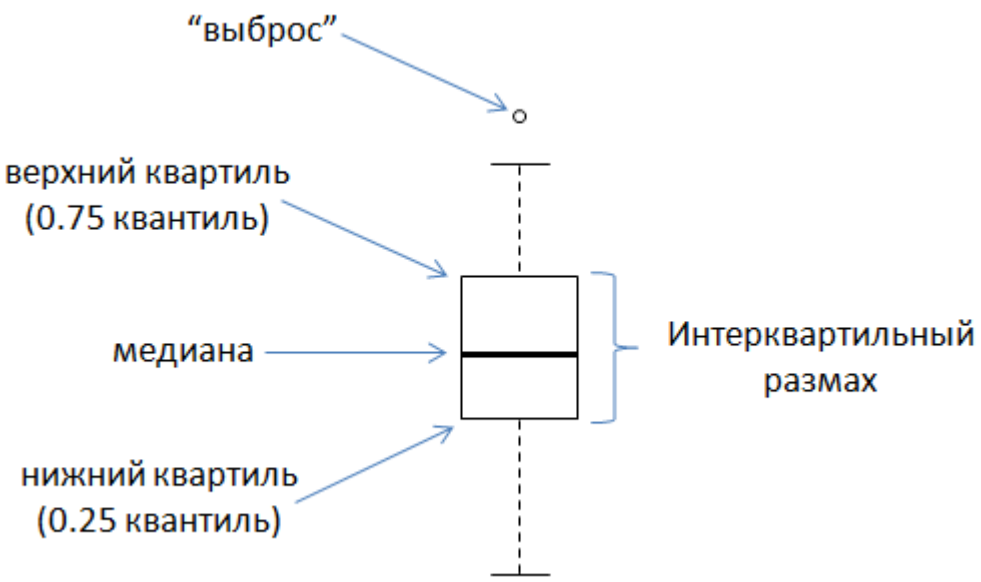


Отдельная возможность `seaborn` — распределения переменных и диаграммы рассеяния можно поместить на один график, причём для всех переменных сразу! Для этого используем функцию `sns.pairplot()`:

```
sns.pairplot(data=penguins, hue="species")

#тут будут видны диаграммы рассеяния для всех пар переменных в д
#а также распределения каждой переменной - только не в виде гист
#и всё это раскрашено по видам пингвинов
```

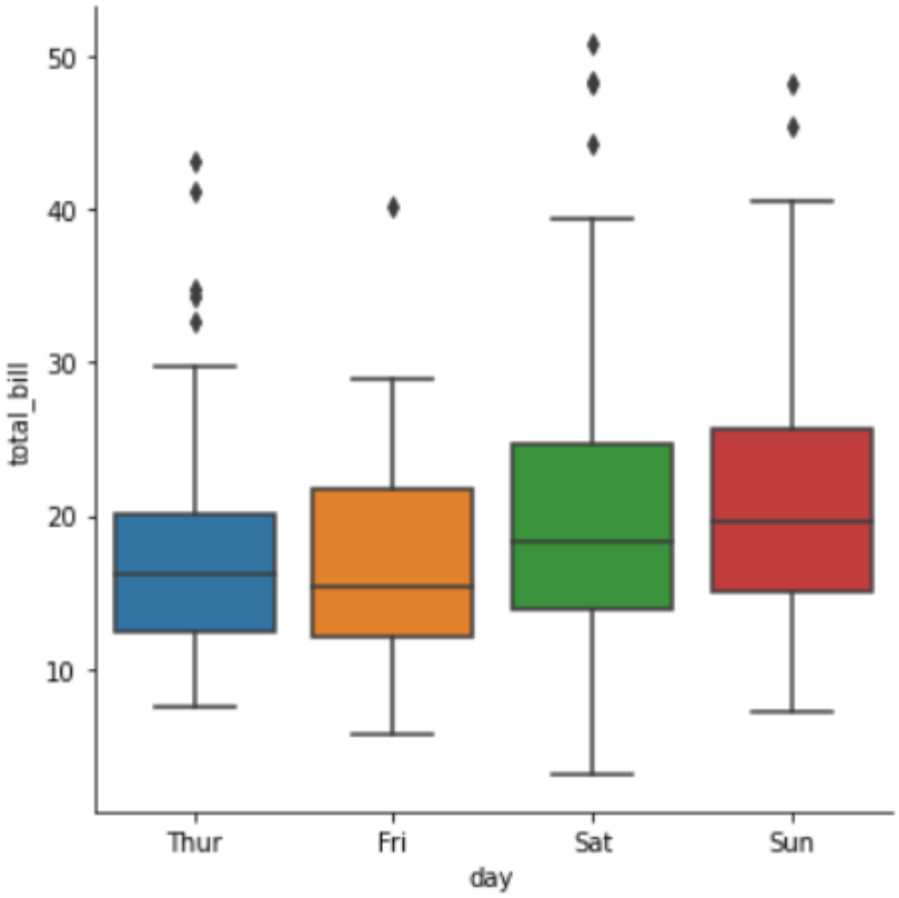


Под конец попробуем порисовать ящики с усами! Для этого подойдет функция `sns.boxplot()` или функция `sns.catplot()` с параметром `kind="box"`.



