Графическая визуализация данных

С.В. Лемешевский (sergey.lemeshevsky@gmail.com)

Институт математики НАН Беларуси

Apr 1, 2020

Одной из важных частей в анализе данных является графическое визуализация. Это может быть частью исследовательского процесса — например, чтобы помочь идентифицировать выбросы или необходимые преобразования данных, или как способ генерирования идей для моделей. В Python есть много дополнительных библиотек для создания статических или динамических визуализаций, но мы сосредоточемся в основном на `matplotlib` и библиотеках, которые построены на её основе.

Со временем `matplotlib` породила ряд дополнительных наборов инструментов для визуализации данных, которые используют `matplotlib` в качестве «ядра». Одним из таких инструментов является `seaborn`.

Содержание

1. Краткий пример использования `matplotlib`

Для импорта библиотеки `matplotlib` будем использовать следующее соглашение:

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

Ниже приведен пример построения простой прямой:

```
In [2]: import numpy as np

In [3]: data = np.arange(10)

In [4]: data
Out[4]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

In [5]: plt.plot(data)
```

Результат представлен на рисунке 1

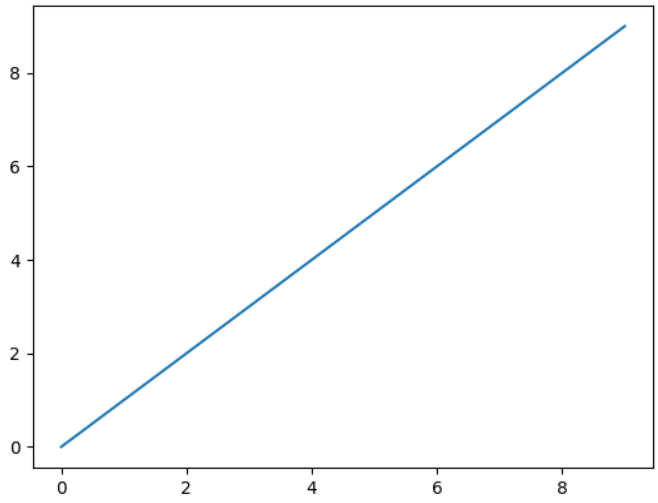


Рис. 1: Построение простейшей прямой линии

1.1. Рисунки и подграфики

Графики в `matplotlib` находятся внутри объекта `Рисунок`. Новый рисунок можно создать с помощью `plt.figure`:

```
In [6]: fig = plt.figure()
```

В интерпретаторе IPython будет построено пустое окно, а в блокноте Jupyter ничего не произойдет. Нельзя создавать окно с пустым рисунком. Нужно создать один или несколько подграфиков (subplots), используя функцию `add_subplot`

```
In [7]: fig.add_subplot(2, 2, 1)
```

Это означает, что рисунок должен быть размером 2×2 (т.е. содержать максимум 4 графика), и мы выбрали первый из четырех графиков (нумерация начинается с единицы). Можно выбрать следующие 2 графика:

```
In [8]: ax2 = fig.add_subplot(2, 2, 2)

In [9]: ax3 = fig.add_subplot(2, 2, 3)
```

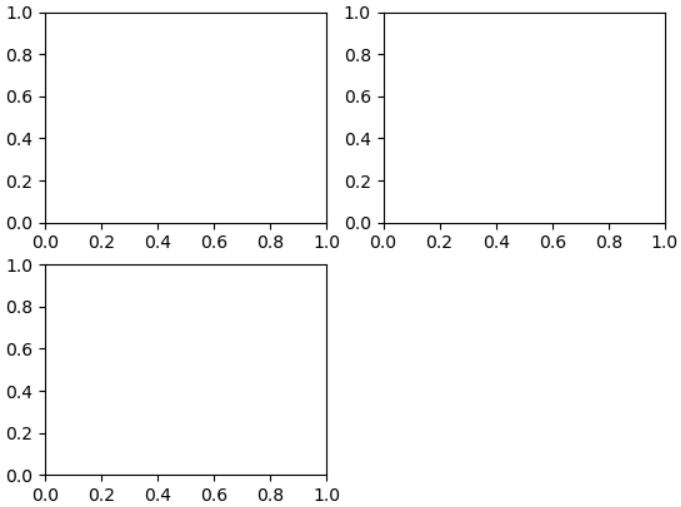


Рис. 2: Пустой рисунок `matplotlib` с тремя графиками

Если выполнить команду построения графика, например, `plt.plot([1.5, 3.5, -2, 1.6])`, вывод будет осуществляться в последний график последнего созданного рисунка. Например, выполнение команды

```
In [10]: plt.plot(np.random.randn(50).cumsum(), 'k--')
```

даст следующий результат:

Выражение `'k–'` задает стиль линии: черная штриховая линия. Метод `fig.add_subplot` возвращает объект `AxesSubplot`, в который можно напрямую выводить график:

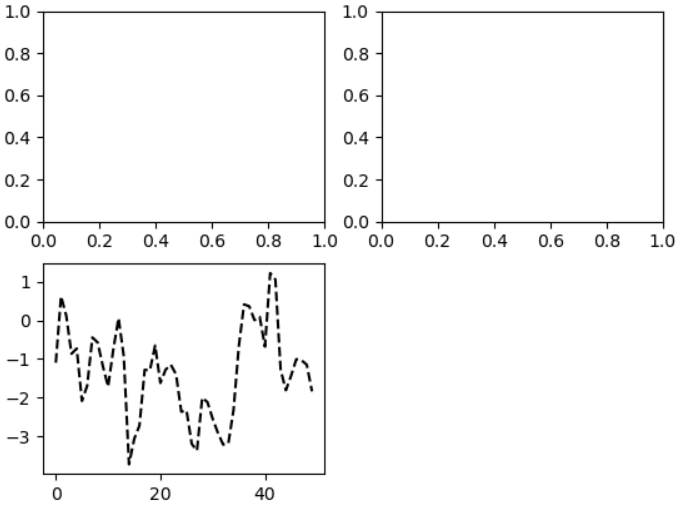


Рис. 3: Визуализация данных простейшей командой

```
In [11]: _ = ax1.hist(np.random.randn(100), bins=20, color='k', alpha=0.3)

In [12]: ax2.scatter(np.arange(30), np.arange(30) + 3 * np.random.randn(30))
```

Полный каталог типов графиков можно найти на сайте https://matplotlib.org/.

