

Инструменты обработки и визуализации данных

Тема №7 – Визуализация в seaborn

Seaborn – это библиотека визуализации данных на Python, основанная на matplotlib. Она предоставляет высокоуровневый интерфейс для построения привлекательных и информативных статистических графиков.

```
In [1]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.__version__
```

```
Out[1]: '0.12.2'
```

1. Построение графиков числовых данных

- relplot()
- scatterplot()
- lineplot()

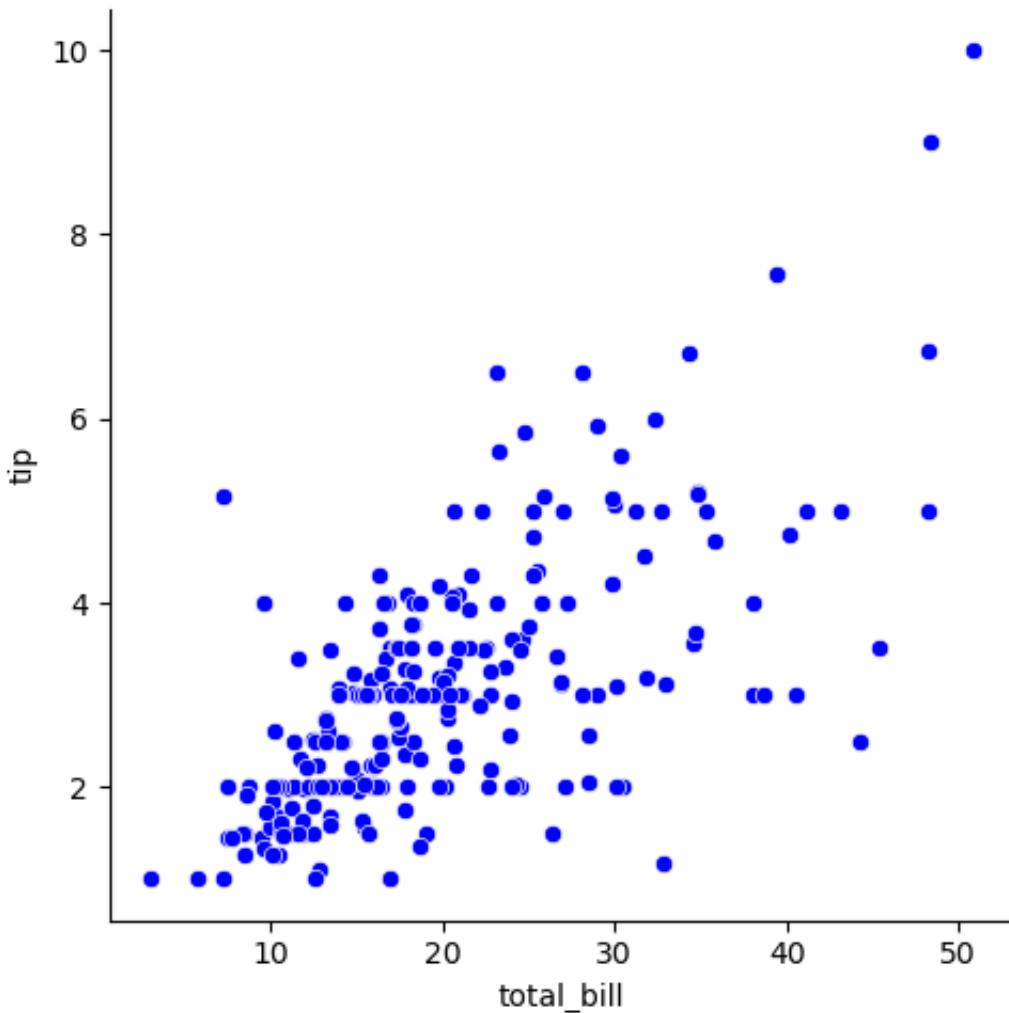
```
In [2]: tip= sns.load_dataset('tips')
tip.head()
```

```
Out[2]:
```

	total_bill	tip	sex	smoker	day	time	size	
0	16.99	1.01	Female		No	Sun	Dinner	2
1	10.34	1.66	Male		No	Sun	Dinner	3
2	21.01	3.50	Male		No	Sun	Dinner	3
3	23.68	3.31	Male		No	Sun	Dinner	2
4	24.59	3.61	Female		No	Sun	Dinner	4

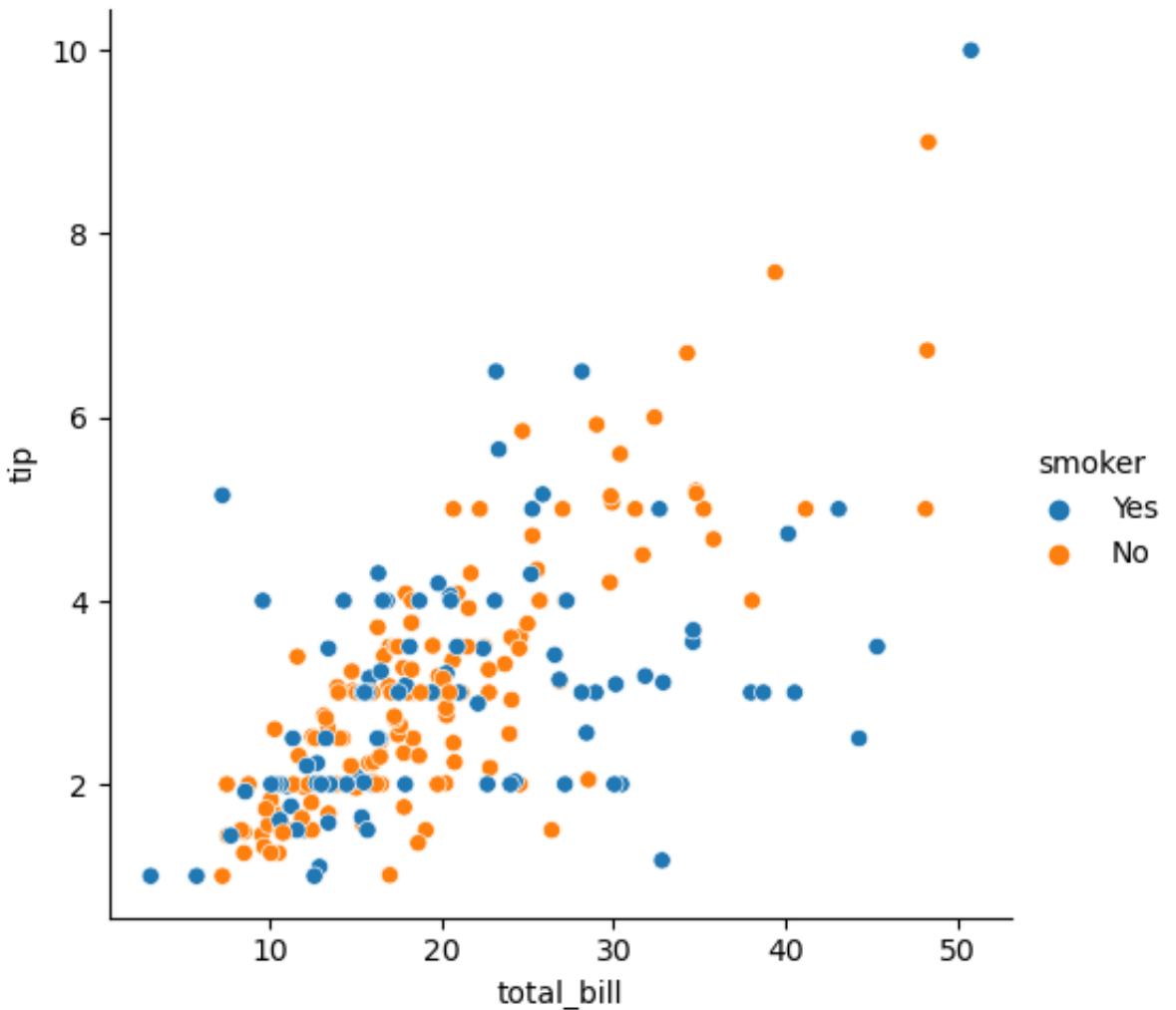
Диаграмма рассеяния

```
In [3]: sns.relplot(x= 'total_bill', y= 'tip', data= tip, color= 'b');
```



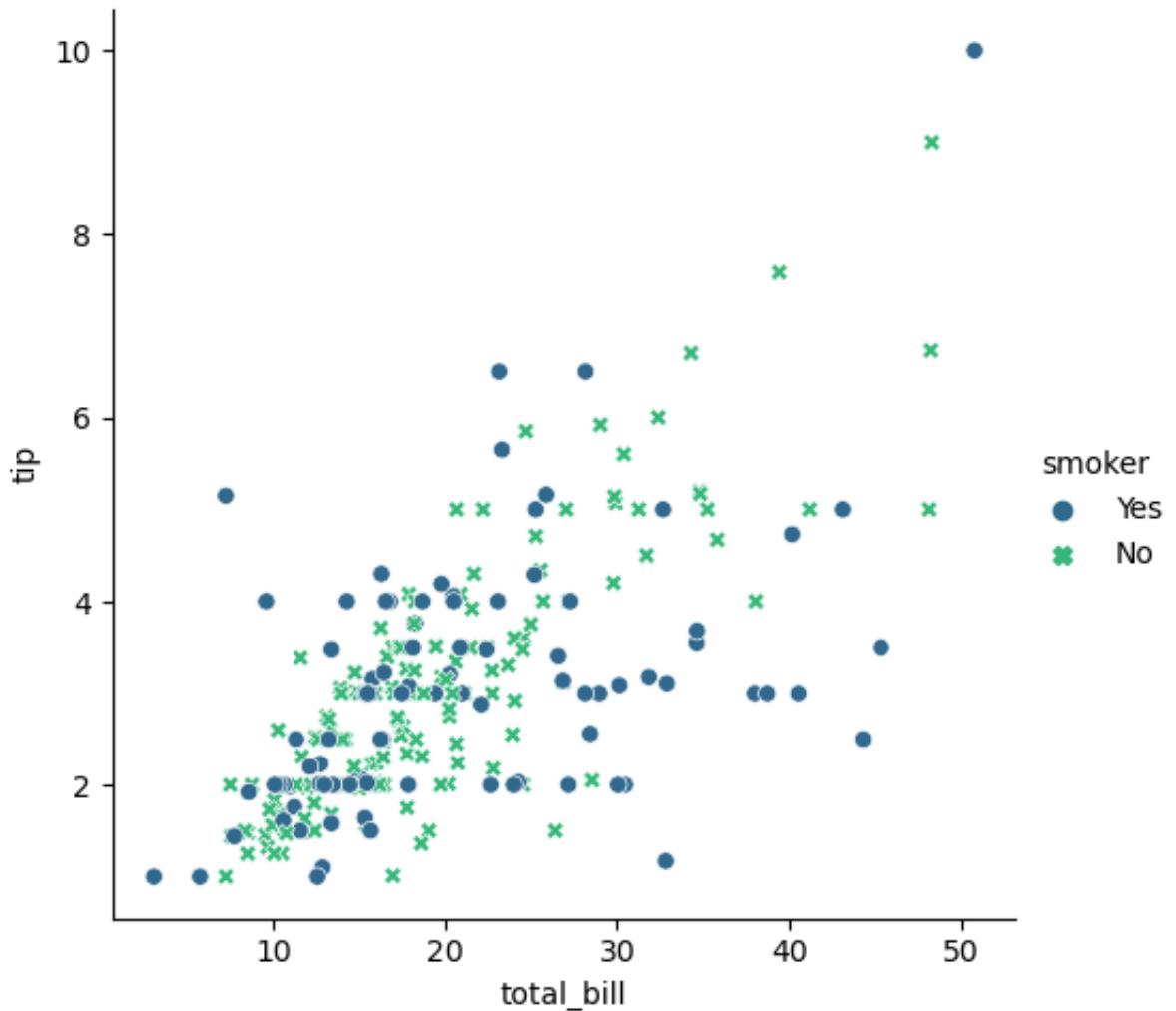
Хотя точки отображаются в двух измерениях, к графику можно добавить ещё одно измерение, раскрасив точки в соответствии с третьей переменной. В языке Seaborn это называется «использованием семантики оттенка» (“hue semantic”), поскольку цвет точки приобретает значение:

```
In [4]: sns.relplot(x= 'total_bill', y= 'tip', data= tip, hue= 'smoker');
```



