Машинное обучение

Практика EDA. Часть 3

Материалы > Анализ и обработка данных

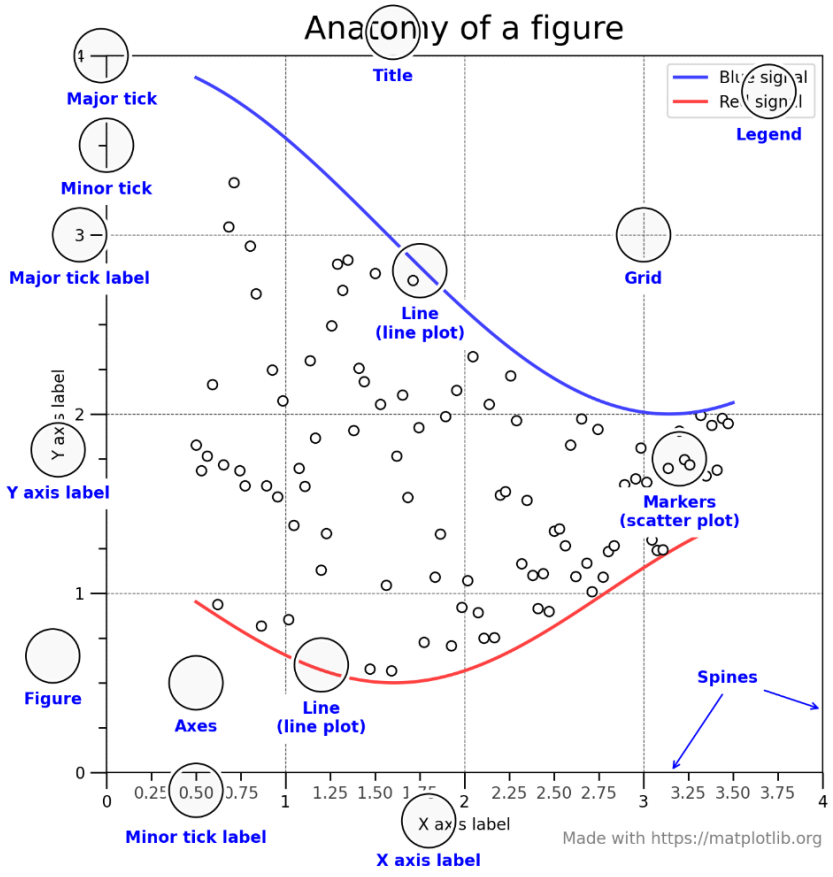
В третьей части занятия рассмотрим построение графиков в Matplotlib.

Продолжим работать в том же блокноте

График в Matplotlib

Теперь более детально рассмотрим некоторые из составляющих графика в Matplotlib. В частности, приведем его «анатомию».

анатомия графика в библиотеке matplotlib



Код для построения этого графика можно посмотреть здесь

И пройдемся по некоторым наиболее важным компонентам с помощью **функции**

np.linspace().

```
# создадим последовательность для оси x
x = np.linspace(0, 10, 100)
```

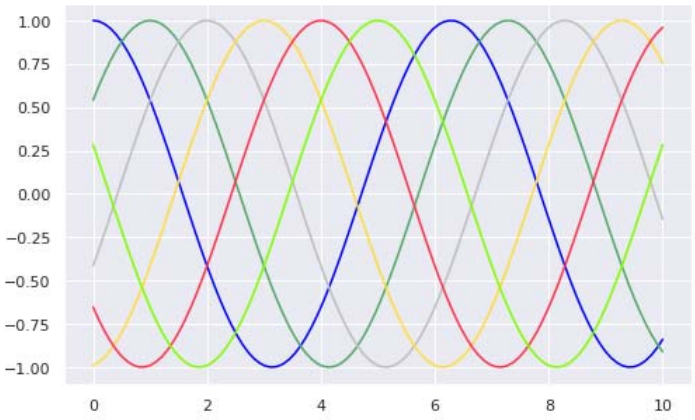
```
# снова зададим размеры графиков и одновременно установим стиль Seaborn
sns.set(rc = {'figure.figsize' : (8, 5)})
```

Стиль графика

Цвет графика

Создадим несколько графиков функции косинуса со сдвигом и зададим цвет каждого графика одним из доступных в Matplotlib способов.

```
plt.plot(x, np.cos(x - 0), color = 'blue')        # по названию
plt.plot(x, np.cos(x - 1), color = 'g')           # по короткому названию (rgbcmyk)
plt.plot(x, np.cos(x - 2), color = '0.75')        # оттенки серого от 0 до 1
plt.plot(x, np.cos(x - 3), color = '#FFDD44')     # HEX код (RRGGBB от 00 до FF)
plt.plot(x, np.cos(x - 4), color = (1.0,0.2,0.3)) # RGB кортеж, значения от 0 до 1
plt.plot(x, np.cos(x - 5), color = 'chartreuse'); # CSS название цветов
```

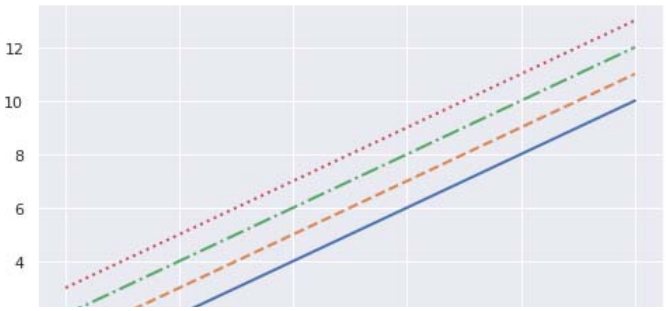


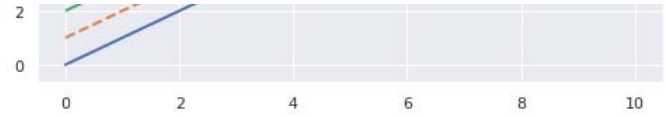
В документации можно найти более полный перечень названий цветов.

Тип линии графика

Посмотрим на различные типы линии.

```
plt.plot(x, x + 0, linestyle = 'solid', linewidth = 2)
plt.plot(x, x + 1, linestyle = 'dashed', linewidth = 2)
plt.plot(x, x + 2, linestyle = 'dashdot', linewidth = 2)
plt.plot(x, x + 3, linestyle = 'dotted', linewidth = 2);
```

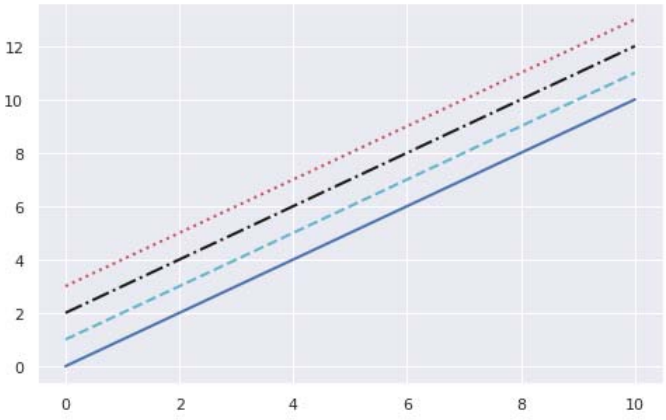




Строка форматирования

Цветом и типом линии можно также управлять с помощью строки форматирования (format string).