1.2. Цвет, маркеры и стили линий

Основная функция `plot` библиотеки `matplotlib` принимает массивы координат `x` и `y` и (опционально) строку, задающую цвет и стиль линии. Например, для того чтобы построить зависимость `y` от `x` зелеными штрихами, необходимо выполнить:

```
ax.plot(x, y, 'g--')
```

Таким образом, мы задали и цвет и стиль линии в виде строки. На практике при программном создании графиков использование строк не удобно. Такой же график можно построить с помощью команды:

```
ax.plot(x, y, linestyle='--', color='g')
```

Графики могут иметь также маркеры для выделения точек данных. Так как `matplotlib` создает непрерывные линии, интерполируя значения между заданными точками, может быть

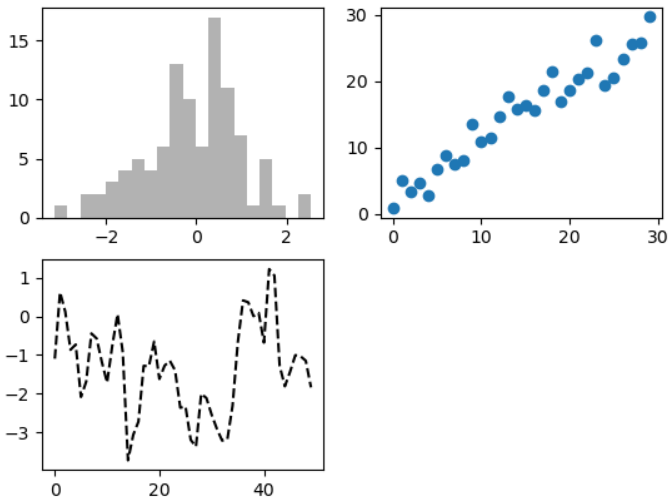


Рис. 4: Визуализация данных

не ясно, где находятся заданные значения. Маркеры могут быть частью строки, задающей стиль линии:

```
In [13]: from numpy.random import randn

In [14]: plt.plot(randn(30).cumsum(), 'ko--')
```

Это же можно было записать более явно:

```
plot(randn(30).cumsum(), color='k', linestyle='dashed', marker='o')
```

Как видно, между последовательными точками строится линейная интерполяция. Это поведение можно изменить с помощью параметра `drawstyle`:

```
In [15]: data = np.random.randn(30).cumsum()

In [16]: plt.plot(data, 'k--', label='Default')
Out[16]: [<matplotlib.lines.Line2D at 0x7fb01ad92f70>]

In [18]: plt.plot(data, 'k-', drawstyle='steps-post', label='steps-post')
Out[18]: [<matplotlib.lines.Line2D at 0x7fb01ad15430>]

In [19]: plt.legend(loc='best')
Out[19]: <matplotlib.legend.Legend at 0x7fb01ad12d00>
```

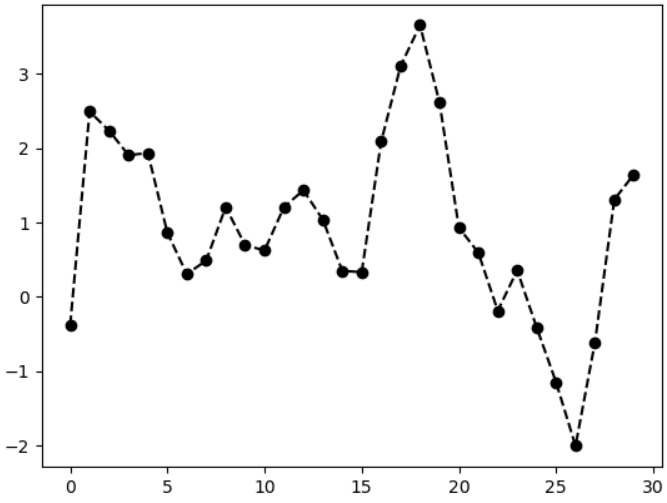


Рис. 5: График с маркерами

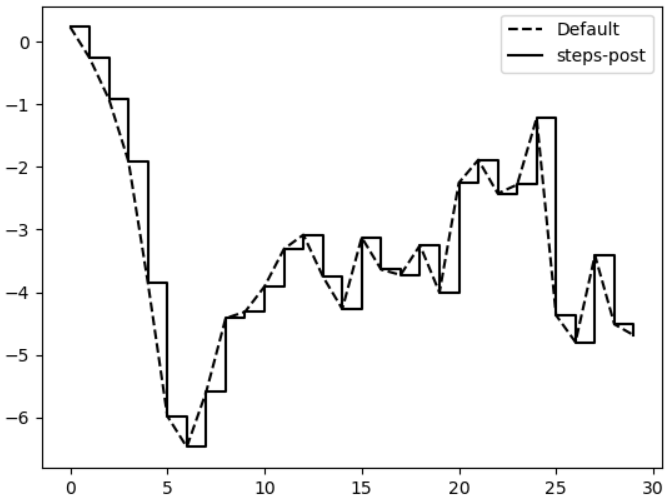
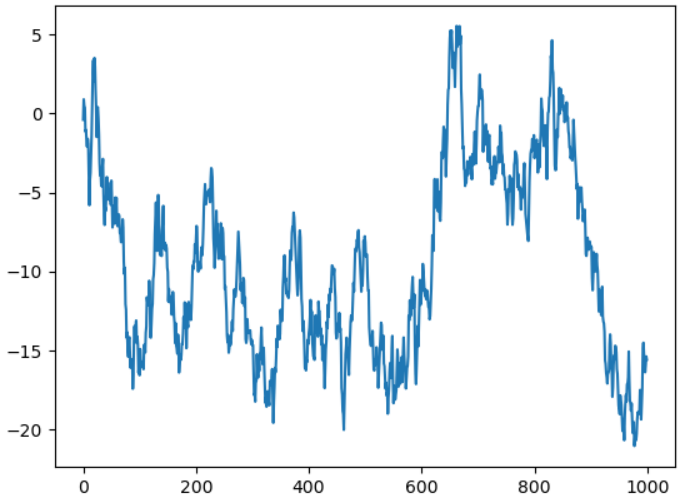


Рис. 6: *Л*инии с разными значениями `drawstyle`

1.3. Подписи к осям, масштаб и легенда

Для иллюстрации настройки графиков создадим простой рисунок и отобразим график случайного блуждания:

```
In [20]: fig = plt.figure()

In [21]: ax = fig.add_subplot(1, 1, 1)

In [22]: ax.plot(np.random.randn(1000).cumsum())
Out[22]: [<matplotlib.lines.Line2D at 0x7fb01ad29c10>]
```