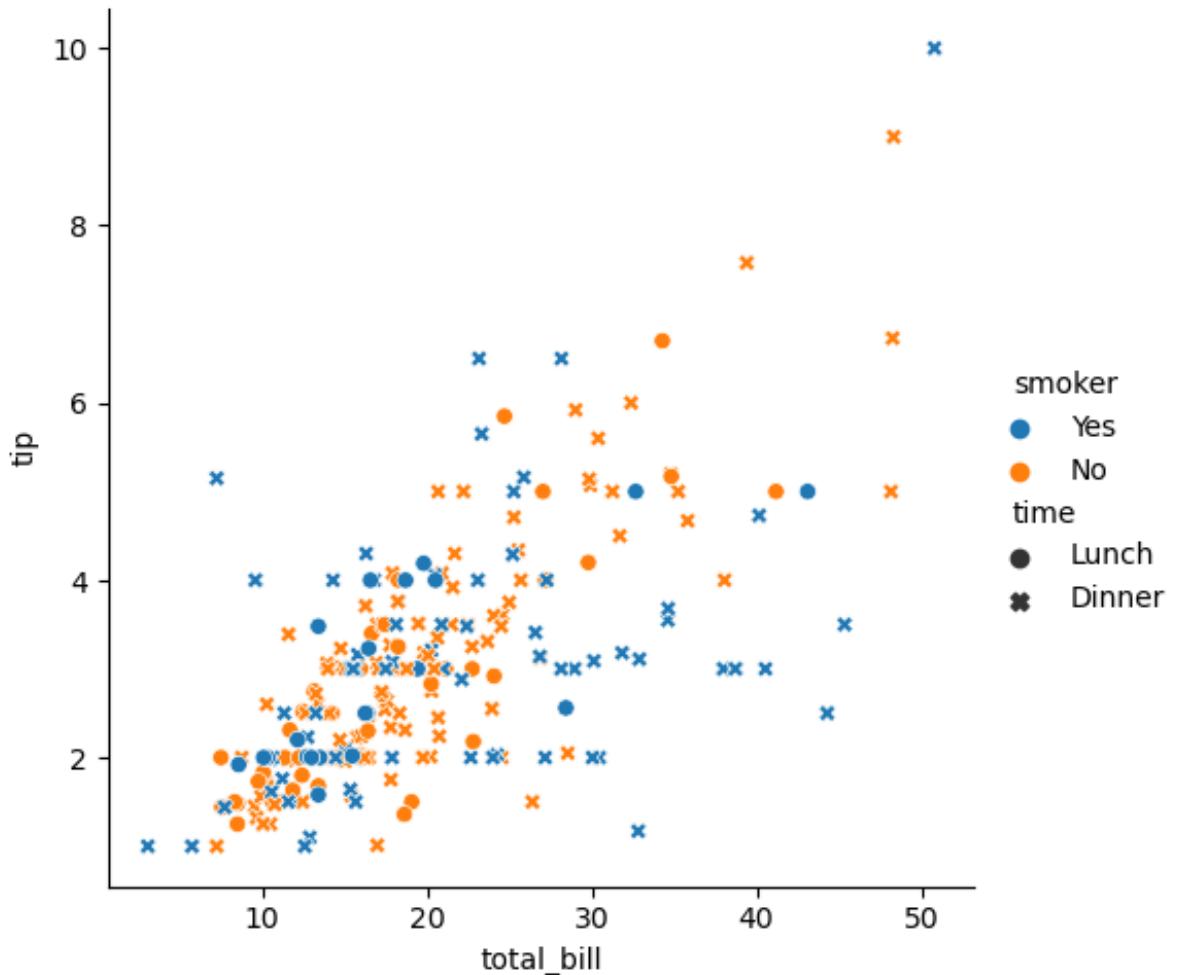
Чтобы подчеркнуть разницу между классами и улучшить доступность, можно использовать разные стили маркеров для каждого класса:

```
In [5]: sns.relplot(x= 'total_bill', y= 'tip', data= tip, hue= 'smoker', st
```



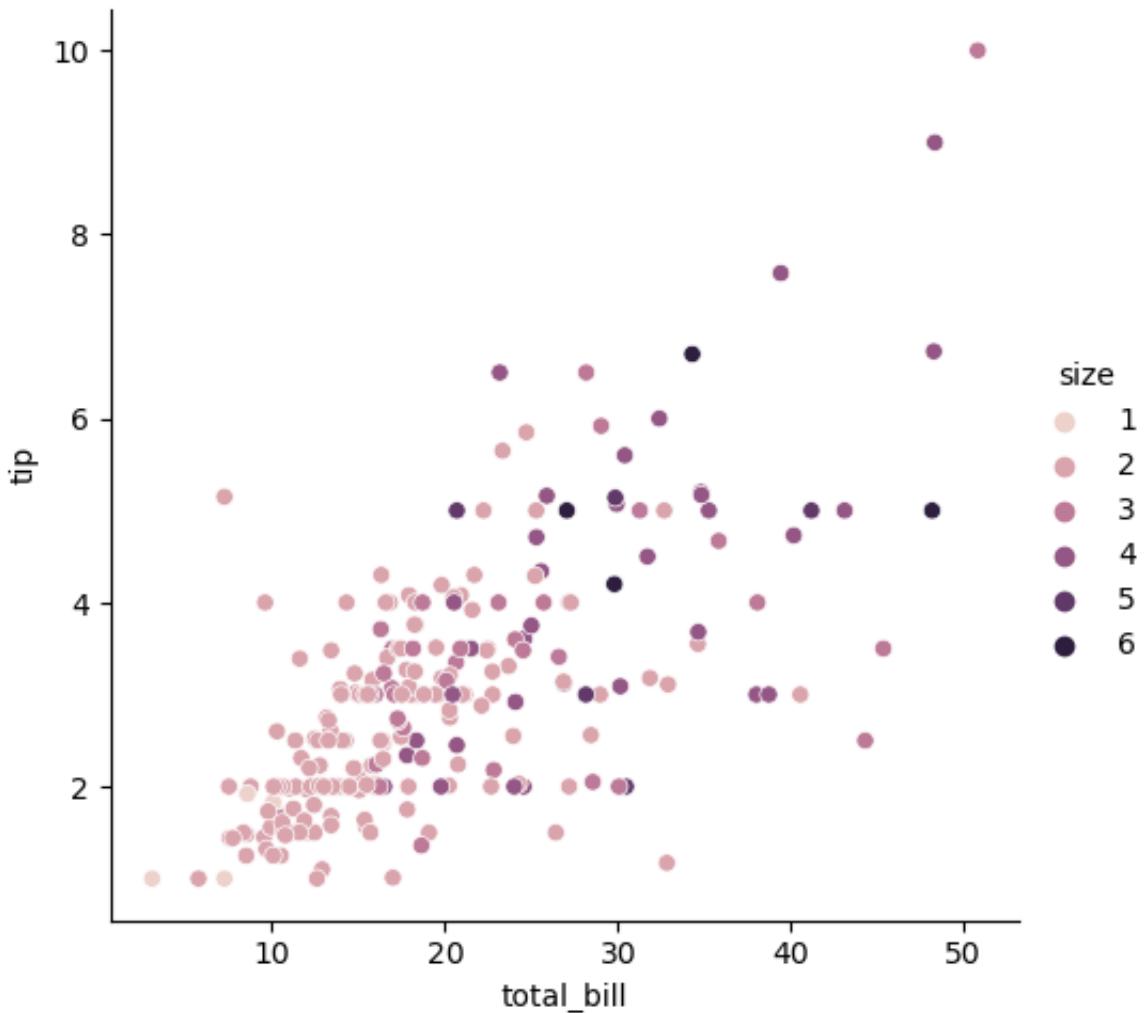
Также можно представить четыре переменные, изменяя оттенок и стиль каждой точки независимо.

```
In [6]: sns.relplot(x= 'total_bill', y= 'tip', data= tip, hue= 'smoker', st
```



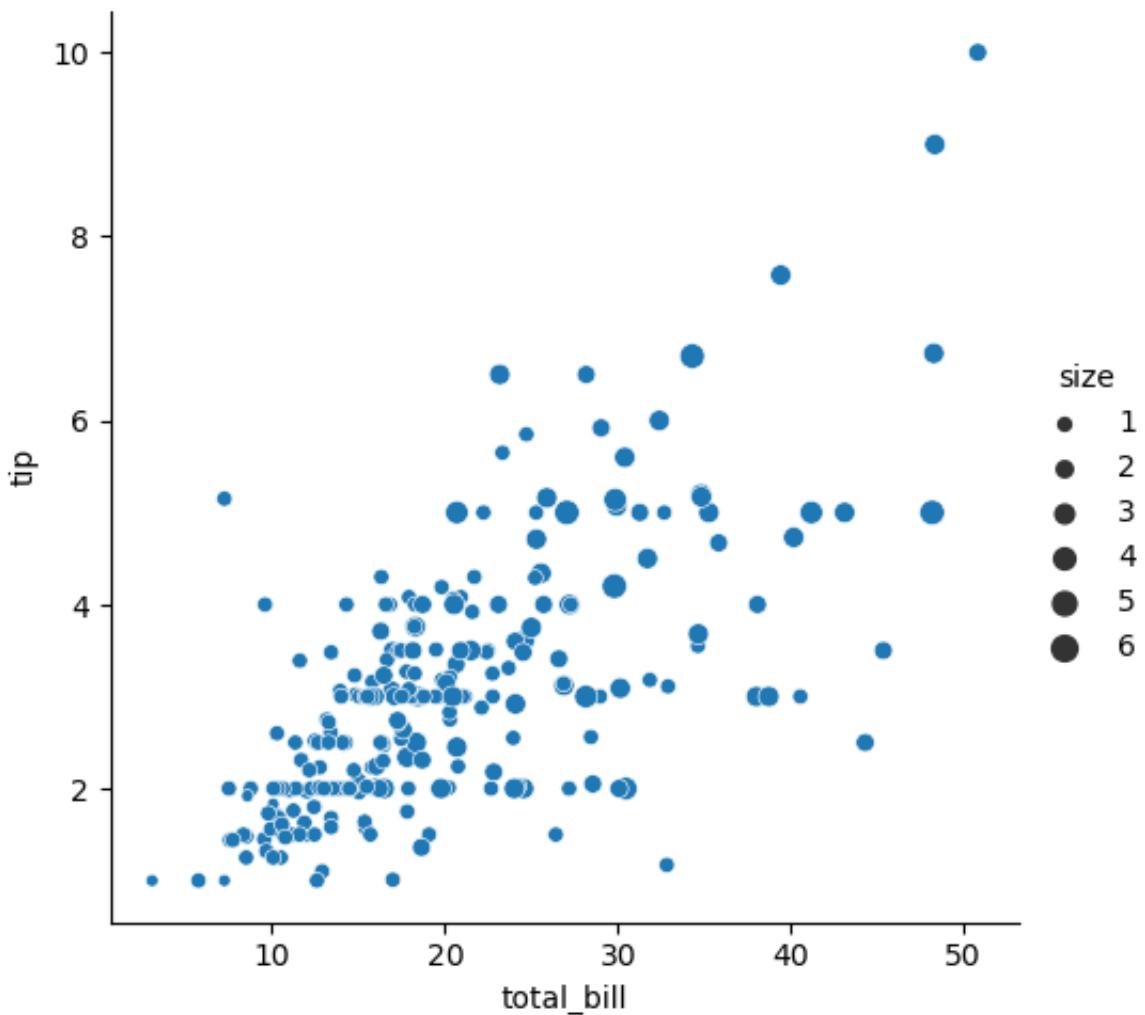
Если семантика оттенка (hue) является числовой (в частности, если ее можно преобразовать в число float), цветовая гамма по умолчанию переключается на последовательную палитру:

```
In [7]: sns.relplot(x= 'total_bill', y= 'tip', hue= 'size', data= tip);
```