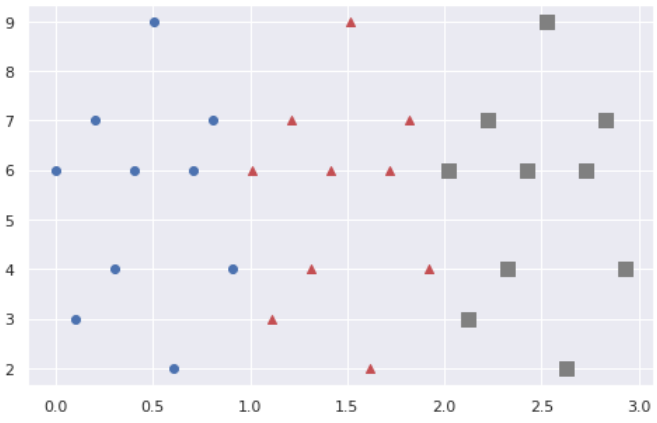
```
plt.plot(x, x + 0, '-b', linewidth = 2)  # сплошная синяя линия (по умолчанию)
plt.plot(x, x + 1, '--c', linewidth = 2) # штриховая линия цвета морской волны (cyan)
plt.plot(x, x + 2, '-.k', linewidth = 2) # черная (key) штрихпунктирная линия
plt.plot(x, x + 3, ':r', linewidth = 2); # красная линия из точек
```



Стиль точечной диаграммы

```
# зададим точку отсчета
np.random.seed(42)
# и последовательность из 10-ти случайных целых чисел от 0 до 10
y = np.random.randint(10, size = 10)
```

```
# выведем первые 10 наблюдений в виде синих (b) кругов (o)
plt.scatter(x[:10], y, c = 'b', marker = 'o')
# выведем вторые 10 наблюдений в виде красных (r) треугольников (^)
plt.scatter(x[10:20], y, c = 'r', marker = '^')
# выведем третьи 10 наблюдений в виде серых (0.50) квадратов (s)
# дополнительно укажем размер квадратов s = 100
plt.scatter(x[20:30], y, c = '0.50', marker = 's', s = 100);
```



Стиль графика в целом

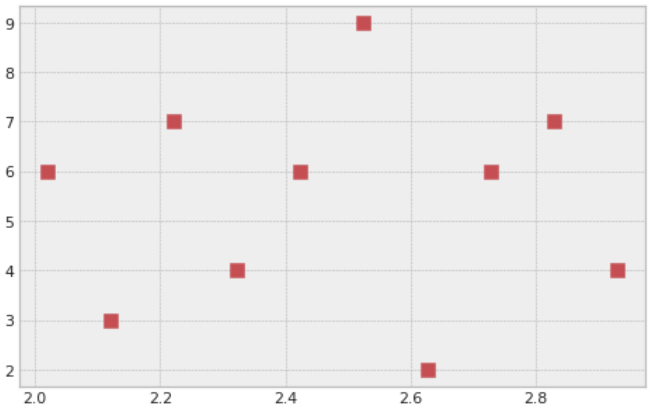
Помимо отдельных элементов графика мы можем управлять стилем графика в целом. Вначале посмотрим на доступные варианты.

```
plt.style.available
```

```
['Solarize_Light2',
 '_classic_test_patch',
 'bmh',
 'classic',
 'dark_background',
 'fast',
 'fivethirtyeight',
 'ggplot',
 'grayscale',
 'seaborn',
 'seaborn-bright',
 'seaborn-colorblind',
 'seaborn-dark',
 'seaborn-dark-palette',
 'seaborn-darkgrid',
 'seaborn-deep',
 'seaborn-muted',
 'seaborn-notebook',
 'seaborn-paper',
 'seaborn-pastel',
 'seaborn-poster',
 'seaborn-talk',
 'seaborn-ticks',
 'seaborn-white',
 'seaborn-whitegrid',
 'tableau-colorblind10']
```

Теперь используем стиль bmh.

```
# применим стиль bmh
plt.style.use('bmh')

# и создадим точечную диаграмму с квадратными красными маркерами размера 100
plt.scatter(x[20:30], y, s = 100, c = 'r', marker = 's');
```



Акроним bmh расшифровывается как Bayesian Methods for Hackers или «Байесовские методы для хакеров». Так называется онлайн-книга по байесовской статистике, в которой этот стиль используется для форматирования графиков.

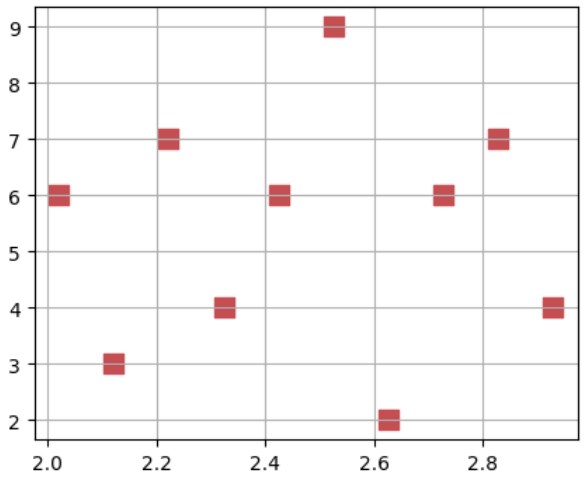
Важно отметить, что изменение стиля коснется всех последующих графиков в рамках действующей сессии. Изменить это можно, например, с помощью стиля default, т.е. стиля по умолчанию (такой стиль существует, хотя и не значится в списке выше).

```
# вернем блокнот к "заводским" настройкам (стиль default)
# такой стиль тоже есть, хоть он и не указан в перечне plt.style.available
plt.style.use('default')

# дополнительно пропишем размер последующих графиков
matplotlib.rcParams['figure.figsize'] = (5, 4)
matplotlib.rcParams['figure.figsize']
```

```
[5.0, 4.0]
```

Теперь все графики будут иметь базовый стиль Matplotlib (по сути, белый прямоугольник).

```
# дополним стиль по умолчанию сеткой и снова выведем график
plt.grid()
plt.scatter(x[20:30], y, s = 100, c = 'r', marker = 's');
```



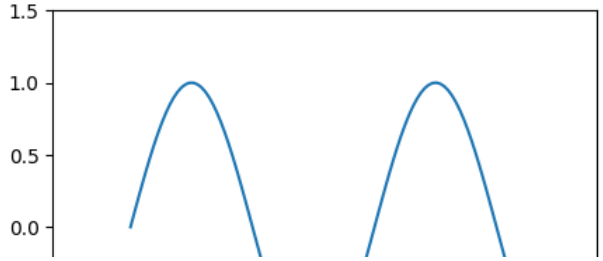
Пределы шкалы и деления осей графика

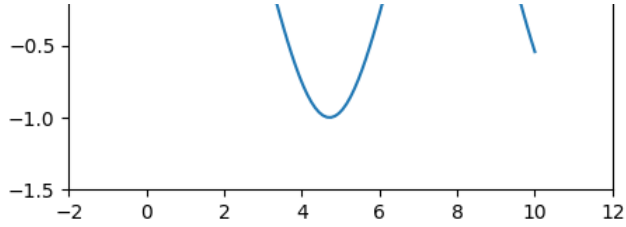
Пределы шкалы

Пределы шкалы по оси x и по оси y можно задавать двумя способами.