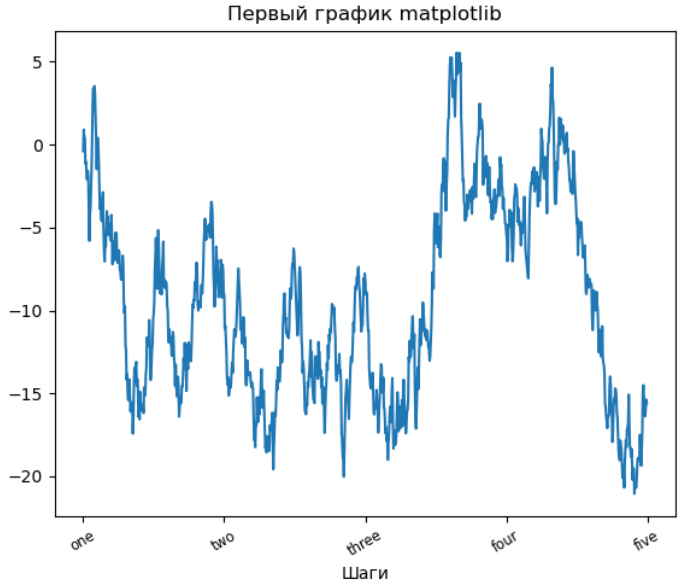
Для изменения подписей на оси x воспользуемся методами `set_xticks` и `set_xticklables`:

```
In [23]: ticks = ax.set_xticks([0, 250, 500, 750, 1000])

In [24]: labels = ax.set_xticklabels(['one', 'two', 'three', 'four', 'five'],
    ...:rotation=30, fontsize='small')
```

Параметр `rotation` поворачивает метки надписей на оси x на 30 градусов. И, наконец, зададим название графика и метку для оси x с помощью методов `set_title` и `set_xlabel`:

```
In [25]: ax.set_title('Первый график matplotlib')
Out[25]: Text(0.5, 1.0, 'Первый график matplotlib')
```

```
In [26]: ax.set_xlabel('Шаги')
Out[26]: Text(0.5, 10.888891973024519, 'Шаги')
```



Модификация оси y осуществляется точно также, только нужно заменить `x` на `y` в приведенном выше коде. У класса осей есть метод `set`, который допускает пакетную настройку свойств графика. В предыдущем примере можно было также написать:

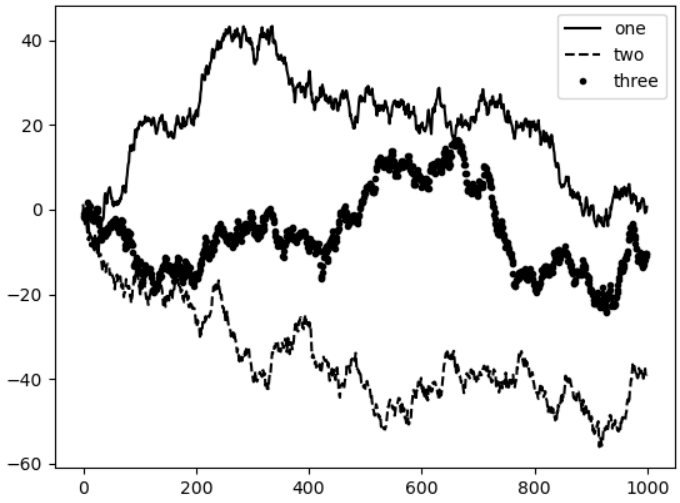
```
props = {
    'title': 'Первый график matplotlib',
    'xlabel': 'Шаги
}
ax.set(**props)
```

Для вывода легенды графика есть несколько способов. Простейший заключается в передаче аргумента `label` при построении графиков:

```
In [27]: fig = plt.figure(); ax = fig.add_subplot(1, 1, 1)

In [28]: ax.plot(randn(1000).cumsum(), 'k', label='one')
Out[28]: [<matplotlib.lines.Line2D at 0x7fb01acbd490>]

In [29]: ax.plot(randn(1000).cumsum(), 'k--', label='two')
Out[29]: [<matplotlib.lines.Line2D at 0x7fb01a7664c0>]

In [30]: ax.plot(randn(1000).cumsum(), 'k.', label='three')
Out[30]: [<matplotlib.lines.Line2D at 0x7fb01a7660a0>]
```

После этого можно вызвать `ax.leend()` или `plt.legend()` для автоматического создания легенды:

```
In [31]: ax.legend(loc='best')
```



1.4. Сохранение рисунков в файл

Можно сохранить активный рисунок в файл с помощью метода `plt.savefig`. Например, чтобы сохранить рисунок в формате SVG достаточно набрать:

```
plt.savefig('figpath.svg')
```

Тип файла определяется расширением. Есть пара важных параметров: `dpi`, который задает разрешение рисунка (точек на дюйм), `bbox_inches`, который может обрезать пустое пространство вокруг рисунка. Например, чтобы сохранить тот же график в формате PNG с разрешением 400 DPI, нудно выполнить:

```
plt.savefig('figpath.png', dpi=400, bbox_inches='tight')
```

Функция `savefig` сохраняет не только на диск. Она может записывать график в любой файлоподобный объект, например в `BytesIO`:

```
from io import BytesIO
buffer = BytesIO()
plt.savefig(buffer)
plot_data = buffer.getvalue()
```

Таблица 1: Метод `savefig`: параметры

Параметр	Описание
`fname`	Строка, содержащая путь к файлу или файлоподобный объект Python. Формат рисунка определяется расширением файла
`dpi`	Разрешение рисунка в точках на дюйм. По умолчанию 100
`facecolor`, `edgecolor`	Цвет фона рисунка вне графика. По умолчанию `w` (белый)
`format`	Явное задание формата файла
`bbox_inches`	Часть рисунка для сохранения. Если задано `'tight'`, 2будет попытка обрезать пустое пространство вокруг

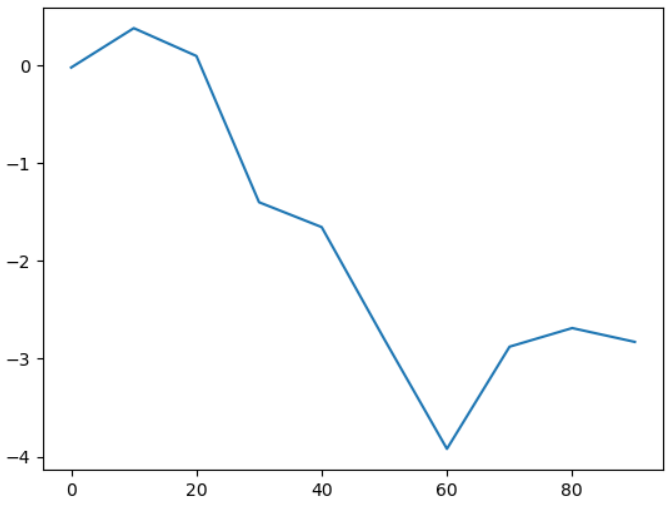
2. Построение графиков с помощью `pandas` и `seaborn`