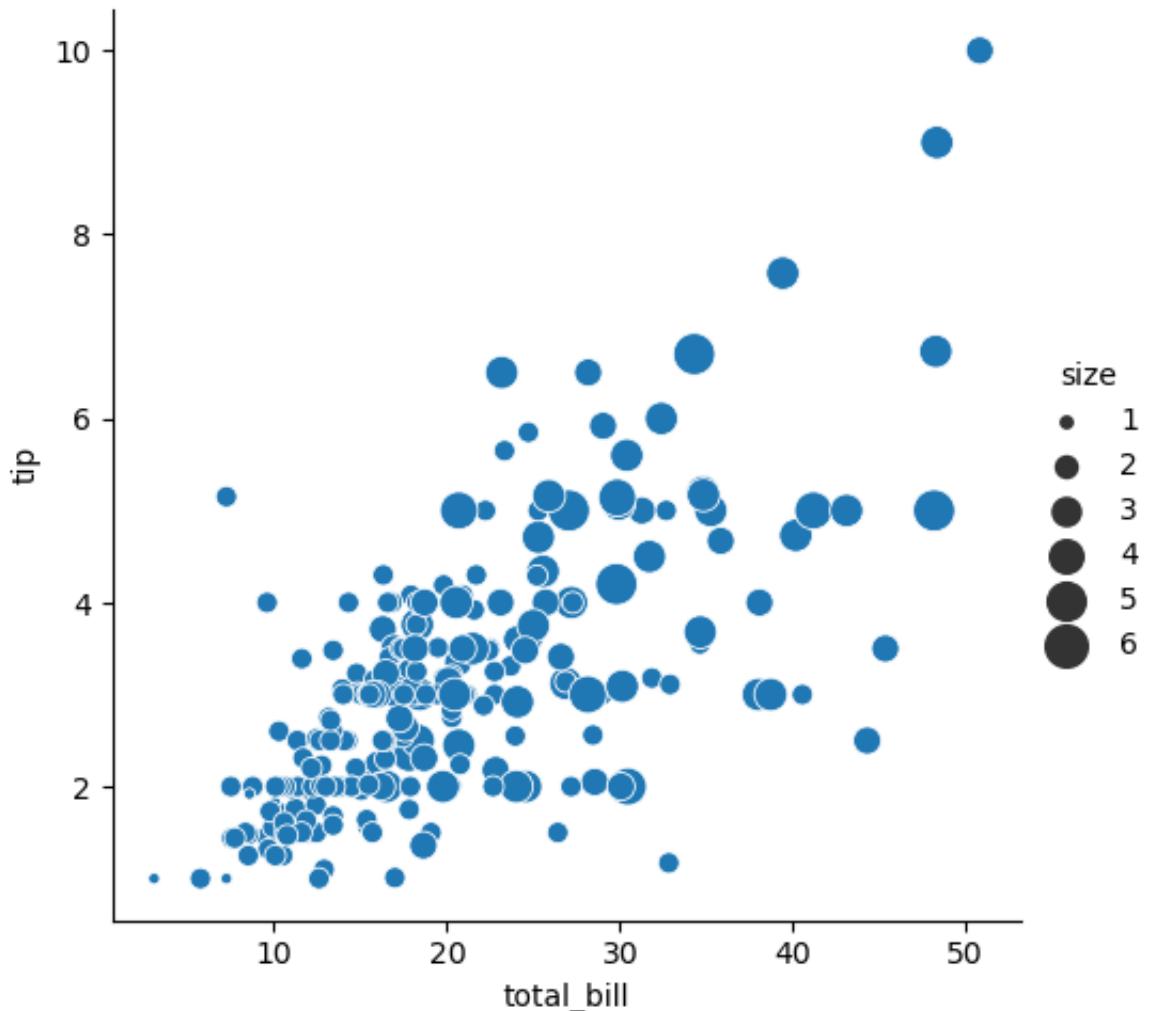
Третий тип семантической переменной изменяет размер каждой точки:

```
In [8]: sns.relplot(x= 'total_bill', y= 'tip', size= 'size', data= tip);
```



Этот диапазон размеров может быть изменен индивидуально:

```
In [9]: sns.relplot(x= 'total_bill', y= 'tip', size= 'size', sizes= (15,200))
```



Линейные графики

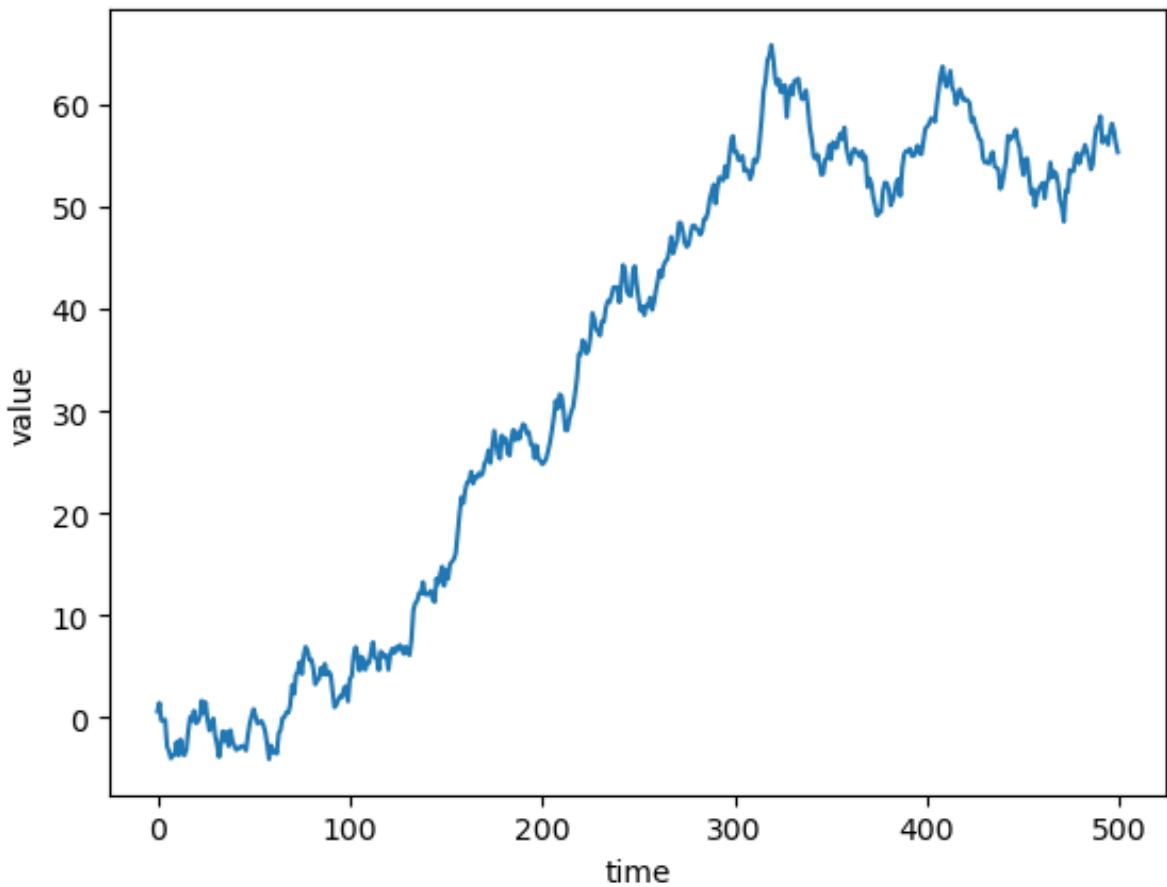
Для некоторых наборов данных может потребоваться представить изменения одной переменной как функцию времени или аналогичной непрерывной переменной. В такой ситуации хорошим выбором будет построение линейного графика. В Seaborn это можно сделать с помощью функции `lineplot()`, как прямую, так и с помощью `relplot()`, установив `kind="line"`:

```
In [10]: df= pd.DataFrame(dict(time= np.arange(500), value= np.random.randn(500)))  
df.head()
```

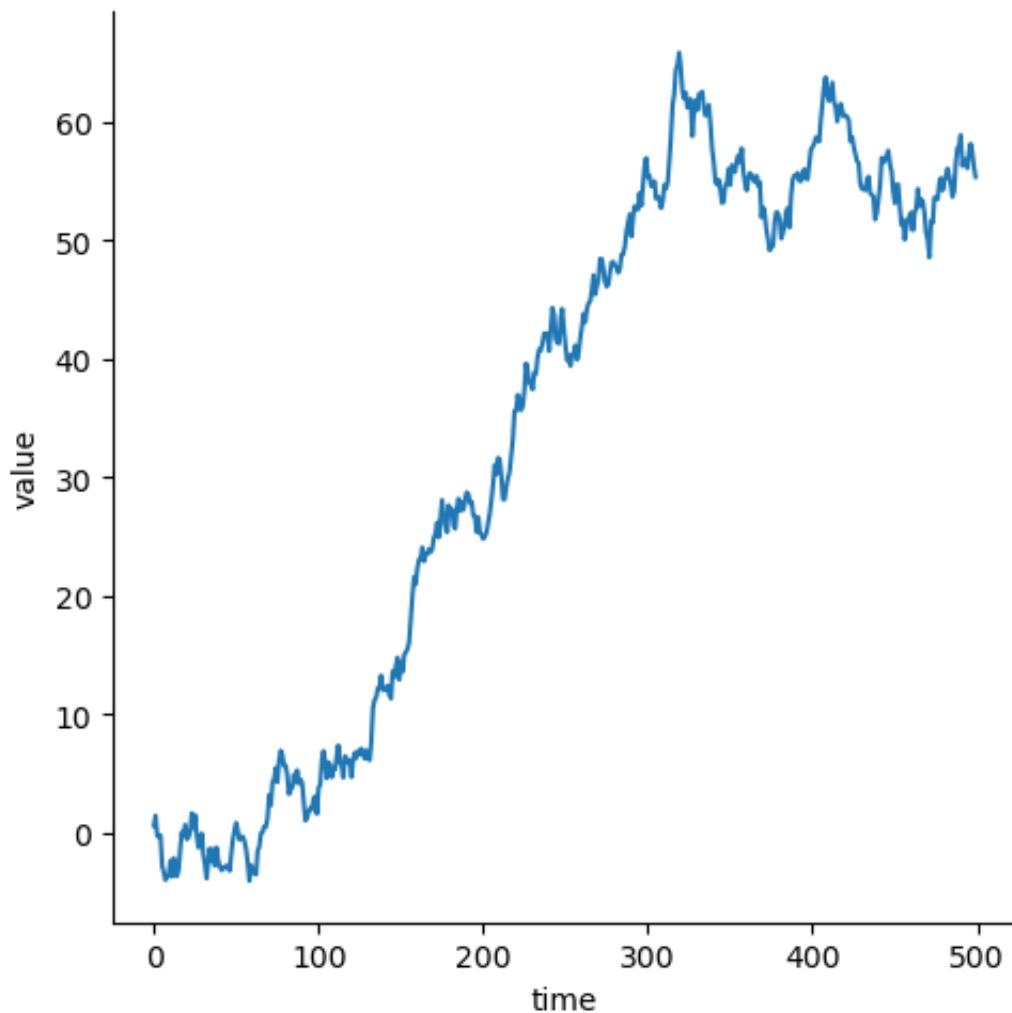
Out [10]:

	time	value
0	0	0.627231
1	1	1.449128
2	2	-0.255407
3	3	-0.318818
4	4	-0.195096

```
In [11]: sns.lineplot( x= 'time', y= 'value', data= df);
```

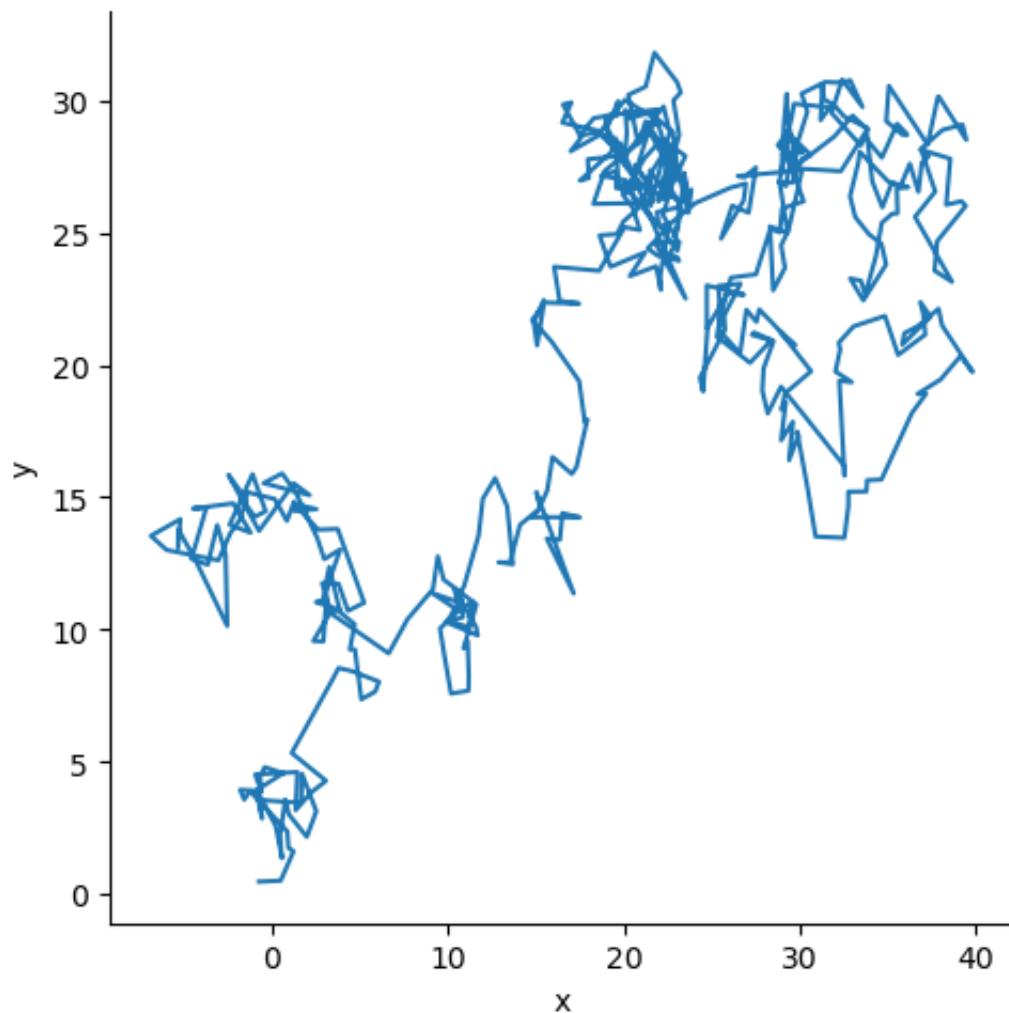


```
In [12]: sns.relplot(x= 'time', y= 'value', kind= 'line', data= df);
```