Способ 1. Использовать функции **plt.xlim()** и **plt.ylim()**.

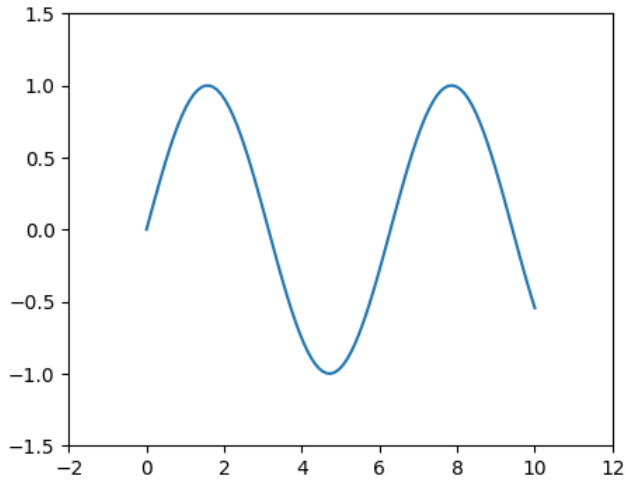
```
# выведем график функции синуса
plt.plot(x, np.sin(x))

# пропишем пределы по обеим осям
plt.xlim(-2, 12)
plt.ylim(-1.5, 1.5);
```





Способ 2. Кроме этого, мы можем воспользоваться **функцией plt.axis()**.

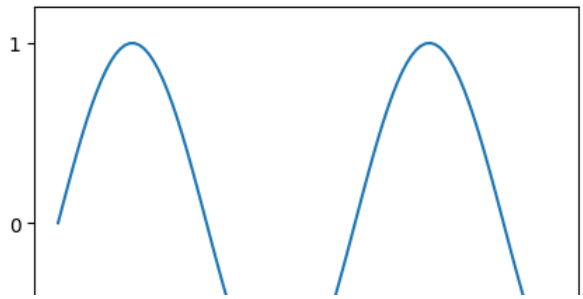
```
# выведем график функции синуса
plt.plot(x, np.sin(x))

# зададим пределы графика с помощью функции plt.axis()
# передадим параметры в следующей очередности: xmin, xmax, ymin, ymax
plt.axis([-2, 12, -1.5, 1.5]);
```

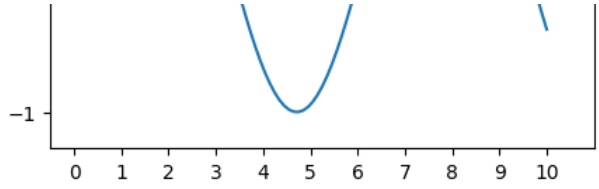


Деления

Иногда бывает необходимо принудительно изменить **деления** (tick) графика. Например, слишком большие числа могут не помещаться на шкале или для наглядности мы хотим изменить их формат. Сделать это можно с помощью функций **plt.xticks()** и **plt.yticks()**.

```
# построим синусоиду и зададим график ее осей
plt.plot(x, np.sin(x))
plt.axis([-0.5, 11, -1.2, 1.2])

# создадим последовательность от 0 до 10 с помощью функции np.arange()
# и передадим ее в функцию plt.xticks()
plt.xticks(np.arange(11))

# в функцию plt.yticks() передадим созданный вручную список
plt.yticks([-1, 0, 1]);
```

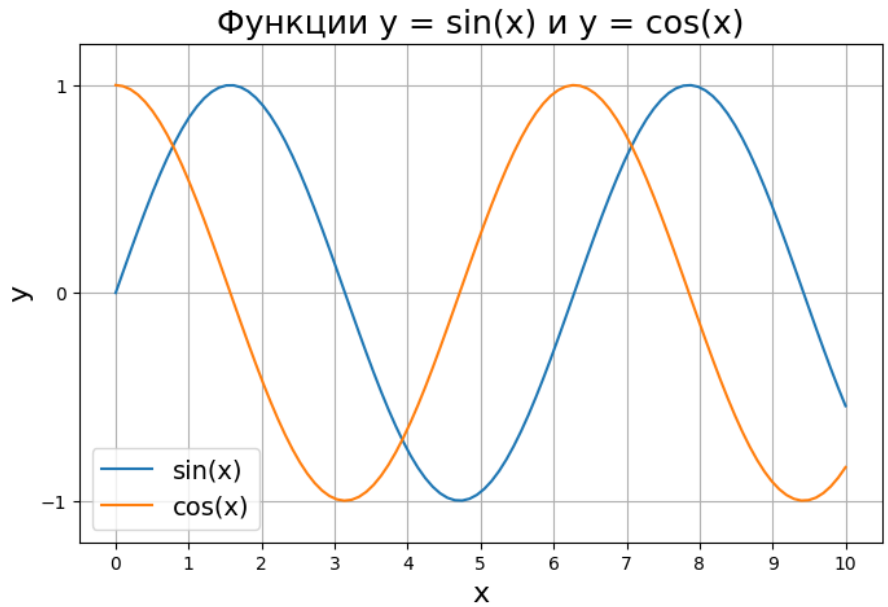




Подписи, легенда и размеры графика

Приведем несложный код, в котором соберем возможности по добавлению заголовка и подписей к графику, созданию легенды, а также управлению размерами различных элементов.

```
# зададим размеры отдельного графика (лучше указывать в начале кода)
plt.figure(figsize = (8,5))

# добавим графики синуса и косинуса с подписями к кривым
plt.plot(x, np.sin(x), label = 'sin(x)')
plt.plot(x, np.cos(x), label = 'cos(x)')

# выведем легенду (подписи к кривым) с указанием места на графике и размера шрифта
plt.legend(loc = 'lower left', prop = {'size': 14})

# добавим пределы шкал по обеим осям,
plt.axis([-0.5, 10.5, -1.2, 1.2])

# а также деления осей графика
plt.xticks(np.arange(11))
plt.yticks([-1, 0, 1])

# добавим заголовок и подписи к осям с указанием размера шрифта
plt.title('Функции y = sin(x) и y = cos(x)', fontsize = 18)
plt.xlabel('x', fontsize = 16)
plt.ylabel('y', fontsize = 16)

# добавим сетку
plt.grid()

# выведем результат
plt.show()
```