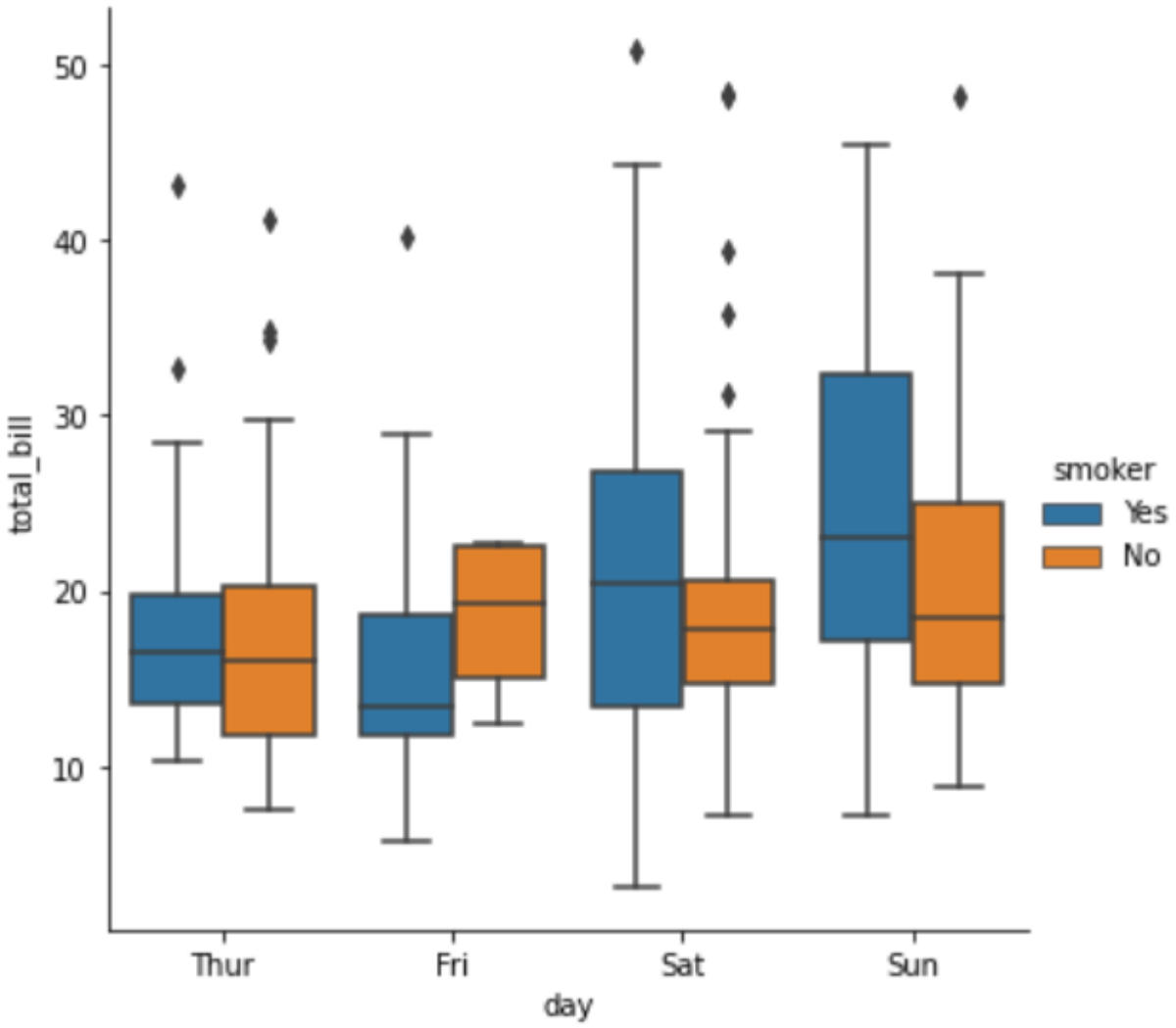
Возьмём для этого набор данных с денежными чеками. Попробуем визуализировать: как меняется распределение стоимости чеков по дням недели?

```
sns.catplot(x="day", y="total_bill", kind="box", data=tips)

#аргумент kind задаёт тип графика
#в этой функции вариантов несколько
```



Теперь раскрасим их в зависимости от того, курит ли плательщик или нет:

```
sns.catplot(x="day", y="total_bill", hue="smoker", kind="box", c
```



> Дополнительные материалы

- Работа с Join в Pandas
- как оформлять таблицы в Jupyter Notebook?
- как не надо строить графики
- 6 powerful libraries in Python for Data Visualization
- Top 5 Python Libraries For Data Visualization
- Гистограмма и ящик с усами на пальцах
- Статья о seaborn с примерами