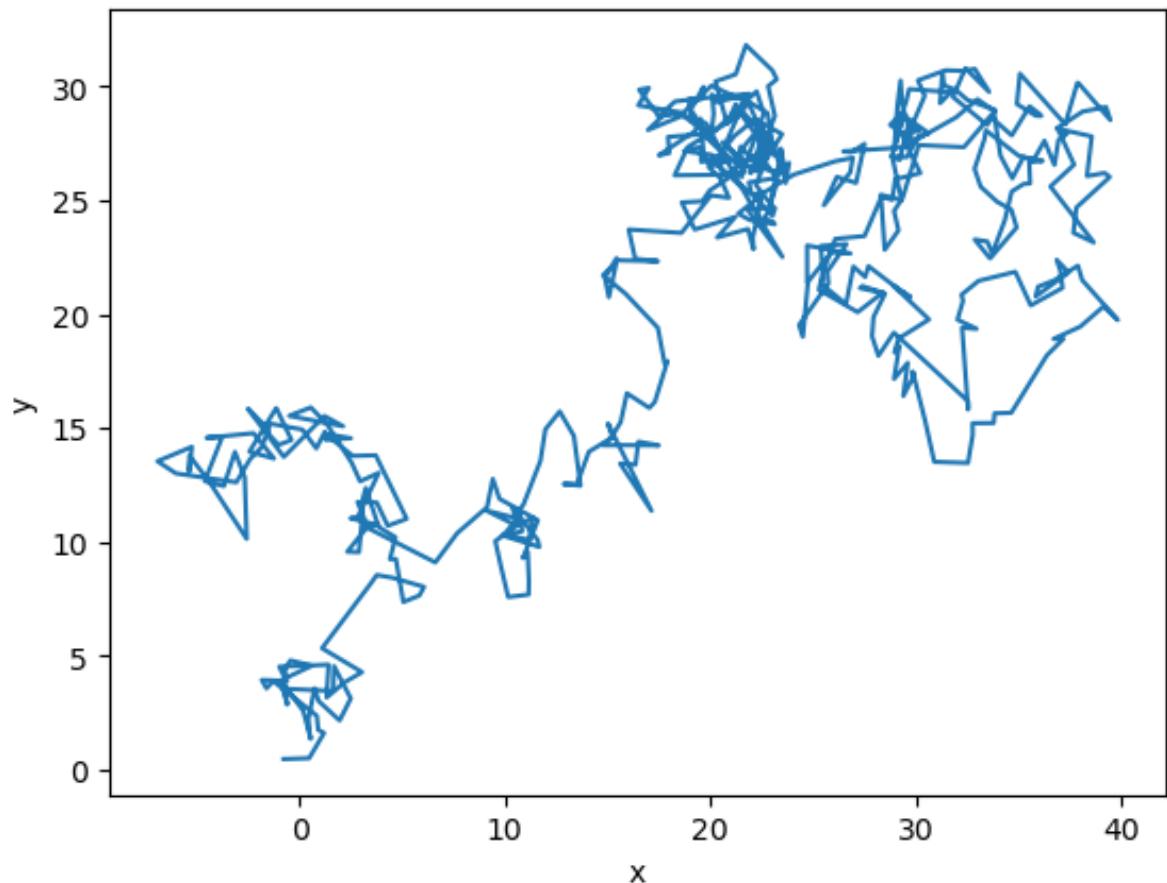


Поскольку функция `lineplot()` предполагает, что вы чаще всего пытаетесь изобразить `y` как функцию `x`, по умолчанию данные сортируются по значениям `x` перед построением графика. Однако это можно отключить:

```
In [13]: df1= pd.DataFrame(np.random.randn(500, 2).cumsum(axis= 0), columns=[  
sns.relplot(x= 'x', y= 'y', data= df1, sort= False, kind= 'line');
```



```
In [14]: sns.lineplot(x= 'x', y= 'y', data= df1, sort= False);
```



Агрегирование и представление неопределенности

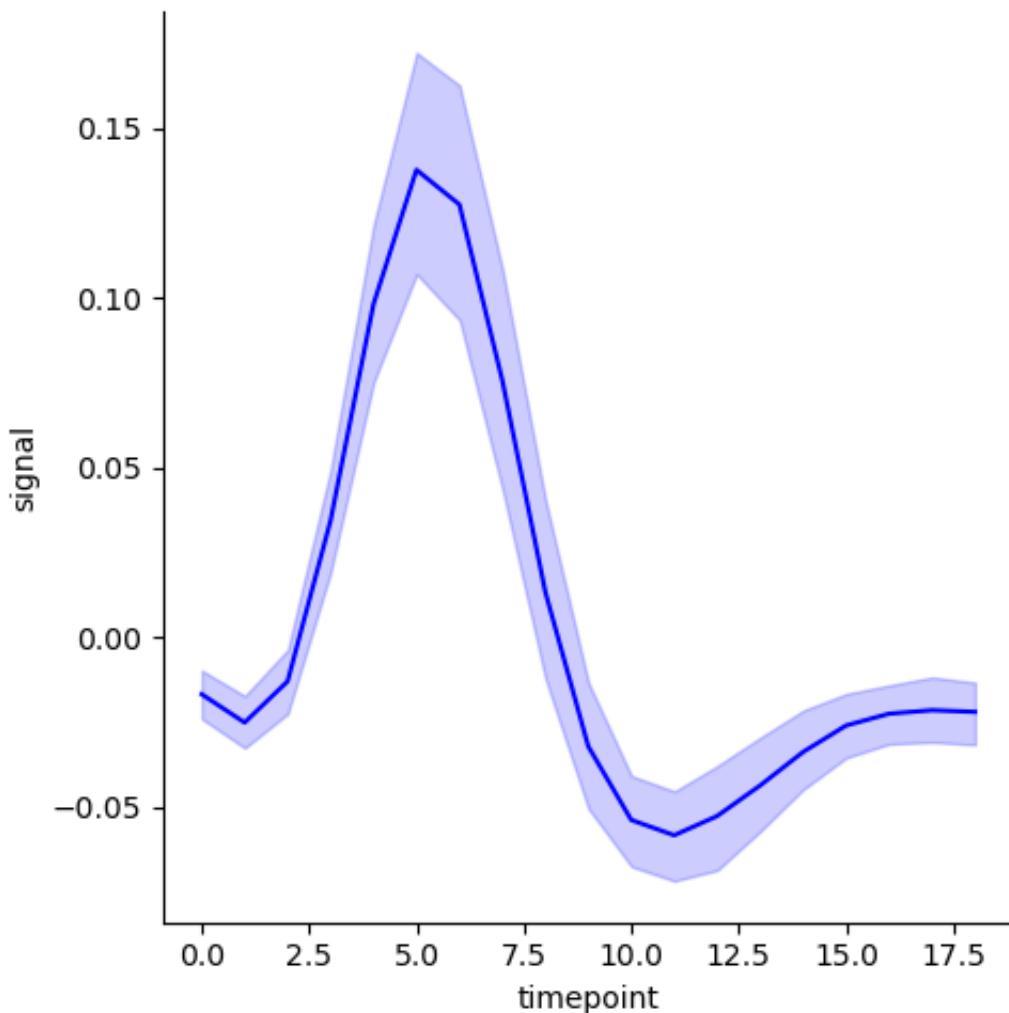
Более сложные наборы данных содержат несколько измерений для одного и того же значения переменной x. По умолчанию в Seaborn агрегируются несколько измерений для каждого значения x путем построения графика среднего значения и 95% доверительного интервала вокруг него:

```
In [15]: fmri= sns.load_dataset('fmri')
fmri.head()
```

Out [15]:

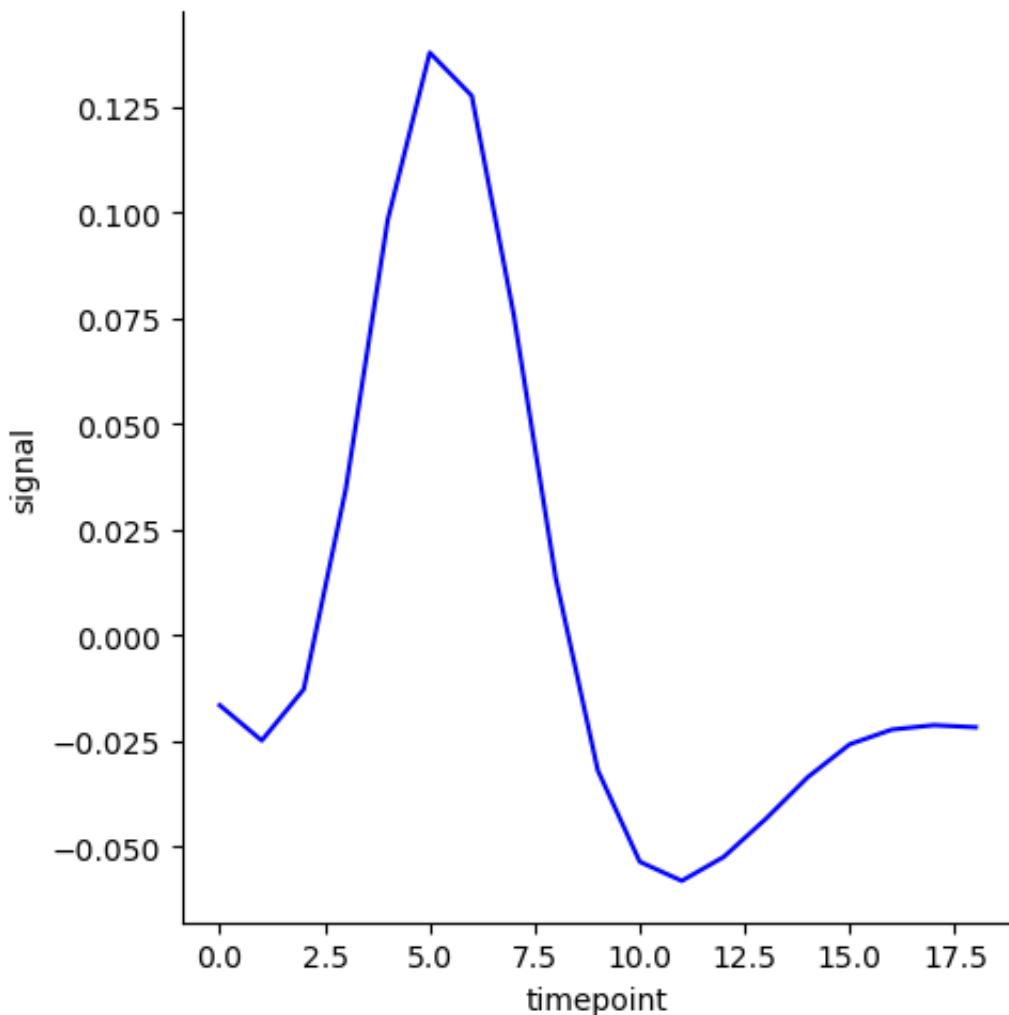
	subject	timepoint	event	region	signal
0	s13	18	stim	parietal	-0.017552
1	s5	14	stim	parietal	-0.080883
2	s12	18	stim	parietal	-0.081033
3	s11	18	stim	parietal	-0.046134
4	s10	18	stim	parietal	-0.037970

```
In [16]: sns.relplot(x= 'timepoint', y= 'signal', kind= 'line', data= fmri,
```