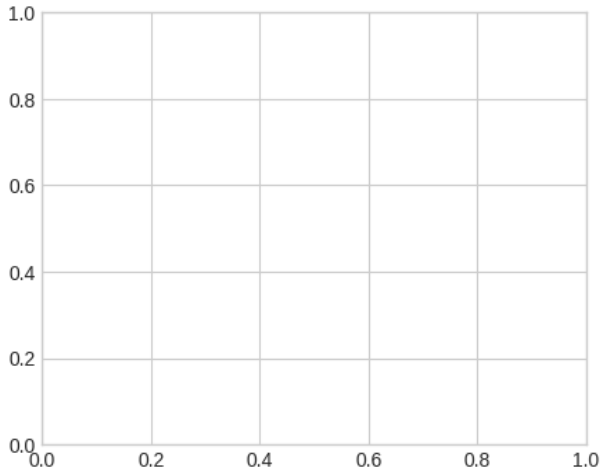


В целом комментарии к коду, я думаю, понятны и дальнейших пояснений не требуют. Замечу лишь, что для выведения легенды важны как **функция plt.legend()**, так и **параметр label** внутри функции, строящей график (в данном случае **plt.plot()**). Без обоих этих элементов легенда не отобразится.

plt.figure() и plt.axes()

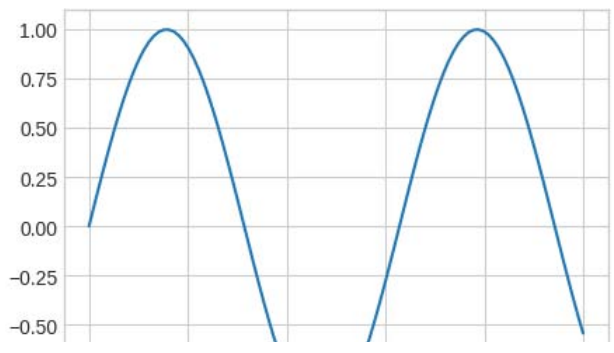
На прошлом занятии мы узнали, что Matplotlib поддерживает объектно-ориентированный подход к созданию графиков, и **plt.figure()** — это класс, который создает некий контейнер для хранения графиков, а **plt.axes()** строит графики внутри него. Рассмотрим простой пример создания объектов этих классов.

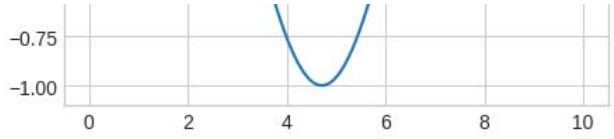
```
# создадим объект класса plt.figure()
fig = plt.figure()

# создадим объект класса plt.axes()
ax = plt.axes()
```



В данном случае объект **fig** мы не видим, изображение выше — объект **ax** (чтобы в этом убедиться, создайте объект класса **plt.figure()** без класса **plt.axes()**). Добавим к этому объекту кривую функции синуса.

```
# создадим объект класса plt.figure()
fig = plt.figure()

# создадим объект класса plt.axes()
ax = plt.axes()

# добавим синусоиду к объекту ax с помощью метода .plot()
ax.plot(x, np.sin(x));
```



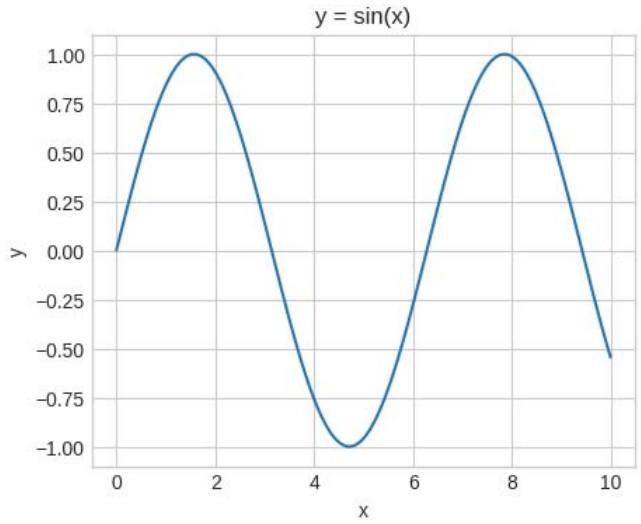


Для добавления других элементов графика можно использовать соответствующие методы, которые немного отличаются от принятых в стиле MATLAB.

- **plt.xlabel()** → **ax.set_xlabel()**
- **plt.ylabel()** → **ax.set_ylabel()**
- **plt.xlim()** → **ax.set_xlim()**
- **plt.ylim()** → **ax.set_ylim()**
- **plt.title()** → **ax.set_title()**

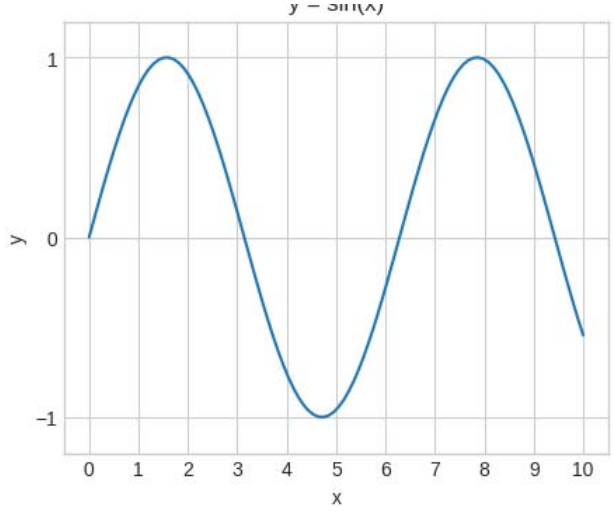
Продемонстрируем их применение.

```
fig = plt.figure()
ax = plt.axes()
ax.plot(x, np.sin(x))

# используем методы класса plt.axes()
ax.set_title('y = sin(x)')
ax.set_xlabel('x')
ax.set_ylabel('y');
```



Кроме этого, можно применить общий **метод .set()**, в параметрах которого прописать необходимые настройки.

```
fig = plt.figure()
ax = plt.axes()
ax.plot(x, np.sin(x))

# применим метод .set() и укажем необходимые параметры
ax.set(title = 'y = sin(x)',
       xlabel='x', ylabel = 'y',
       xlim = (-0.5, 10.5), ylim = (-1.2, 1.2),
       xticks = (np.arange(11)),
       yticks = [-1, 0, 1]);
```



Построение подграфиков

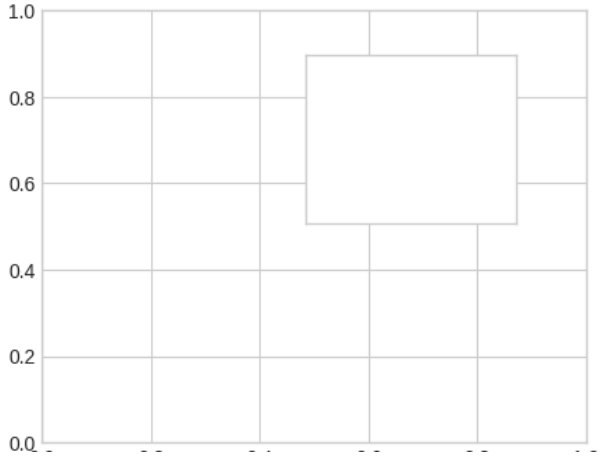
Остается рассмотреть важный вопрос создания **подграфиков** (subplots), то есть нескольких объектов класса **plt.axes()** внутри **plt.figure()**. Сделать это можно несколькими способами.