Библиотека `matplotlib` может быть инструментом довольно низкого уровня. График собирается из его базовых компонентов: отображения данных (т.е. тип графика: линия, полоса, прямоугольник, разброс, контур и т.д.), легенды, заголовка, меток и других аннотаций. В библиотеке `pandas` мы можем получить множество столбцов данных, а также метки строк и столбцов. В `pandas` имеются встроенные методы, которые упрощают визуализацию объектов `DataFrame` и `Series`. Еще одна библиотека для статистических графиков — `seaborn`.

2.1. Линейные графики

Объекты `Series` и `DataFrame` имеют метод `plot` для создания базовых типов графиков. По умолчанию `plot()` создает линейные графики

```
In [32]: s = pd.Series(np.random.randn(10).cumsum(), index=np.arange(0, 100, 10))

In [33]: s.plot()
Out[33]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb01a5a7eb0>
```



Индекс объекта `Series` передается в `plot` библиотеки `matplotlib` для оси x. При этом такое поведение можно отключить с помощью параметра `use_index = False`. В таблице 2 дается полный список параметров функции `Series.plot`.

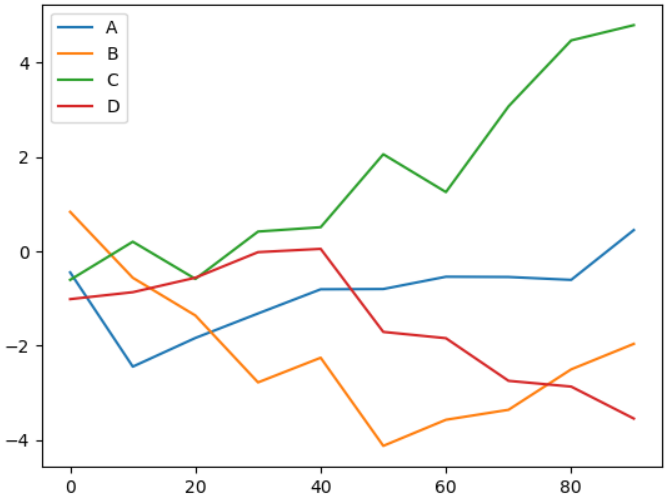
Большинство графических методов `pandas` принимают опциональный параметр `ax`, который может являться объектом `subplot`. Это позволяет размещать подграфики на сетке.

Таблица 2: Параметры метода `Series.plot`

Параметр	**Описани**
`label`	Метка для легенды
`ax`	Объект `subplot` из `matplotlib`, в который выводится график. Если не задан — вывод идет в активный подграфик
`style`	Строка, задающая стиль графика (например, `ko–`)
`alpha`	Прозрачность заполнения графика (от 0 до 1)
`kind`	Тип графика. Может быть: 'area' , 'bar' , 'barh' , 'density', 'hist' , 'kde' , 'line' , 'pie'
`logy`	Использовать ли логарифмический масштаб по оси `y`
`use_index`	Использовать ли объект индекс для меток оси
`rot`	Поворот меток оси
`xticks`	Значения для меток оси `x`
`yticks`	Значения для меток оси `x`
`xlim`	Границы по оси `x` (например, `[0, 10]`)
`ylim`	Границы по оси `y`
`grid`	Отображать ли сетку по осям (включено по умолчанию)

Метод `plot` объекта `DataFrame` выводит график для каждого столбца данных в виде линии на одном и том же подграфике, создавая при этом легенду автоматически:

```
In [34]: df = pd.DataFrame(np.random.randn(10, 4).cumsum(0),
   ....: columns=['A', 'B', 'C', 'D'],
   ....: index=np.arange(0, 100, 10))

In [35]: df.plot()
Out[35]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb01acbd220>
```



Атрибут `plot` содержит «семейство» методов для различных типов графиков. Например, `df.plot()` эквивалентно `df.plot.line()`.

В `DataFrame` есть несколько параметры, которые обеспечивают некоторую гибкость при обработке столбцов. Например, следует ли разместить их все на одном подграфике или создавать отдельные. В таблице 3 представлены такие параметры.

Таблица 3: Специфичные для `DataFrame` параметры `plot`

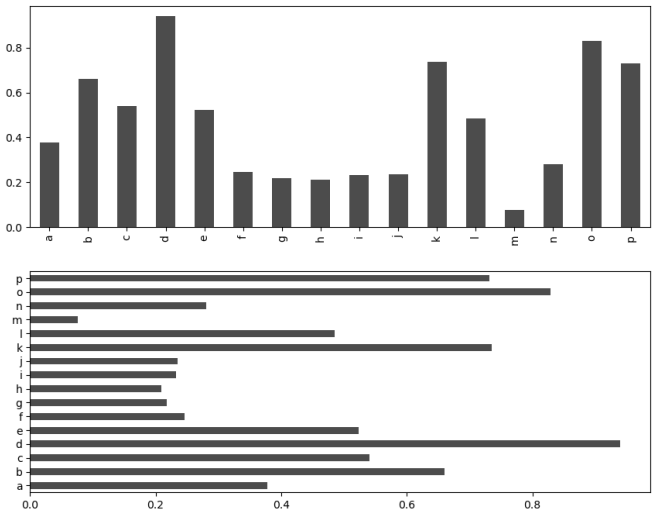
Параметр	Описание
`subplots`	Рисовать ли каждый столбец `DataFrame` в отдельном подграфике
`sharex`	Если `subplots=True`, использовать ли одну и ту же ось `x`, связывая метки оси

`sharey`	Если `subplots=True`, использовать ли одну и ту же ось `y`
`figsize`	Размер рисунка для создания в виде кортежа
`title`	Заголовок рисунка в виде строки
`legend`	Добавлять ли легенду на рисунок (по умолчанию `True`)
`sort_columns`	Отображать ли столбцы в алфавитном порядке

2.2. Столбчатые диаграммы

Методы `plot.bar()` и `plot.barh()` строят вертикальные и горизонтальные столбчатые диаграммы. В этом случае индексы объектов `Series` и `DataFrame` в качестве меток на оси `x` (`bar`) или `y` (`barh`).