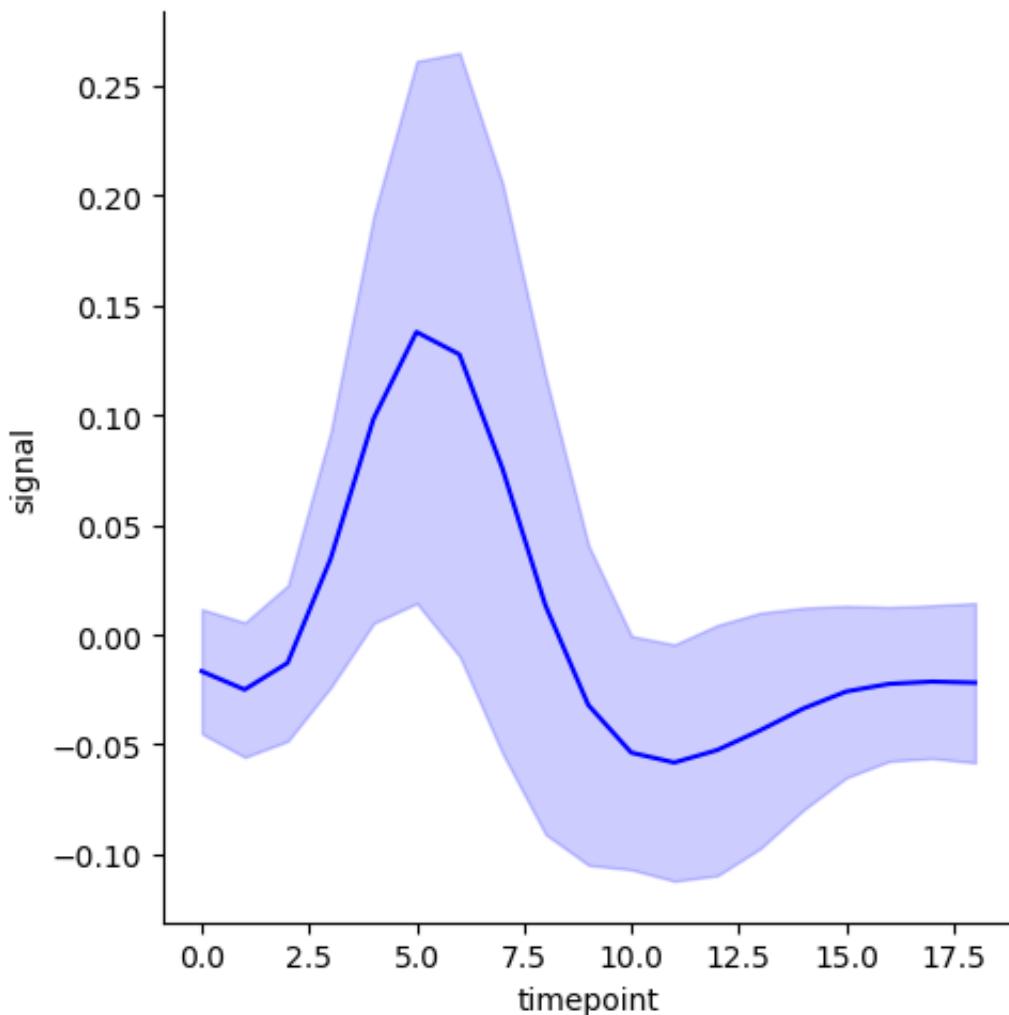
Вычисление доверительных интервалов может быть трудоёмким для больших наборов данных. Поэтому их можно отключить:

```
In [17]: sns.relplot(x= 'timepoint', y= 'signal', kind= 'line', data= fmri, )
```



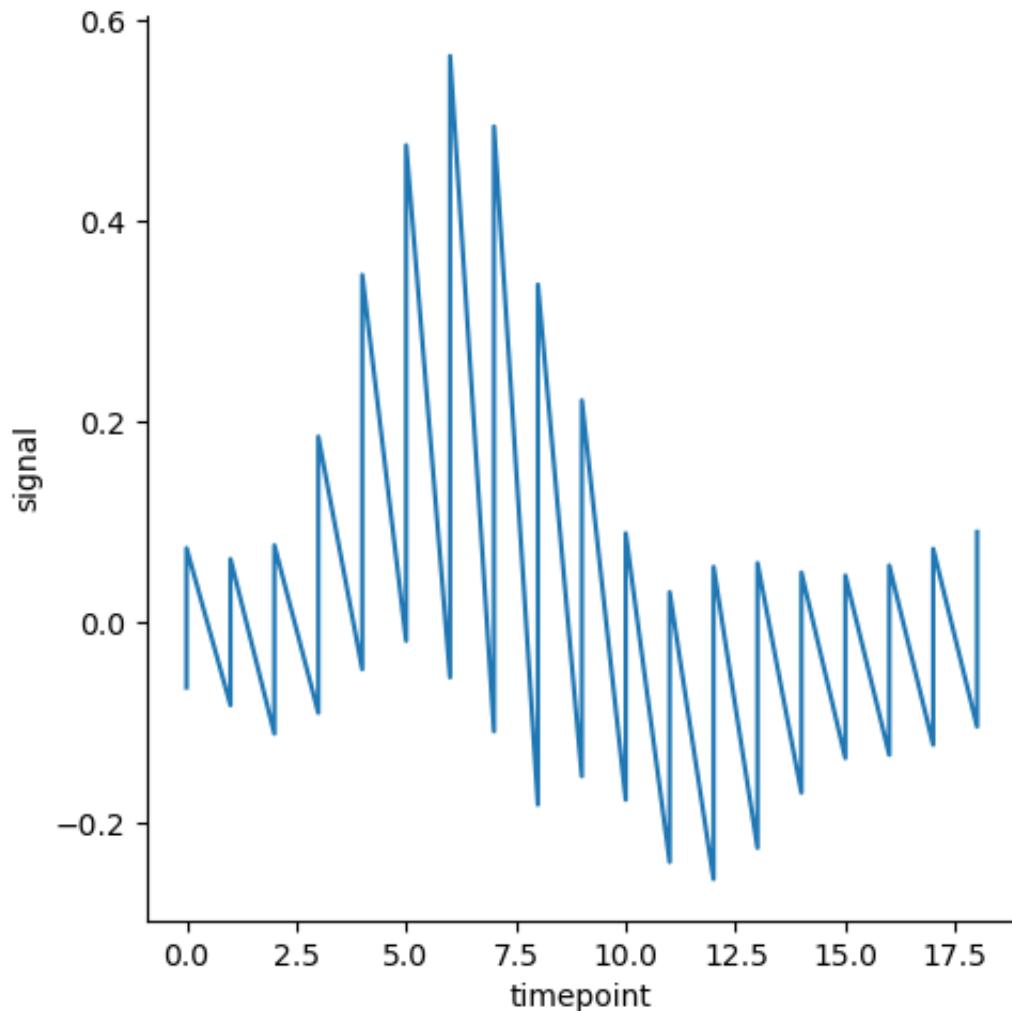
Другой хороший вариант, особенно при наличии больших объемов данных, — представить разброс распределения в каждой точке времени, построив график стандартного отклонения вместо доверительного интервала:

```
In [18]: sns.relplot(x= 'timepoint', y= 'signal', kind= 'line', data= fmri, )
```



Чтобы полностью отключить агрегацию, установите параметр `estimator` в `None`. Это может привести к странному эффекту, если данные содержат несколько наблюдений в каждой точке.

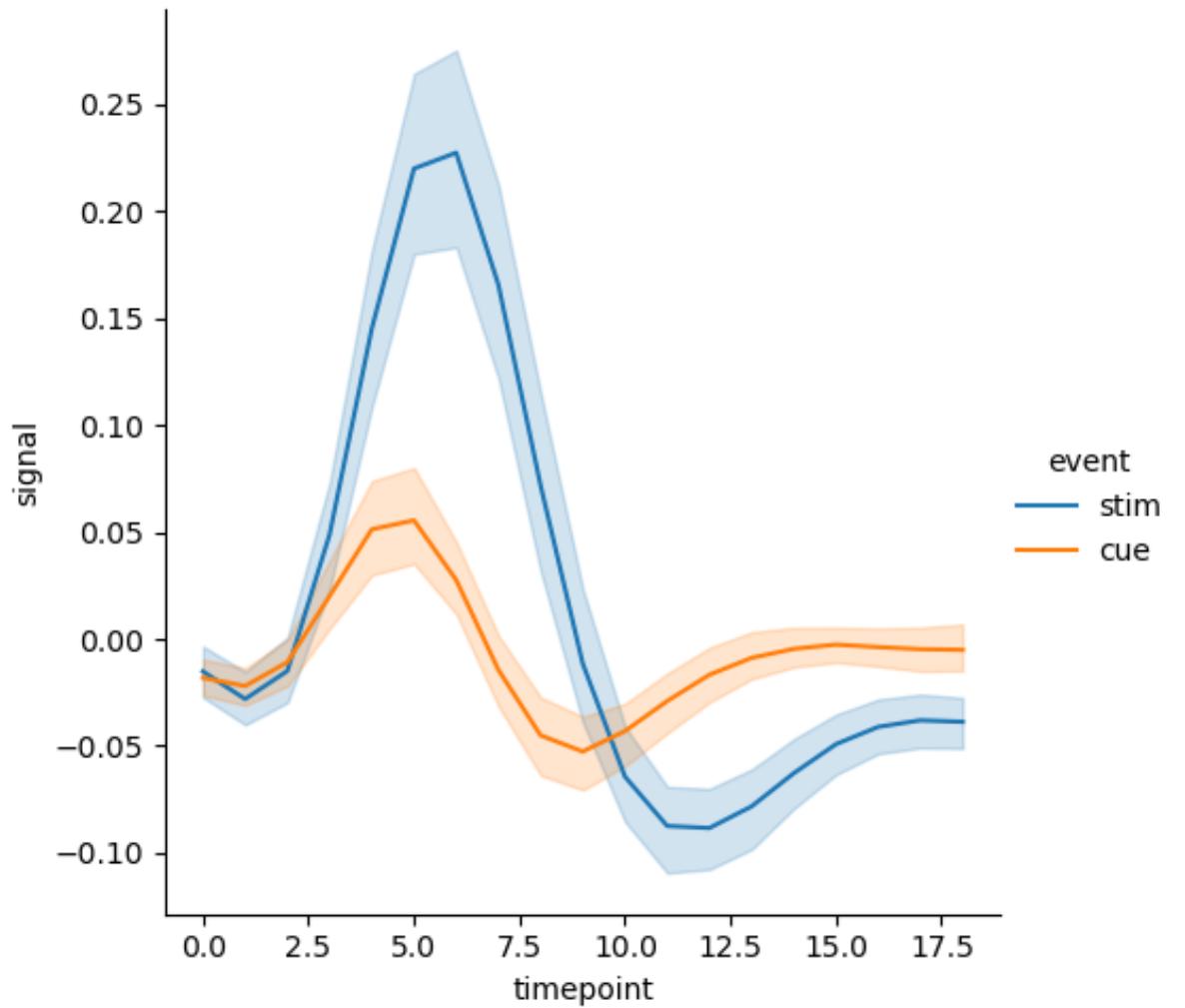
```
In [19]: sns.relplot(x= 'timepoint', y= 'signal', data= fmri, kind= 'line',
```



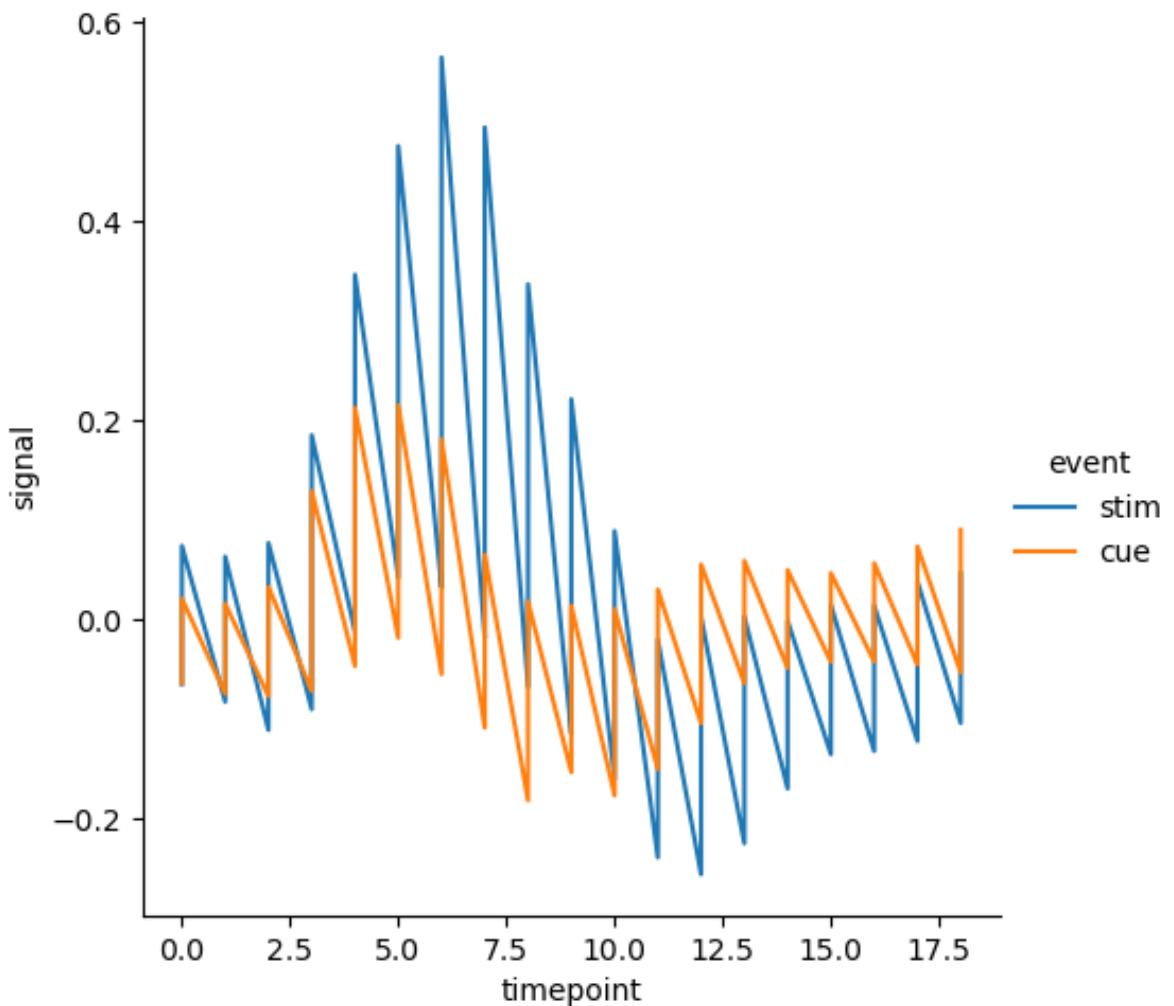
Построение графиков подмножеств данных

Функция `lineplot()` обладает такой же гибкостью, как и `scatterplot()` : она может отображать до трёх дополнительных переменных, изменяя оттенок, размер и стиль элементов графика.

```
In [20]: sns.relplot(x= 'timepoint', y= 'signal', hue= 'event', data= fmri,
```