Способ 1. Создание вручную

Вначале создадим два объекта **plt.axes()**, один стандартный, второй — по координатам и размерам окна по следующей схеме **plt.axes([left, bottom, width, height])**. Другими словами, мы передаем координату левого нижнего угла и размеры по ширине и высоте.

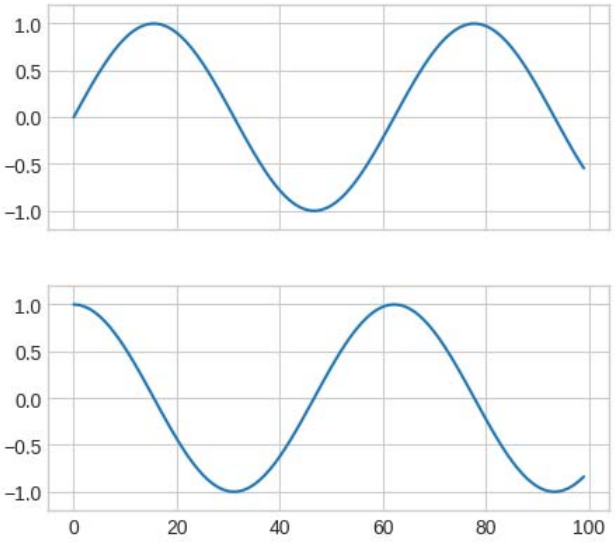
Например, создадим вложенный график, левый нижний угол которого будет находиться в центре основного графика [0.5, 0.5] и иметь размеры [0.3, 0.3] по шкале основного графика.

```
# создадим объект fig,
fig = plt.figure()

# стандартный подграфик
ax1 = plt.axes()

# и подграфик по следующим координатам и размерам
ax2 = plt.axes([0.5, 0.5, 0.3, 0.3])

# дополнительно покажем, как можно убрать деления на "вложенном" подграфике
ax2.set(xticks = [], yticks = []);
```



Посмотрим, как расположить графики один над другим.

```
# создадим объект класса plt.figure()
fig = plt.figure()

# зададим координаты угла [0.1, 0.6] и размеры [0.8, 0.4] верхнего подграфика,
# дополнительно зададим пределы шкалы по оси y и уберем шкалу по оси x
ax1 = fig.add_axes([0.1, 0.6, 0.8, 0.4],
                   ylim = (-1.2, 1.2),
                   xticklabels = [])

# добавим координаты угла [[0.1, 0.1] и размеры [0.8, 0.4] нижнего подграфика
ax2 = fig.add_axes([0.1, 0.1, 0.8, 0.4],
                   ylim = (-1.2, 1.2))

# выведем на них синусоиду и косинусоиду соответственно
ax1.plot(np.sin(x))
ax2.plot(np.cos(x));
```



Для того чтобы лучше понять, как получился такой график, сопоставьте координаты и размеры верхнего и нижнего графиков.

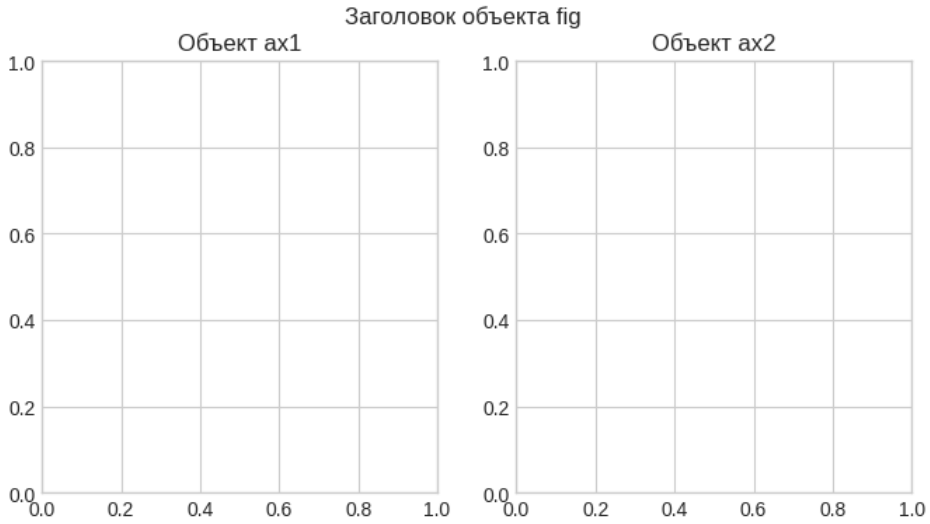
Способ 2. Метод .add_subplot()

Еще один способ создания подграфиков — применять к объекту класса **plt.figure()** метод **.add_subplot()**. По большому счету **.add_subplot()** принимает три основных параметра: количество столбцов, количество строк и индекс подграфика. Например, если мы хотим создать сетку из двух подграфиков в один ряд, количество строк будет равно одному, столбцов — двум (т.е. 1 x 2).

Потребуется, соответственно, два раза использовать метод **.add_subplot()**.

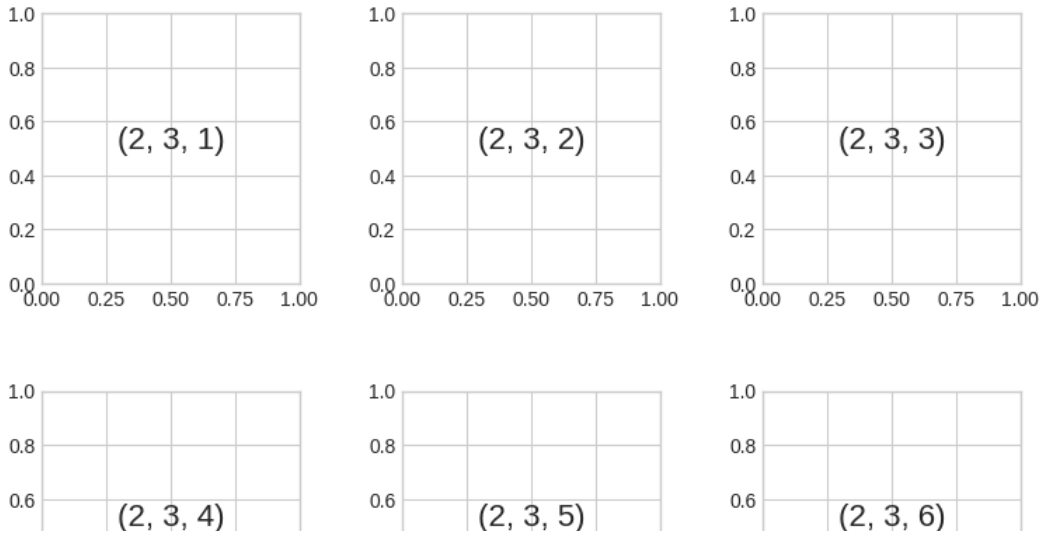
```
# создаем объект figure, задаем размер объекта,
fig = plt.figure(figsize = (8,4))
# указываем общий заголовок через метод .suptitle()
fig.suptitle('Заголовок объекта fig') # можно использовать plt.suptitle('Заголовок об

# внутри него создаем объект ax1, прописываем сетку из одной строки и двух столбцов
```