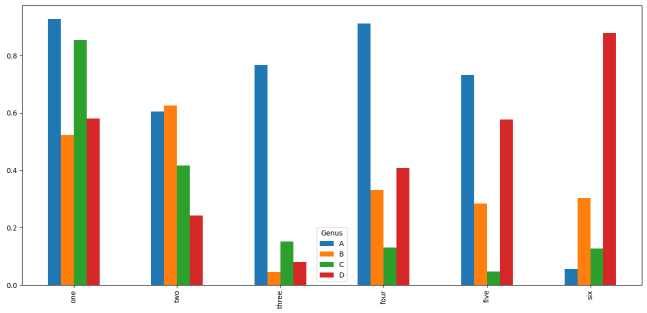
```
In [36]: fig, axes = plt.subplots(2, 1)

In [37]: data = pd.Series(np.random.rand(16), index=list('abcdefghijklmnop'))

In [38]: data.plot.bar(ax=axes[0], color='k', alpha=0.7)
Out[38]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb01add0880>

In [39]: data.plot.barh(ax=axes[1], color='k', alpha=0.7)
Out[39]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb01ac37af0>
```



Параметры `color='k'` и `alpha=0.7` устанавливают цвет графика в черный и частичную прозрачность для заполнения.

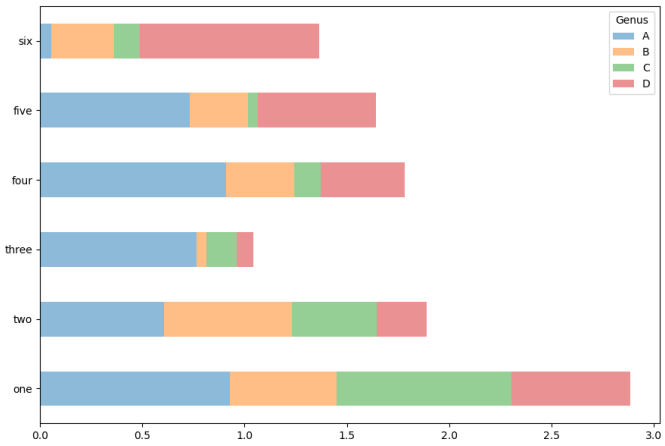
В `DataFrame` столбчатые диаграммы группируют каждую строку значений вместе в группу столбиков, соответствующих каждому значению в строке:

```
In [40]: df = pd.DataFrame(np.random.rand(6, 4),
   ....: index=['one', 'two', 'three', 'four', 'five', 'six'],
   ....: columns=pd.Index(['A', 'B', 'C', 'D'], name='Genus'))

In [41]: df
Out[41]:
Genus         A         B         C         D
one    0.927796  0.523404  0.854142  0.581220
two    0.605077  0.625539  0.416703  0.243262
three  0.766480  0.045345  0.151112  0.079766
four   0.911277  0.331451  0.130595  0.408353
five   0.732850  0.283842  0.046650  0.577424
six    0.056685  0.303711  0.126910  0.879065

In [42]: df.plot.bar()
Out[42]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb01ae47e20>
```



Обратите внимание на то, что имя столбцов `'Genus'` используется в качестве заголовка легенды. Для создания столбчатых диаграмм с накоплением для `DataFrame` задается параметр `stacked=True`, в результате чего значение в каждой строке будут сгруппировано вместе

```
In [42]: df.plot.barh(stacked=True, alpha=0.5)
Out[42]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb018fef2e0>
```

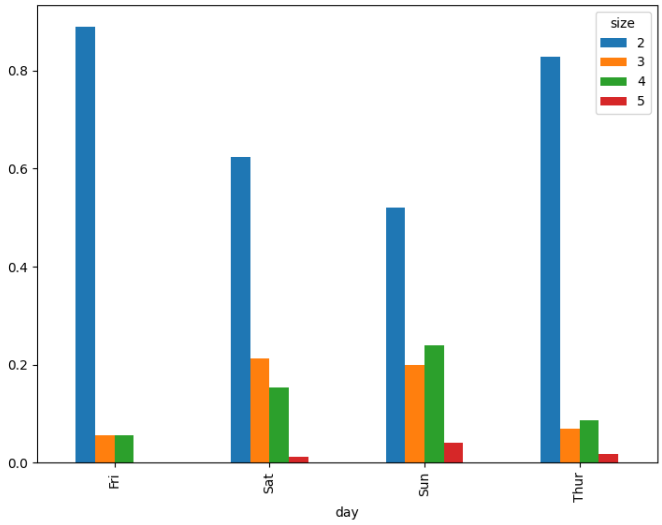


Предположим, что есть набор данных по счетам и чаевым в ресторане, и нам нужно построить столбчатую диаграмму с накоплением, показывающую процентное соотношение точек данных для каждого размера группы в каждый день. Загрузим данные из файла tips.csv и создадим сводную по дням и размеру вечеринки (количество человек):

```
In [43]: tips = pd.read_csv('tips.csv')

In [44]: tips.head()
Out[44]:
   total_bill   tip smoker  day    time  size
0       16.99  1.01     No  Sun  Dinner     2
1       10.34  1.66     No  Sun  Dinner     3
2       21.01  3.50     No  Sun  Dinner     3
3       23.68  3.31     No  Sun  Dinner     2
4       24.59  3.61     No  Sun  Dinner     4

In [45]: party_counts = pd.crosstab(tips['day'], tips['size'])

In [46]: party_counts
Out[46]:
size  1   2   3   4  5  6
day
Fri   1  16   1   1  0  0
Sat   2  53  18  13  1  0
Sun   0  39  15  18  3  1
Thur  1  48   4   5  1  3

In [47]: party_counts = party_counts.loc[:, 2:5]
```

Теперь нормализуем данные так, чтобы сумма в каждой строке была равна 1 и построим столбчатую диаграмму:

```
In [48]: party_pcts = party_counts.div(party_counts.sum(1), axis=0)

In [49]: party_pcts
Out[49]:
size         2         3         4         5
day
Fri   0.888889  0.055556  0.055556  0.000000
Sat   0.623529  0.211765  0.152941  0.011765
Sun   0.520000  0.200000  0.240000  0.040000
Thur  0.827586  0.068966  0.086207  0.017241

In [50]: party_pcts.plot.bar()
Out[50]: <matplotlib.axes._subplots.AxesSubplot at 0x7f41cb024a60>
```