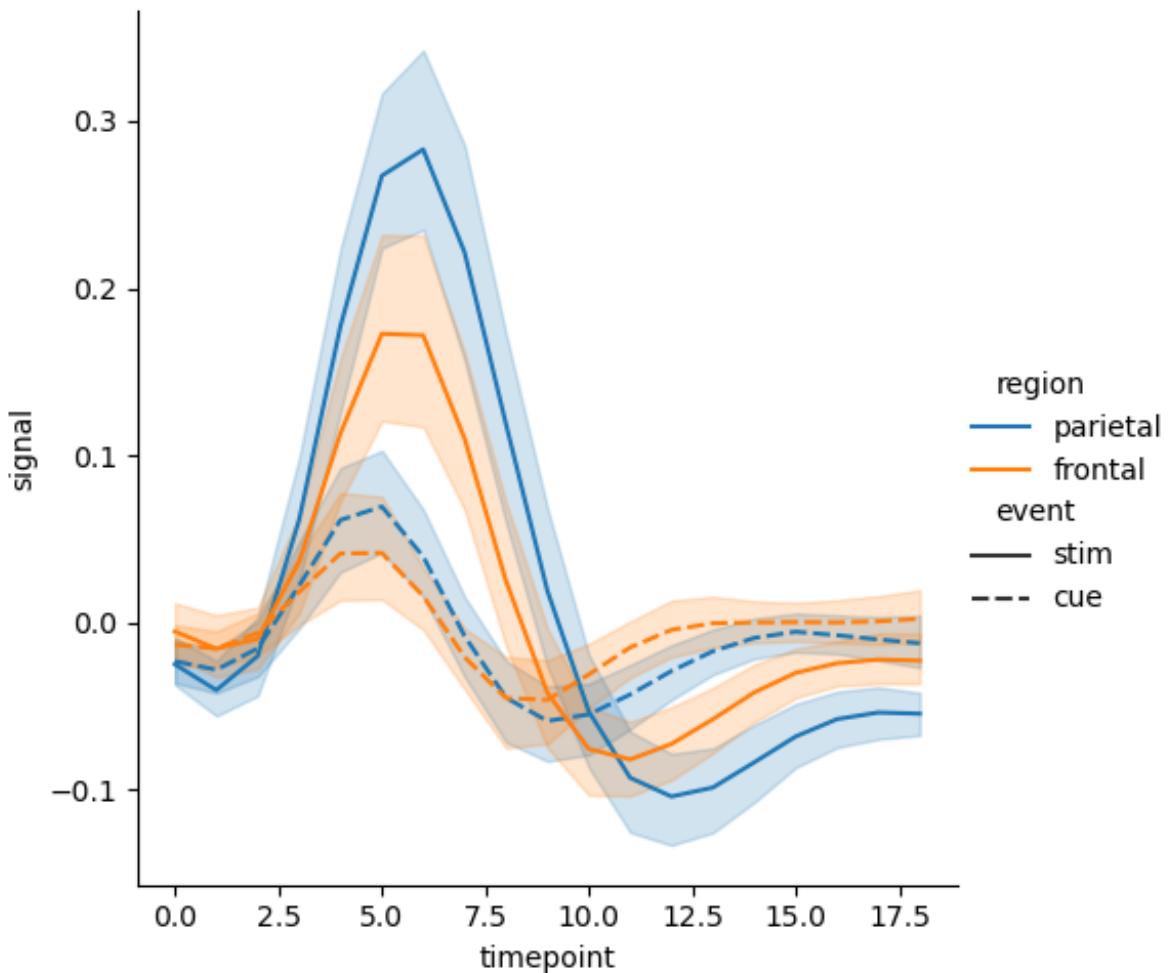


```
In [21]: sns.relplot(x= 'timepoint', y= 'signal', hue= 'event', data= fmri,
```



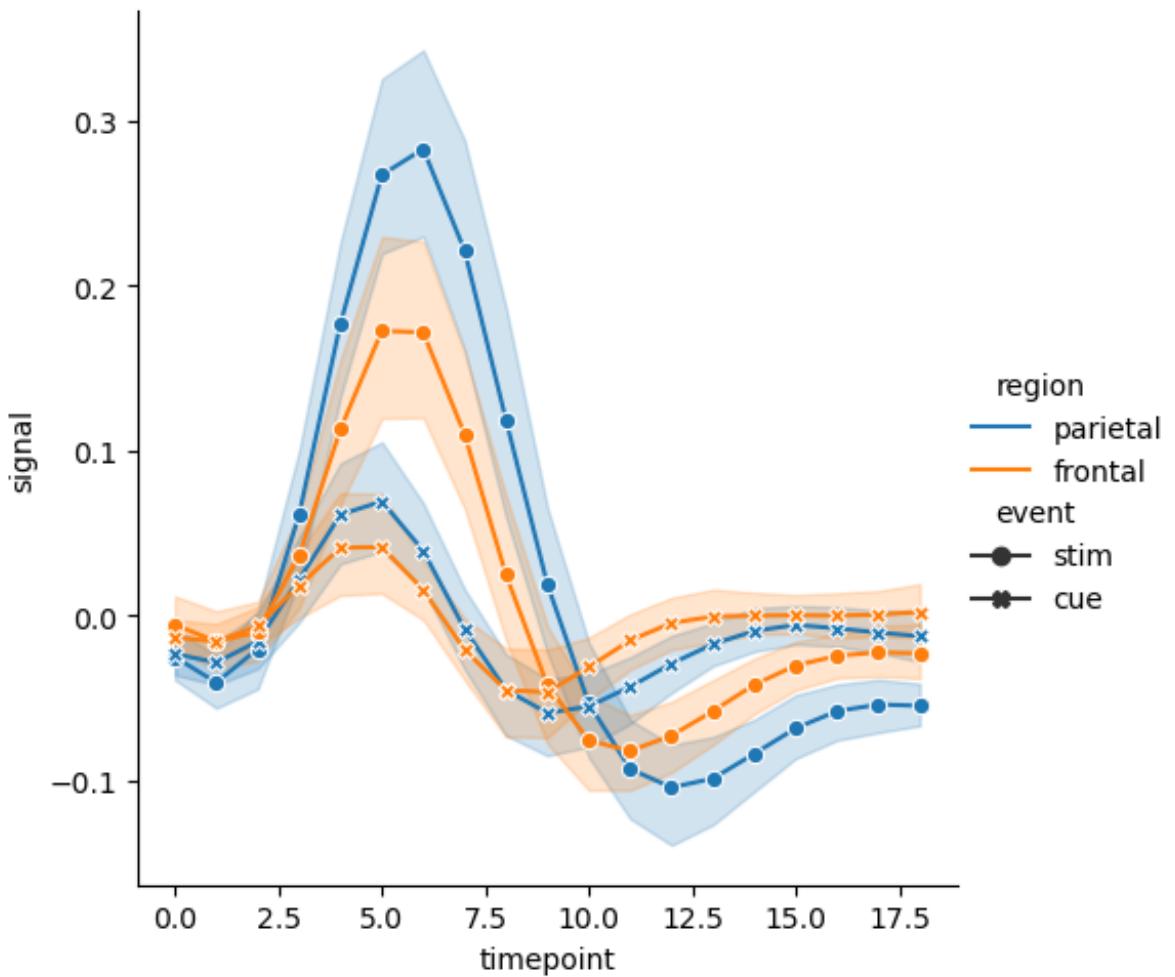
Добавление семантики стиля к линейному графику изменяет
рисунок пунктиротов в линии по умолчанию:

```
In [22]: sns.relplot(x= 'timepoint', y= 'signal', hue= 'region', style= 'eve
```



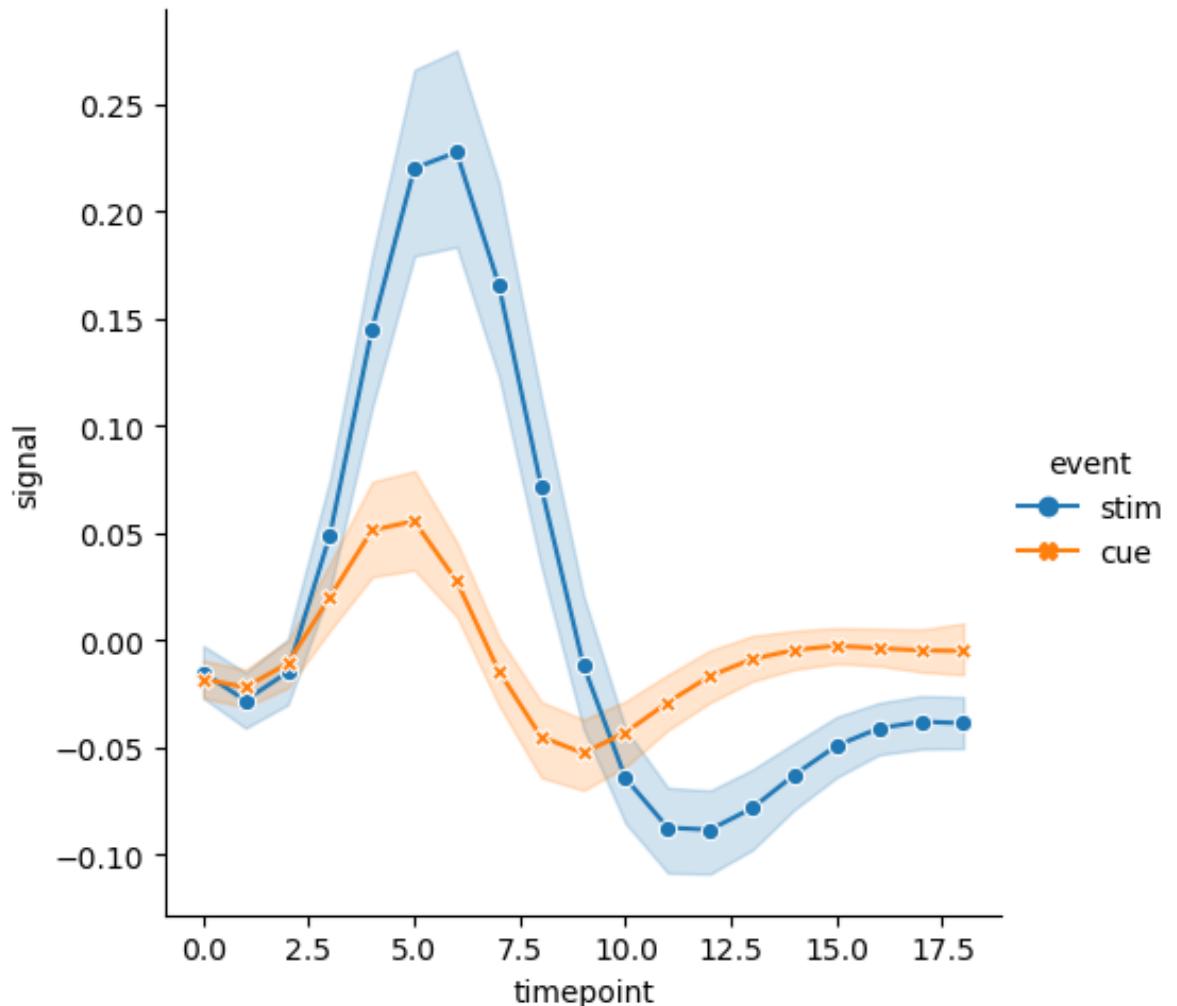
Но вы можете идентифицировать подмножества по маркерам, используемым при каждом наблюдении, либо вместе с тире, либо вместо них:

```
In [23]: sns.relplot(x= 'timepoint', y= 'signal', hue= 'region', style= 'event', kind= 'line', markers= True, dashes= False);
```



Даже если вы изучаете изменения только одной дополнительной переменной, может быть полезно изменить цвет и стиль линий. Это может сделать график более понятным при печати в чёрно-белом формате или для просмотра человеком с цветовой слепотой:

```
In [24]: sns.relplot(x= 'timepoint', y= 'signal', hue= 'event', style= 'event',
                     kind= 'line', markers= True, dashes= False);
```