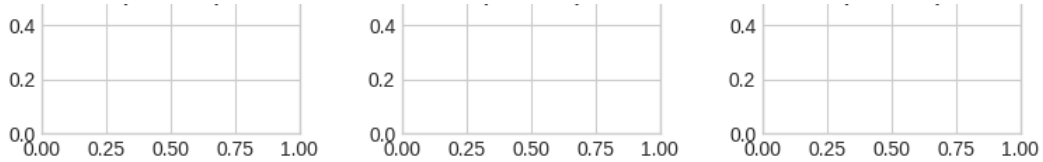
```
# и положение (индекс) ax1 в сетке
ax1 = fig.add_subplot(1, 2, 1)
# используем метод .set_title() для создания заголовка объекта ax1
ax1.set_title('Объект ax1')

# создаем и наполняем объект ax2
# запятые для значений сетки не обязательны, а заголовок можно передать параметром
ax2 = fig.add_subplot(122, title = 'Объект ax2')

plt.show()
```



Создать подграфики можно и с помощью цикла for.

```
# создадим объект figure и зададим его размер
fig = plt.figure(figsize = (9, 6))
# укажем горизонтальное и вертикальное расстояние между графиками
fig.subplots_adjust(hspace = 0.4, wspace = 0.4)

# в цикле от 1 до 6 (так как у нас будет шесть подграфиков)
for i in range(1, 7):
  # поочередно создадим каждый подграфик
  # первые два параметра задают сетку, в переменной i содержится индекс подграфика
  ax = fig.add_subplot(2, 3, i)
  # метод .text() позволяет написать текст в заданном месте подграфика
  ax.text(0.5, 0.5,       # разместим текст в центре
          str((2, 3, i)), # выведем параметры сетки и индекс графика
          fontsize = 16,  # зададим размер текста
          ha = 'center')  # сделаем выравнивание по центру
```



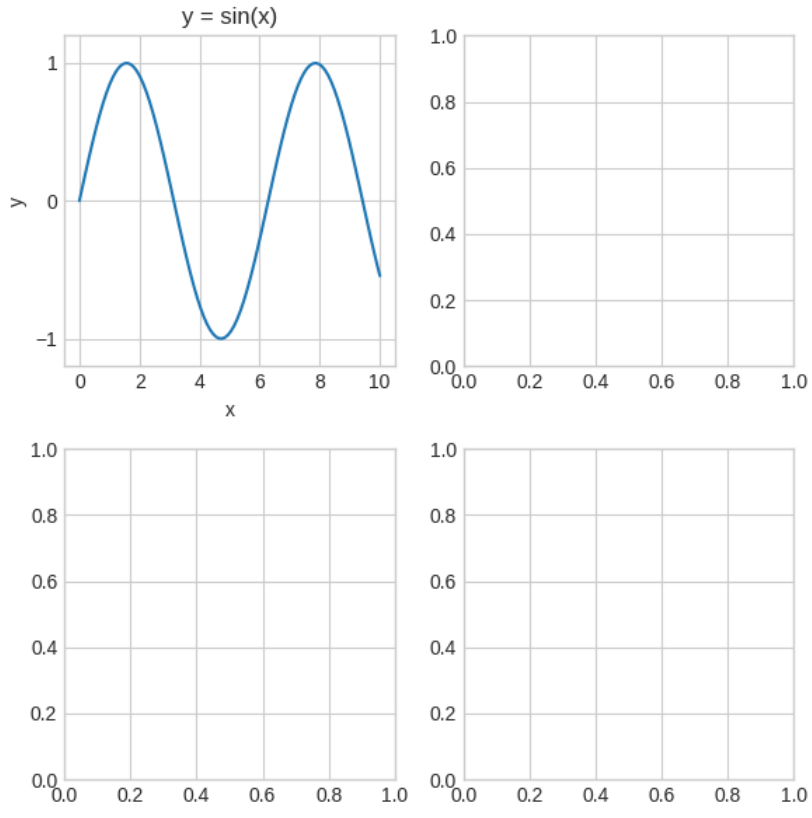


Способ 3. Функция plt.subplots()

С **функцией plt.subplots()** в принципе мы уже давно знакомы. Мы использовали ее на занятии по компьютерному зрению, а затем более подробно рассмотрели в ответах на вопросы к этому же занятию.

Функция **plt.subplots()** сразу создает фигуру и набор подграфиков. Приведем пример.

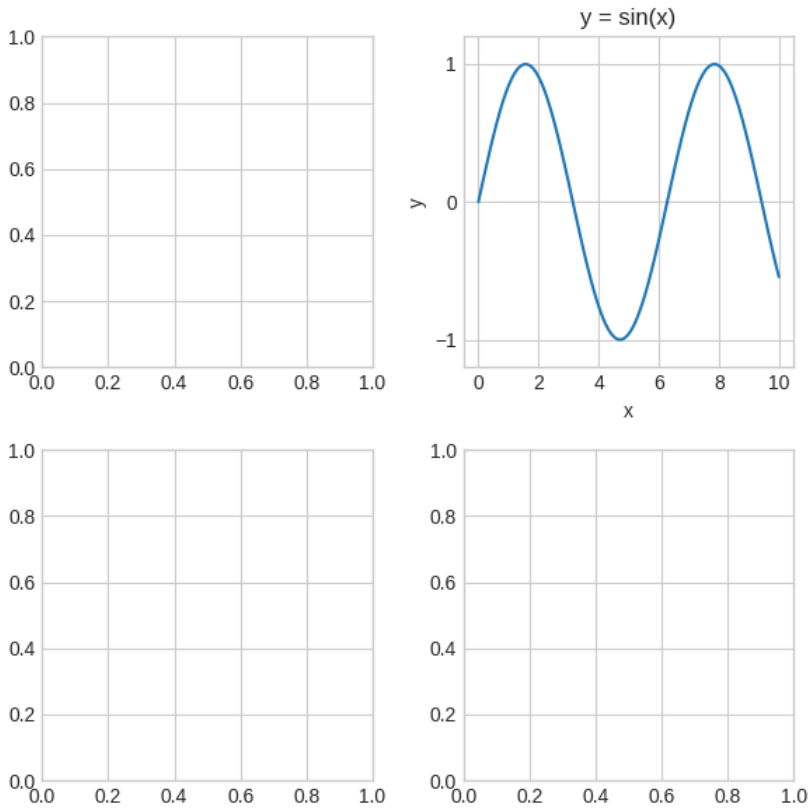
```
# создаем объекты fig и ax
# в параметрах указываем число строк и столбцов, а также размер фигуры
fig, ax = plt.subplots(nrows = 2, ncols = 2, figsize = (6,6))

# с помощью индекса объекта ax заполним левый верхний график
ax[0, 0].plot(x, np.sin(x))

# через метод .set() задаем параметры графика
ax[0, 0].set(title = 'y = sin(x)',
             xlabel = 'x', ylabel = 'y',
             xlim = (-0.5, 10.5), ylim = (-1.2, 1.2),
             xticks = (np.arange(0, 11, 2)),
             yticks = [-1, 0, 1])

plt.tight_layout();
```



Функция plt.tight_layout() корректирует внешние отступы (padding) фигуры и отступы между графиками таким образом, чтобы подграфики не перекрывали друг друга.