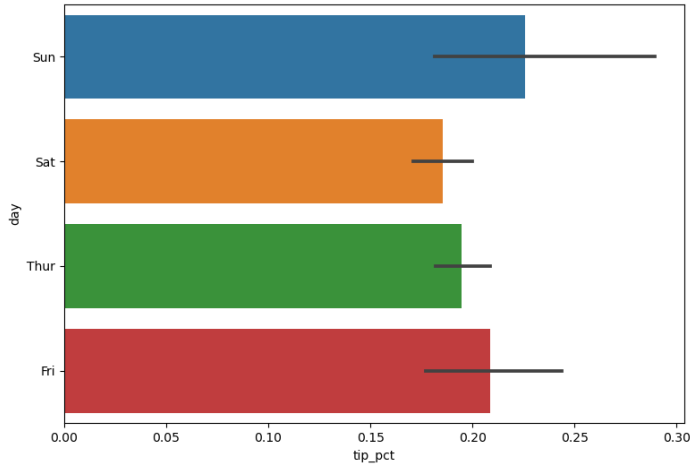
Таким образом, видно, что количество участников вечеринок в данном наборе увеличивается в выходные дни.

В случае, если требуется агрегировать или суммировать данные перед построением графика, использование пакета `seaborn` может значительно упростить задачу. Давайте посмотрим на процент чаевых в день с помощью библиотеки `seaborn`:

```
In [51]: import seaborn as sns

In [52]: tips['tip_pct'] = tips['tip']/(tips['total_bill'] - tips['tip'])

In [53]: tips.head()
Out[53]:
   total_bill   tip smoker  day    time  size   tip_pct
```

```
0       16.99  1.01     No  Sun  Dinner     2  0.063204
1       10.34  1.66     No  Sun  Dinner     3  0.191244
2       21.01  3.50     No  Sun  Dinner     3  0.199886
3       23.68  3.31     No  Sun  Dinner     2  0.162494
4       24.59  3.61     No  Sun  Dinner     4  0.172069
```

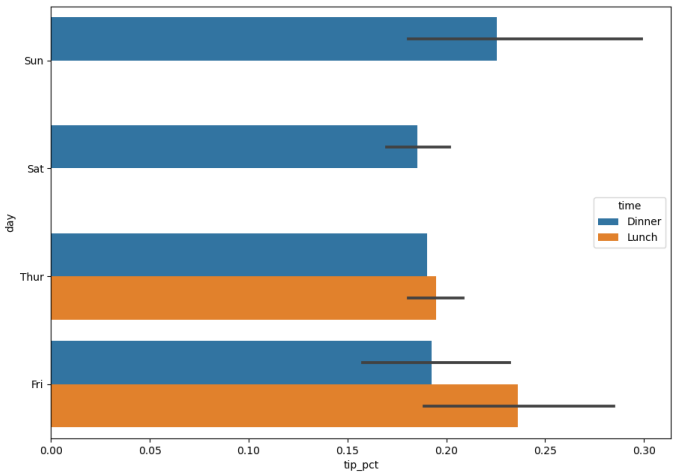
```
In [54]: sns.barplot(x='tip_pct', y='day', data=tips, orient='h')
Out[54]: <matplotlib.axes._subplots.AxesSubplot at 0x7f41cb608dc0>
```



Функция `barplot` библиотеки `seaborn` принимает параметр `data`, который может быть объектом `DataFrame`. Остальные параметры ссылаются на имена столбцов. Поскольку в день имеется несколько наблюдений, то столбцы диаграммы представляют собой среднее значение параметра `tip_pct`. Черные линии, нарисованные на столбцах диаграммы, представляют 95-процентный доверительный интервал (это можно настроить с помощью опционального параметра).

Функция `barplot` имеет параметр `hue`, который позволяет разделить отображение по дополнительному категориальному значению:

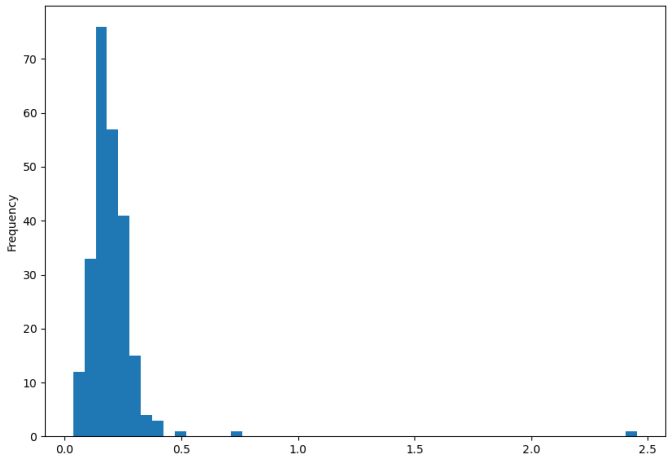
```
In [55]: sns.barplot(x='tip_pct', y='day', hue='time', data=tips, orient='h')
Out[55]: <matplotlib.axes._subplots.AxesSubplot at 0x7f41ad8258b0>
```



2.3. Гистограммы и графики плотности распределения

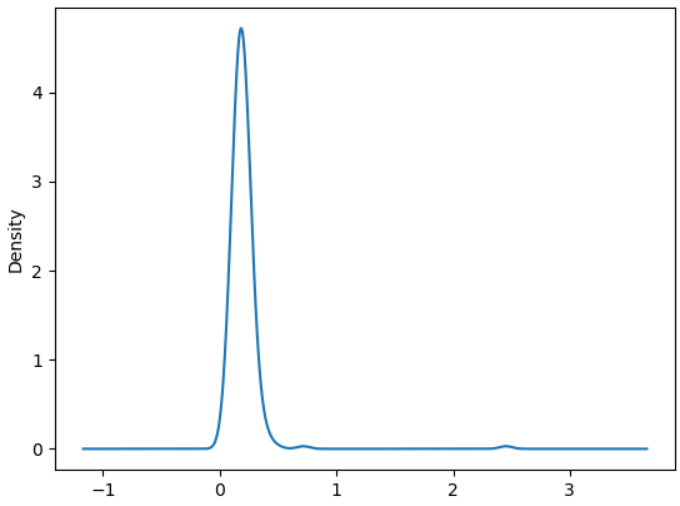
Гистограмма — это своего рода столбчатая диаграмма, которая дает дискретное отображение частоты значений. Составим гистограмму процентных долей от общего счета, используя метод `plot.hist` объекта `Series`:

```
In [56]: tips['tip_pct'].plot.hist(bins=50)
Out[56]: <matplotlib.axes._subplots.AxesSubplot at 0x7f41ad456460>
```