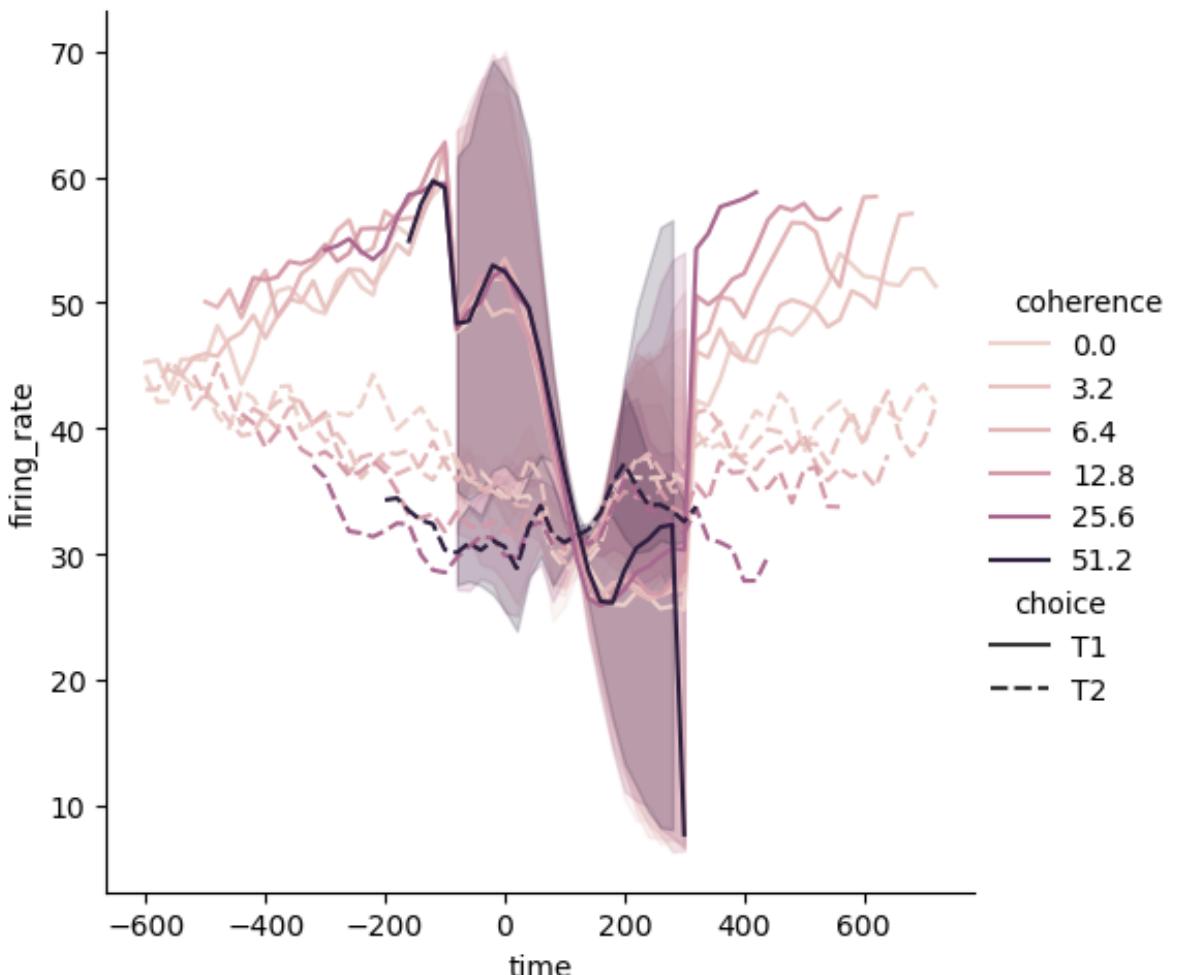
Цветовая карта по умолчанию и обработка легенды в `lineplot()` также зависят от того, является ли семантика оттенка категориальной или числовой:

```
In [25]: dots=sns.load_dataset('dots')
dots.head()
```

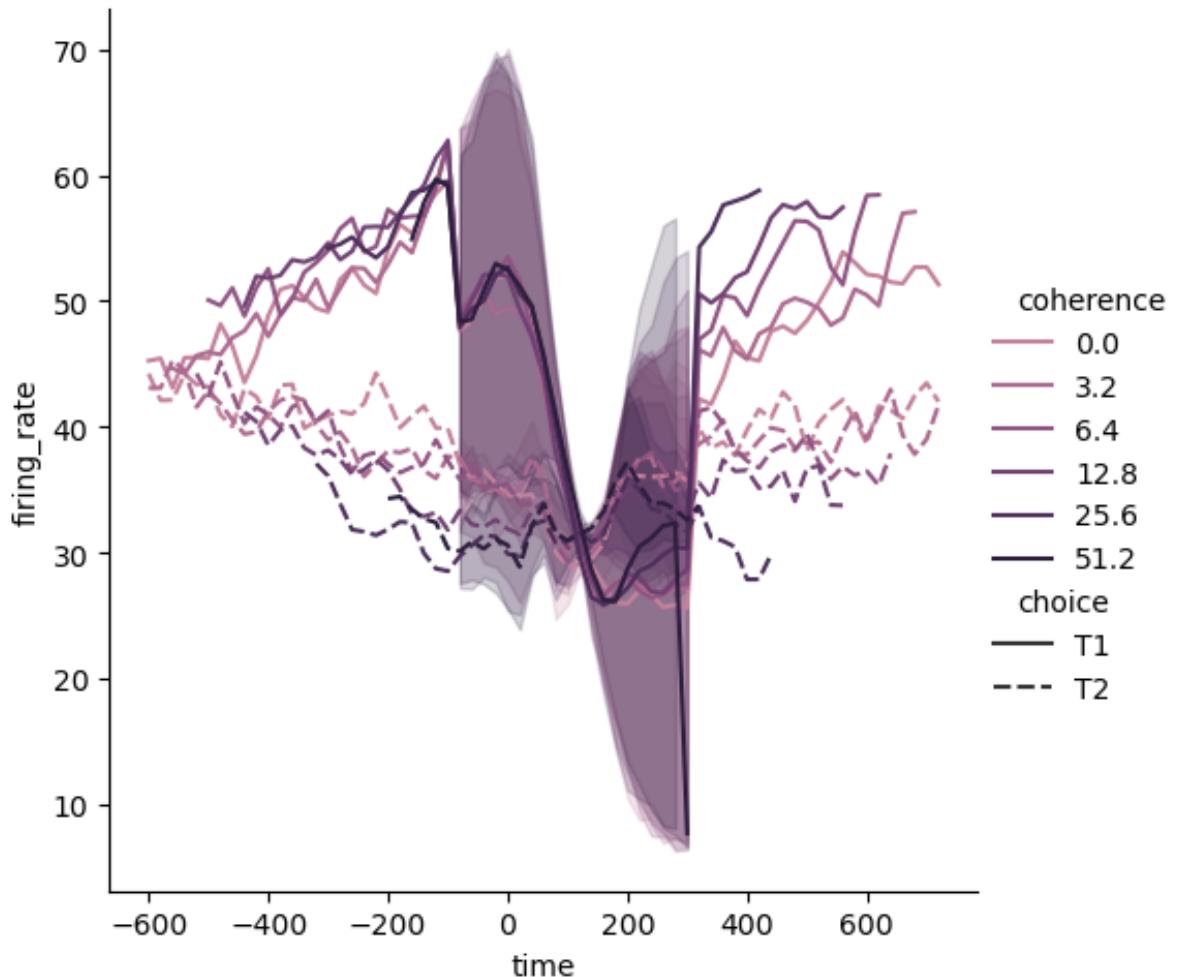
Out [25]:

	align	choice	time	coherence	firing_rate
0	dots	T1	-80	0.0	33.189967
1	dots	T1	-80	3.2	31.691726
2	dots	T1	-80	6.4	34.279840
3	dots	T1	-80	12.8	32.631874
4	dots	T1	-80	25.6	35.060487

```
In [26]: sns.relplot(x= 'time', y= 'firing_rate', data= dots, kind= 'line',
```



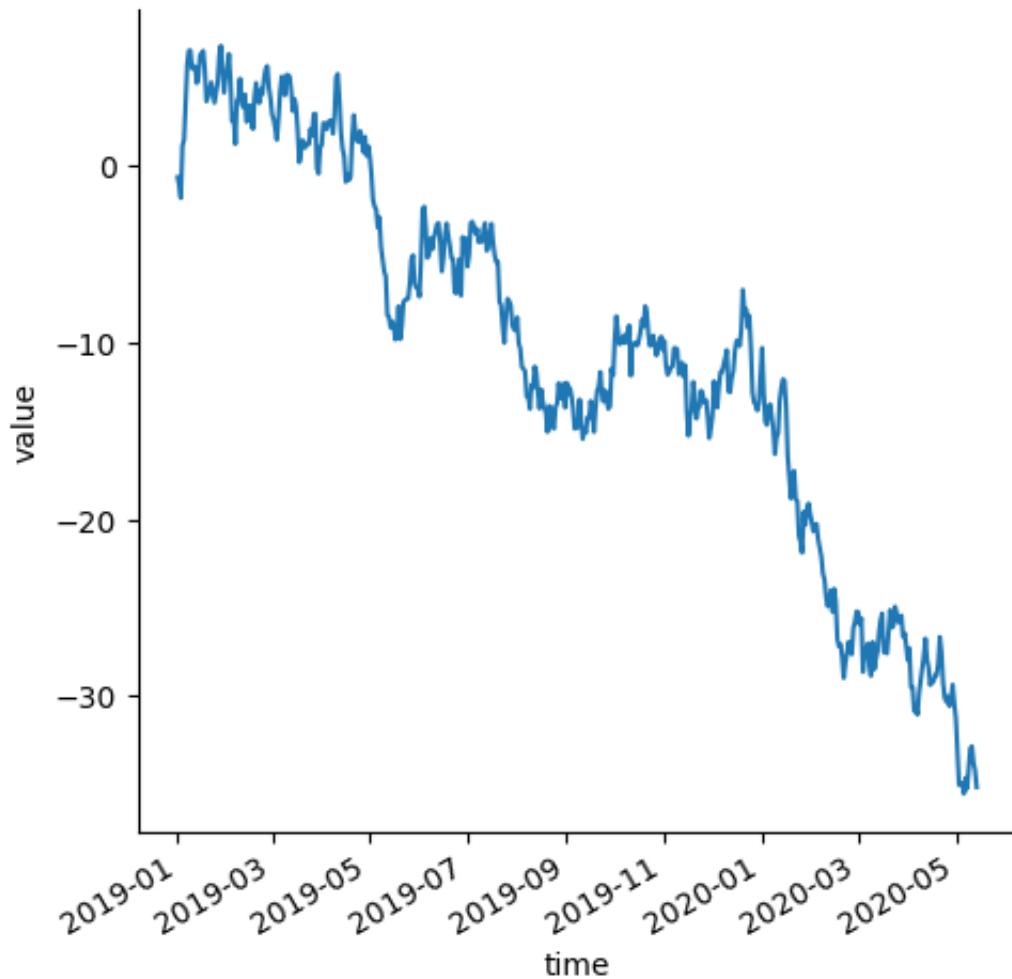
```
In [27]: palette= sns.cubehelix_palette(light= 0.6, n_colors= 6)
sns.relplot(x= 'time', y= 'firing_rate', data= dots, kind= 'line',
            hue= 'coherence', style= 'choice', palette= palette);
```



Построение графиков с использованием данных дат

Линейные графики часто используются для визуализации данных, связанных с реальными датами и временем. Эти функции передают данные в исходном формате базовым функциям matplotlib, что позволяет им использовать возможности matplotlib по форматированию дат в виде меток делений. Однако всё это форматирование должно выполняться на уровне matplotlib:

```
In [28]: df2= pd.DataFrame(dict(time= pd.date_range('2019-1-1', periods= 500
g= sns.relplot(x= 'time', y= 'value', kind= 'line', data= df2)
g.fig.autofmt_xdate());
```



Многооконные рисунки

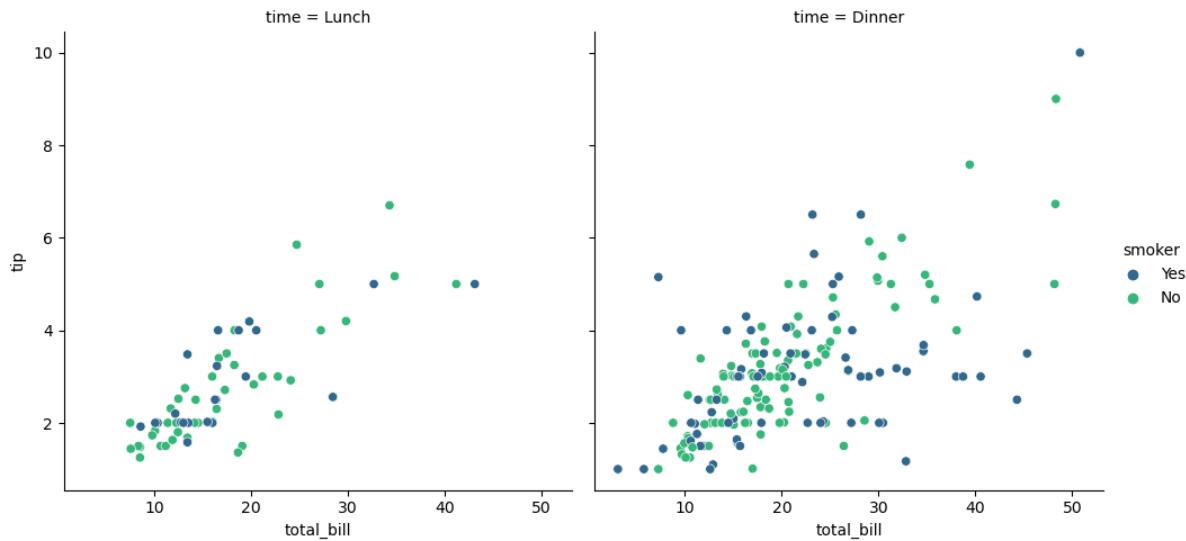
Это означает, что вы создаете несколько рисунков и отображаете подмножества данных на каждом из них:

```
In [29]: tips= sns.load_dataset('tips')
tips.tail()
```