Подграфики можно сразу передать в соответствующие переменные. Мы уже делали так выше, когда строили совмещенный график гистограммы и boxplot.

```
# передадим подграфики в соответствующие переменные
# в первых внутренних скобках - первая строка, во вторых - вторая
fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize = (6, 6))

# поместим функцию np.sin(x) во второй столбец первой строки
ax2.plot(x, np.sin(x))
ax2.set(title = 'y = sin(x)',
        xlabel='x', ylabel = 'y',
        xlim = (-0.5, 10.5), ylim = (-1.2, 1.2),
        xticks = (np.arange(0, 11, 2)),
        yticks = [-1, 0, 1])

plt.tight_layout();
```



Еще один вариант использования **функции plt.subplots()**: «на ходу», т.е. в цикле for, заполнять объекты класса **plt.axes()** нужными графиками. Начнем с данных.

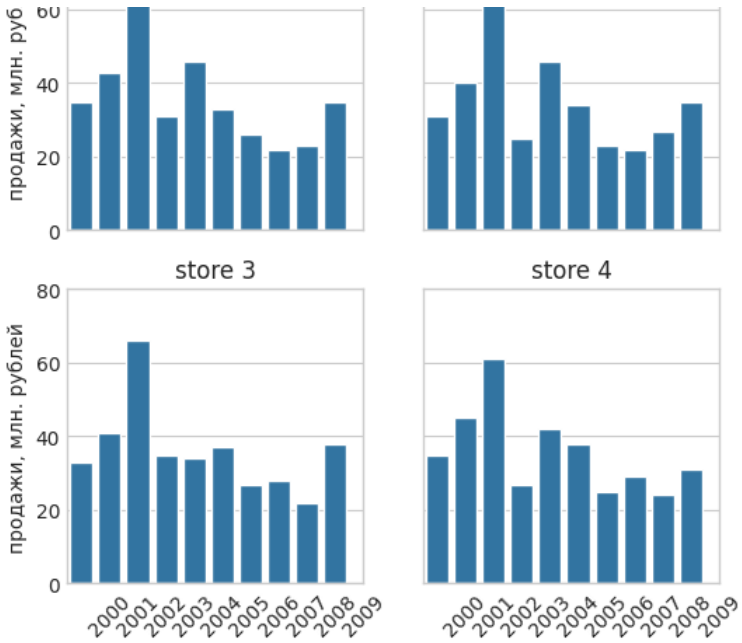
```
# возьмем данные о продажах в четырех магазинах
sales = pd.DataFrame({'year' :    [2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 200
                      'store 1' : [35, 43, 76, 31, 46, 33, 26, 22, 23, 35],
                      'store 2' : [31, 40, 66, 25, 46, 34, 23, 22, 27, 35],
                      'store 3' : [33, 41, 66, 35, 34, 37, 27, 28, 22, 38],
                      'store 4' : [35, 45, 61, 27, 42, 38, 25, 29, 24, 31]
                      })

# сделаем столбец year индексом
sales.set_index('year', inplace = True)

# посмотрим на данные
sales
```

```
      store 1  store 2  store 3  store 4
```

year				
2000	35	31	33	35
2001	43	40	41	45
2002	76	66	66	61
2003	31	25	35	27
2004	46	46	34	42
2005	33	34	37	38
2006	26	23	27	25
2007	22	22	28	29
2008	23	27	22	24
2009	35	35	38	31

Теперь создадим сетку из четырех подграфиков (2 x 2) и в цикле будем создавать столбчатую диаграмму продаж каждого магазина по годам.

```
# определимся с количеством строк и столбцов
nrows, ncols = 2, 2
# создадим счетчик для столбцов
col = 0

# создадим объекты fig и ax (в ax уже будет четыре подграфика)
# дополнительно, помимо размера, зададим общую шкалу по обеим осям
fig, ax = plt.subplots(nrows = nrows, ncols = ncols, figsize = (6,6), sharex = True, s

# в цикле пройдемся по строкам
for i in range(nrows):
  # затем во вложенном цикле - по столбцам
  for j in range(ncols):
    # для каждой комбинации i и j (координат подграфика) выведем столбчатую диаграмму
    # по оси x - годы, по оси y - соответстующий столбец (магазин)
    # в параметр ax мы передадим текущий подграфик с координатами
    sns.barplot(x = sales.index, y = sales.iloc[:, col], ax = ax[i, j])

    # дополнительно в методе .set() зададим заголовок подграфика,
    # уберем подпись к оси x и зададим единые для всех подграфиков пределы по оси y
    ax[i, j].set(title = sales.columns[col], xlabel = '', ylim = (0, 80))
    # укажем, количество делений шкалы (по сути, список от 1 до 10)
    ax[i, j].set_xticks(list(range(1, len(sales.index)+1)))
    # в качестве делений шкалы по оси x зададим годы и повернем их на 45 градусов
    ax[i, j].set_xticklabels(sales.index, rotation = 45)

    # общая шкала по осям предполагает общие деления, но не общую подпись,
    # чтобы подпись оси y была только слева от первого столбца, выведем ее при j == 0
    # (индекс j как раз отвечает за столбцы)
    if j == 0:
      ax[i, j].set_ylabel('продажи, млн. рублей')
    # в противном случае выведем пустую подпись
    else:
      ax[i, j].set_ylabel('')

    # обновим счетчик столбцов
    col += 1

# выведем результат
plt.show()
```



Мы так делали опять же на занятии по компьютерному зрению или, например, обсуждая центральную предельную теорему на занятии по модулю random.

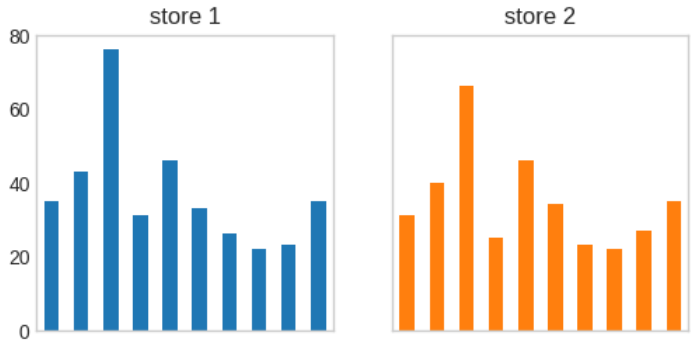
Обратите внимание, как используется объект ax с функциями библиотеки Seaborn.