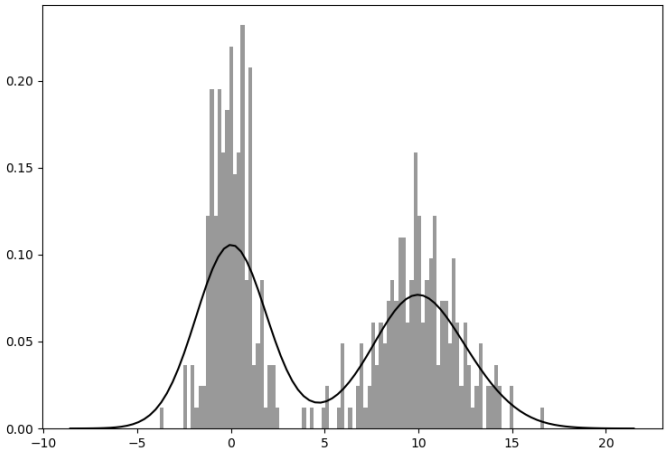
Связанный с гистограммой тип графиков — *график плотности*, который формируется путем вычисления оценки непрерывного распределения вероятности, которое могло бы генерироваться наблюдаемыми данными. Обычная процедура заключается в аппроксимации этого распределение как смеси «ядер», то есть более простых распределений, таких как нормальное распределение. Таким образом, графики под графиками плотности также можно понимать графики оценки плотности ядра (*K*ernel *D*ensity *E*stimate). Функции `plot.kde` и `plot.density` строят график плотности, используя подход KDE:

```
In [57]: tips['tip_pct'].plot.kde()
Out[57]: <matplotlib.axes._subplots.AxesSubplot at 0x7f41ada33670>
```



Библиотека `seaborn` упрощает создание гистограмм и графиков плотности с помощью метода `distplot`, который позволяет одновременно строить как гистограмму, так и непрерывную оценку плотности. В качестве примера рассмотрим бимодальное распределение, состоящее из двух разных стандартных нормальных распределений:

```
In [58]: comp1 = np.random.normal(0, 1, size=200)

In [59]: comp2 = np.random.normal(10, 2, size=200)

In [60]: values = pd.Series(np.concatenate([comp1, comp2]))

In [61]: sns.distplot(values, bins=100, color='k')
Out[61]: <matplotlib.axes._subplots.AxesSubplot at 0x7f41cbf66c40>
```



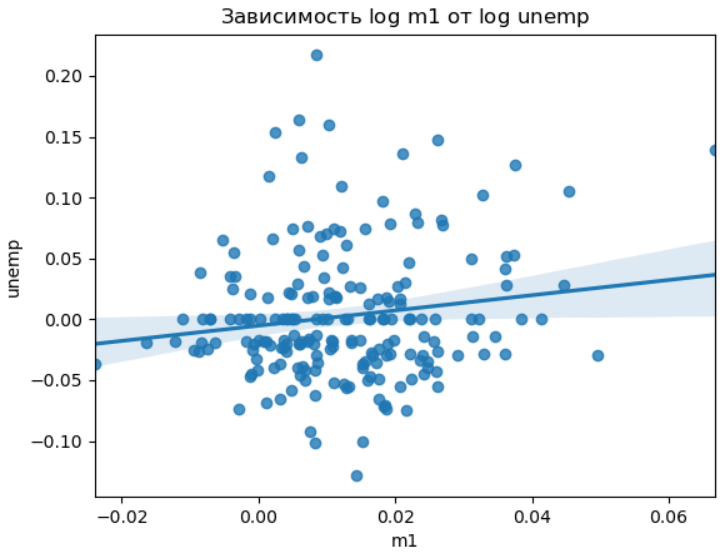
2.4. *Диаграммы рассеяния или точечные графики*

Диаграммы рассеяния полезны при изучении связей между двумя одномерными рядами данных. Например, загрузим набор данных из файла macrodata.csv проекта Statmodels. Выберем некоторые переменные и вычислим «логарифмические разности»:

```
In [62]: macro = pd.read_csv('macrodata.csv')

In [63]: data = macro[['cpi', 'm1', 'tbilrate', 'unemp']]

In [64]: trans_data = np.log(data).diff().dropna()

In [65]: trans_data[-5:]
Out[65]:
          cpi        m1  tbilrate     unemp
198 -0.007904  0.045361 -0.396881  0.105361
199 -0.021979  0.066753 -2.277267  0.139762
200  0.002340  0.010286  0.606136  0.160343
201  0.008419  0.037461 -0.200671  0.127339
202  0.008894  0.012202 -0.405465  0.042560
```

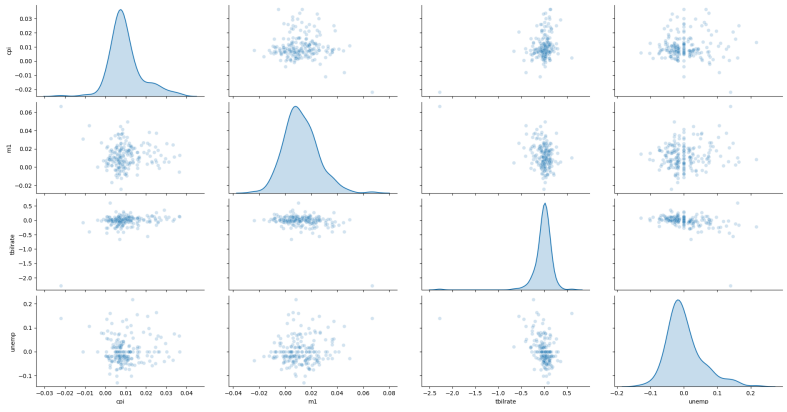
Теперь воспользуемся функцией `regplot` библиотеки `seaborn`, которая строит графики рассеяния и предлагает график линейной регрессии:

```
In [66]: sns.regplot('m1', 'unemp', data=trans_data)
Out[66]: <matplotlib.axes._subplots.AxesSubplot at 0x7f41ad901100>

In [67]: plt.title('Зависимость $\log$ {} от $\log$ {}'.format('m1', 'unemp'))
Out[67]: Text(0.5, 1.0, 'Зависимость $\\log$ m1 от $\\log$ unemp')
```



При анализе данных полезно иметь возможность просматривать все диаграммы рассеяния среди группы переменных, т.е. строить, так называемые, *парные графики* или *матрицу диаграмм рассеяния*. В библиотеке `seaborn` для этого есть удобная функция `pairplot`, которая, в частности, поддерживает размещение гистограмм или оценок плотности каждой переменной по диагонали:

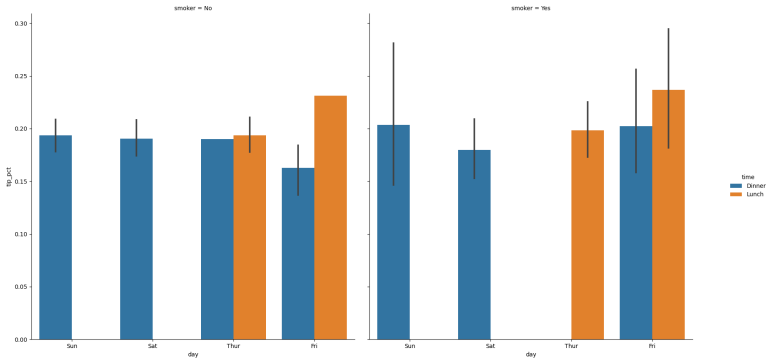
```
In [68]: sns.pairplot(trans_data, diag_kind='kde', plot_kws={'alpha': 0.2})
Out[68]: <seaborn.axisgrid.PairGrid at 0x7f419d6126d0>
```



2.5. Категориальные данные

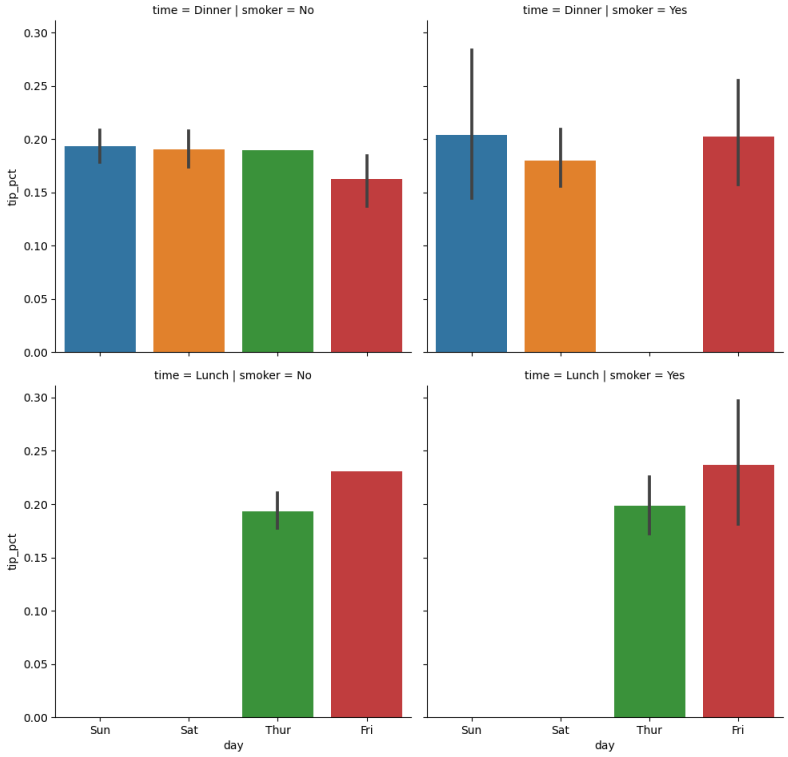
Одним из способов визуализации данных с множеством категориальных переменных является использование сетки фасетов (*facet grid*). В библиотеке `seaborn` есть удобная функция `catplot`, которая упрощает создание сетки фасетов:

```
In [69]: sns.catplot(x='day', y='tip_pct', hue='time', col='smoker',
    ...: kind='bar', data=tips[tips.tip_pct < 1])
Out[69]: <seaborn.axisgrid.FacetGrid at 0x7f419d6d96a0>
```



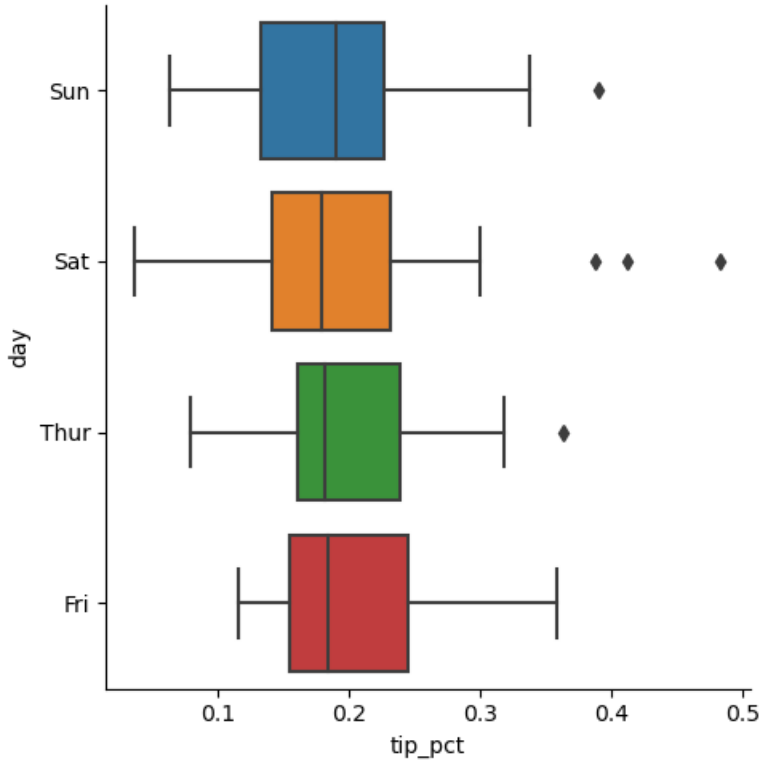
Вместо отображения разными цветами столбцов диаграмм в фасете мы также можем расширить сетку фасетов, добавив одну строку по времени:

```
In [70]: sns.catplot(x='day', y='tip_pct', row='time',
    ...: col='smoker',
    ...: kind='bar', data=tips[tips.tip_pct < 1])
Out[70]: <seaborn.axisgrid.FacetGrid at 0x7f419deeb1f0>
```



Функция `catplot` поддерживает другие типы графиков, которые могут быть полезны. Например, блочные графики, которые показывают медиану, квартили и выбросы:

```
In [71]: sns.catplot(x='tip_pct', y='day', kind='box', data=tips[tips.tip_pct < 0.5])
Out[71]: <seaborn.axisgrid.FacetGrid at 0x7f419df999d0>
```



Можно создавать свои собственные сетки фасетов, используя более общий класс `seaborn.FacetGrid` (см. документацию seaborn).