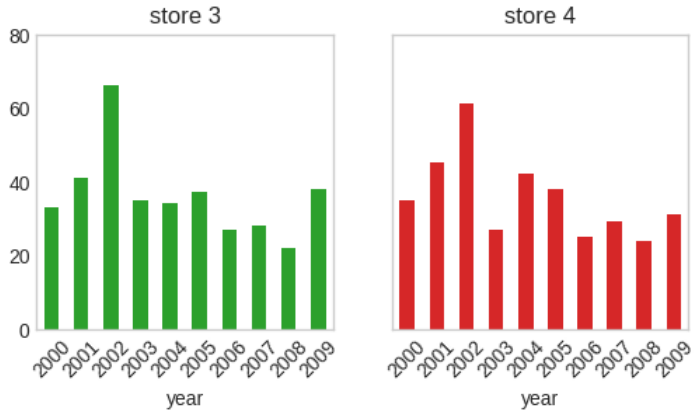
> Объект ax передается в одноименный параметр ax с указанием соответствующего индекса `ax = ax[i, j]`.

Способ 4. Метод .plot() библиотеки Pandas

Метод .plot() библиотеки Pandas также позволяет строить подграфики. Причем в базовом варианте (без дополнительных настроек) мы можем обойтись относительно небольшим количеством кода.

```
# применим метод .plot() ко всем столбцам датафрейма
sales.plot(subplots = True,    # укажем, что хотим создать подграфики
           layout = (2,2),     # пропишем размерность сетки
           kind = 'bar',       # укажем тип графика
           figsize = (6,6),    # зададим размер фигуры
           sharey = True,      # сделаем общую шкалу по оси y
           ylim = (0, 80),     # зададим пределы по оси y
           grid = False,       # уберем сетку
           legend = False,     # уберем легенду
           rot = 45);          # повернем подписи к делениям по оси x на 45 градусов
```

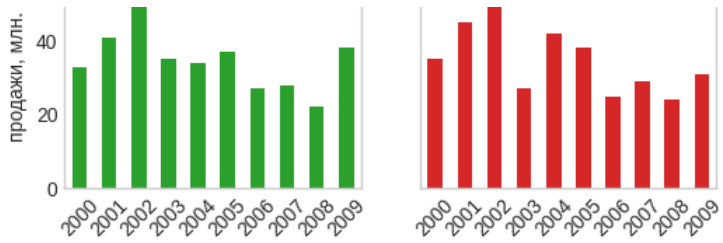




Если же мы захотим создать график, практически идентичный созданному с помощью **функции subplots()**, потребуется уточнить некоторые настройки.

Здесь важно понимать, что метод **.plot()** также основан на библиотеке Matplotlib и также создает объекты класса **plt.axes()**. Для того чтобы ими воспользоваться, передадим результат метода **.plot()** в переменную ax и через индексы модифицируем каждый из подграфиков.

```
# зададим размер строк и столбцов
nrows, ncols = 2, 2

ax = sales.plot(subplots = True,          # укажем, что хотим создать подграфики
                layout = (nrows, ncols), # пропишем размерность сетки
                kind = 'bar',            # укажем тип графика
                figsize = (6,6),         # зададим размер фигуры
                sharey = True,           # сделаем общую шкалу по оси y
                ylim = (0, 80),          # зададим пределы по оси y
                grid = False,            # уберем сетку
                legend = False,          # уберем легенду
                rot = 45);               # повернем подписи к делениям по оси x на 45

# пройдемся по индексам столбцов и строк
for i in range(nrows):
  for j in range(ncols):

    # удалим подписи к оси x
    ax[i, j].set_xlabel('')

    # сделаем подписи по оси y только к первому столбцу
    if j == 0:
      ax[i, j].set_ylabel('продажи, млн. рублей')
    else:
      ax[i, j].set_ylabel('')
```



Также для закрепления понимания выведем индексы каждого из объектов сетки.

```
for i in range(nrows):
  for j in range(ncols):
    print(i,j)
```

```
0 0
0 1
1 0
1 1
```

Дополнительные возможности

Метод .twinx()

Если мы хотим построить две визуализации на одном графике с, например, *разными шкалами по оси y*, то мы можем воспользоваться **методом .twinx()**. По сути, этот метод накладывает один объект ax на другой.

Мы уже использовали этот прием, когда при изучении нормального распределения, рассматривали связь функции плотности (pdf) и функции распределения (cdf).

3D визуализации

Кроме этого, в следующем разделе мы будем использовать 3D визуализации при изучении оптимизации многомерной функции методом градиентного спуска.

Такой график можно построить с помощью **метода .add_subplot()** с параметром *projection = '3d'*.

Подведем итог

На сегодняшнем занятии мы на практике разобрали основные инструменты, применяемые для решения каждой из трех задач EDA (описания данных, выявления различий и нахождения зависимостей), а также посмотрели на некоторые технические детали создания графиков в основных библиотеках Питона.

Замечу, что, как правило, визуализация, вычисление статистических показателей и обработка данных объединяется в один большой этап подготовки данных к моделированию. В рамках же этого курса я сознательно разделил исследовательский анализ и обработку данных на несколько разделов для целей наглядности и системности изложения.

Ответы на вопросы

Вопрос. Зачем использовать Питон, в котором нужно прописывать элементы графиков руками, когда можно использовать инструменты BI, например Tableau?

Ответ. В моем понимании у них разное назначение. Tableau — инструмент для создания интерактивных графиков, которые удобно показывать бизнес-пользователям. Графические инструменты Питона чаще используются внутри блокнота, например, как часть пайплайна по созданию модели ML.

Вопрос. Что такое грамматика графики?

Ответ. *Грамматика графики* (the grammar of graphics) — подход к построению визуализаций, который разбивает график на части подобно тому, как это делает обычная грамматика с естественным языком. По большому счету, это попытка систематизировать графический способ передачи информации и сделать его более осмысленным.

На практике такой подход наилучшим образом реализован в библиотеке ggplot (эта библиотека есть в R и Питоне). Более подробно можно почитать вот в этой статье.

Вопрос. Как посмотреть, какая версия библиотеки используется в Google Colab?

Ответ. Версию библиотеки можно посмотреть с помощью **атрибута __version__**.

```
matplotlib.__version__
```

```
'3.7.1'
```

Более полную информацию о библиотеке можно узнать с помощью **команды show**.

> О программе pip мы уже говорили на прошлом курсе. В Google Colab к ней можно получить доступ через `!pip`.

```
!pip show matplotlib
```

```
Name: matplotlib
Version: 3.7.1
Summary: Python plotting package
Home-page: https://matplotlib.org
Author: John D. Hunter, Michael Droettboom
Author-email: matplotlib-users@python.org
License: PSF
Location: /usr/local/lib/python3.10/dist-packages
Requires: contourpy, cycler, fonttools, kiwisolver, numpy, packaging, pillow, pyparsir
Required-by: arviz, bigframes, datascience, fastai, geemap, imgaug, matplotlib-venn, m
```

Кроме этого, мы можем совместить три команды, **list** (выводит список библиотек), **|** (pipe operator, оператор объединения команд) и **grep <название библиотеки>** (ищет совпадения с передаваемой строкой), для поиска упоминания конкретной библиотеки и ее версии в общем

списке.

```
# посмотрим, упоминается ли слово matplotlib в списке библиотек
# и если да, выведем название библиотеки с этим словом и ее версию
!pip list | grep matplotlib
```

```
matplotlib                      3.7.1
matplotlib-inline               0.1.7
matplotlib-venn                 0.11.10
```