Out [29]:

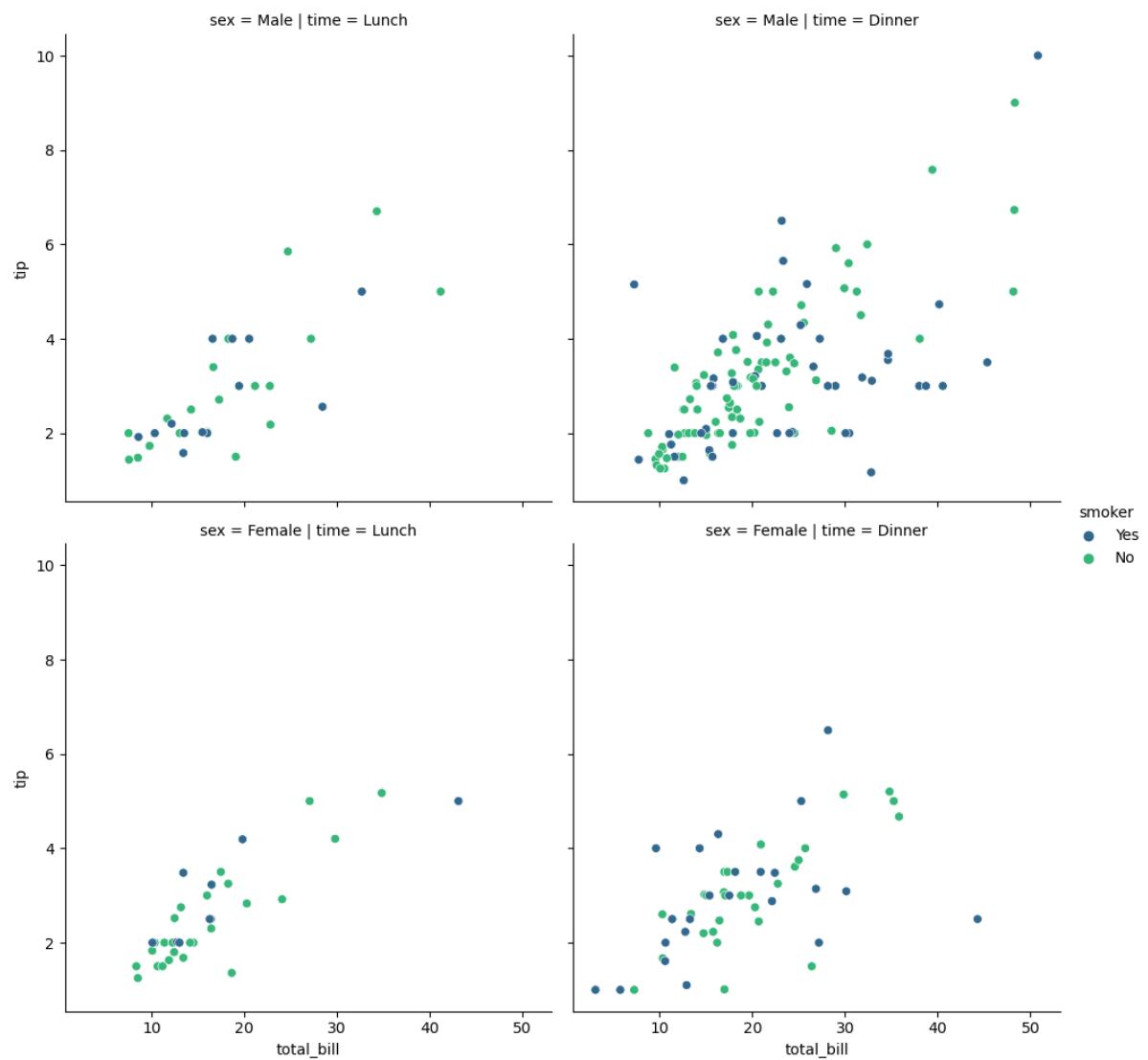
	total_bill	tip	sex	smoker	day	time	size
239	29.03	5.92	Male	No	Sat	Dinner	3
240	27.18	2.00	Female	Yes	Sat	Dinner	2
241	22.67	2.00	Male	Yes	Sat	Dinner	2
242	17.82	1.75	Male	No	Sat	Dinner	2
243	18.78	3.00	Female	No	Thur	Dinner	2

```
In [30]: sns.relplot(x= 'total_bill', y= 'tip', data= tips, hue= 'smoker', c
```

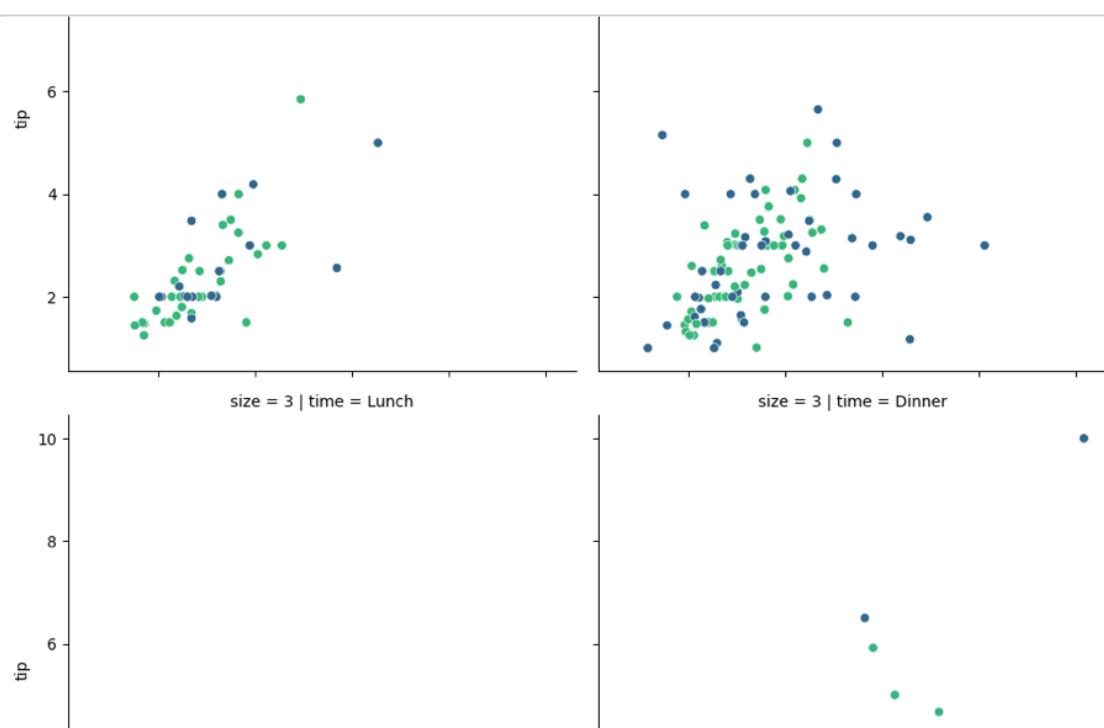


Вы также можете отобразить влияние двух переменных таким образом: одну, используя рисунки по столбцам, и одну, используя рисунки по строкам. По мере добавления переменных в сетку может возникнуть необходимость уменьшить размер рисунка. Помните, что размер объекта `FacetGrid` параметризуется высотой и соотношением сторон каждого рисунка:

```
In [31]: sns.relplot(x= 'total_bill', y= 'tip', data= tips, hue= 'smoker', row
```



```
In [32]: sns.relplot(x= 'total_bill', y= 'tip', data= tips, hue= 'smoker', row
```



Вы также можете отобразить влияние двух переменных таким образом: одну, используя рисунки по столбцам, и одну, используя рисунки по строкам. По мере добавления переменных в сетку может возникнуть необходимость уменьшить размер рисунка. Помните, что размер FacetGrid параметризуется высотой и соотношением сторон каждого рисунка:

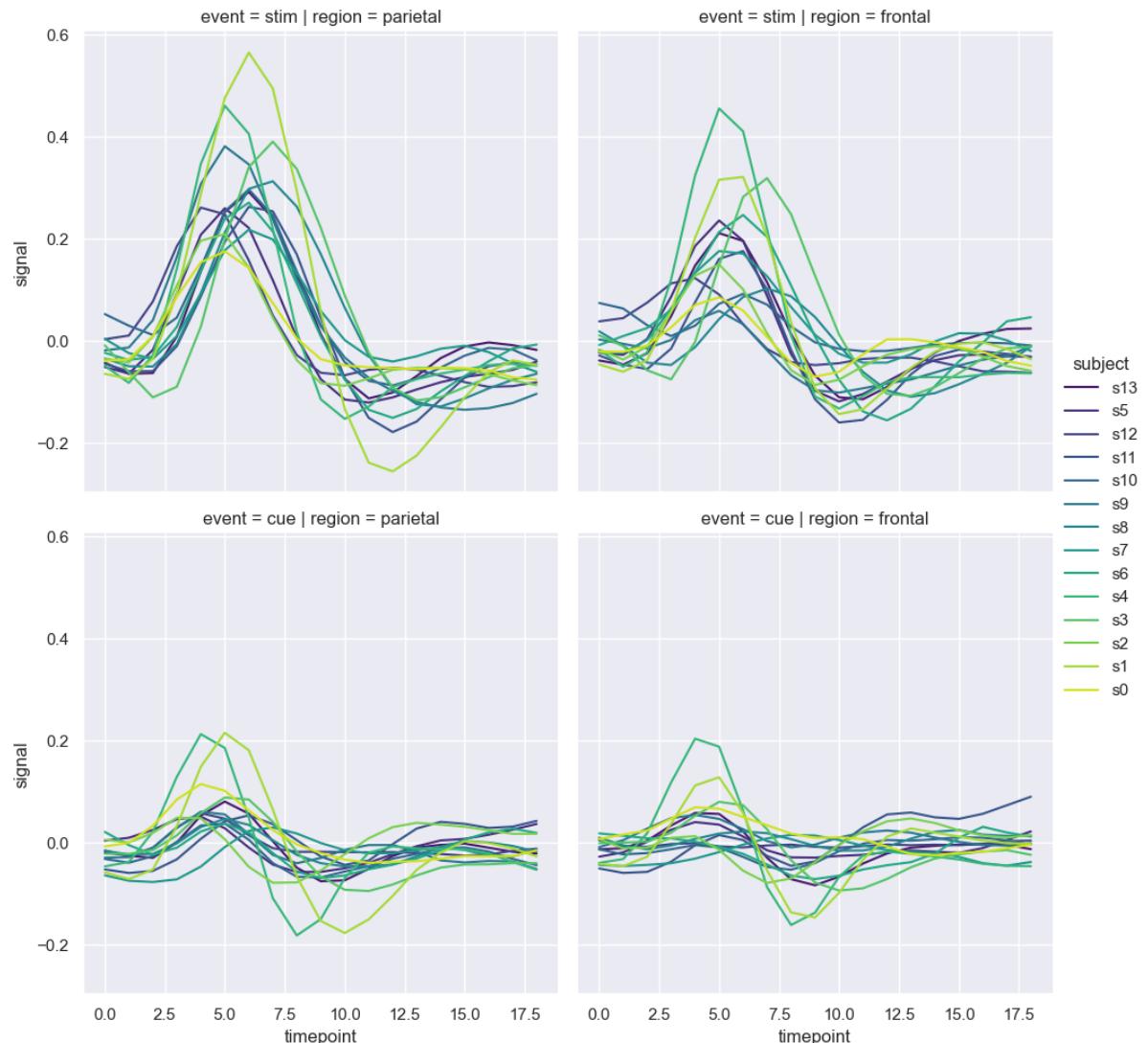
In [33]: `fmri.tail()`

Out[33]:

	subject	timepoint	event	region	signal
1059	s0	8	cue	frontal	0.018165
1060	s13	7	cue	frontal	-0.029130
1061	s12	7	cue	frontal	-0.004939
1062	s11	7	cue	frontal	-0.025367
1063	s0	0	cue	parietal	-0.006899

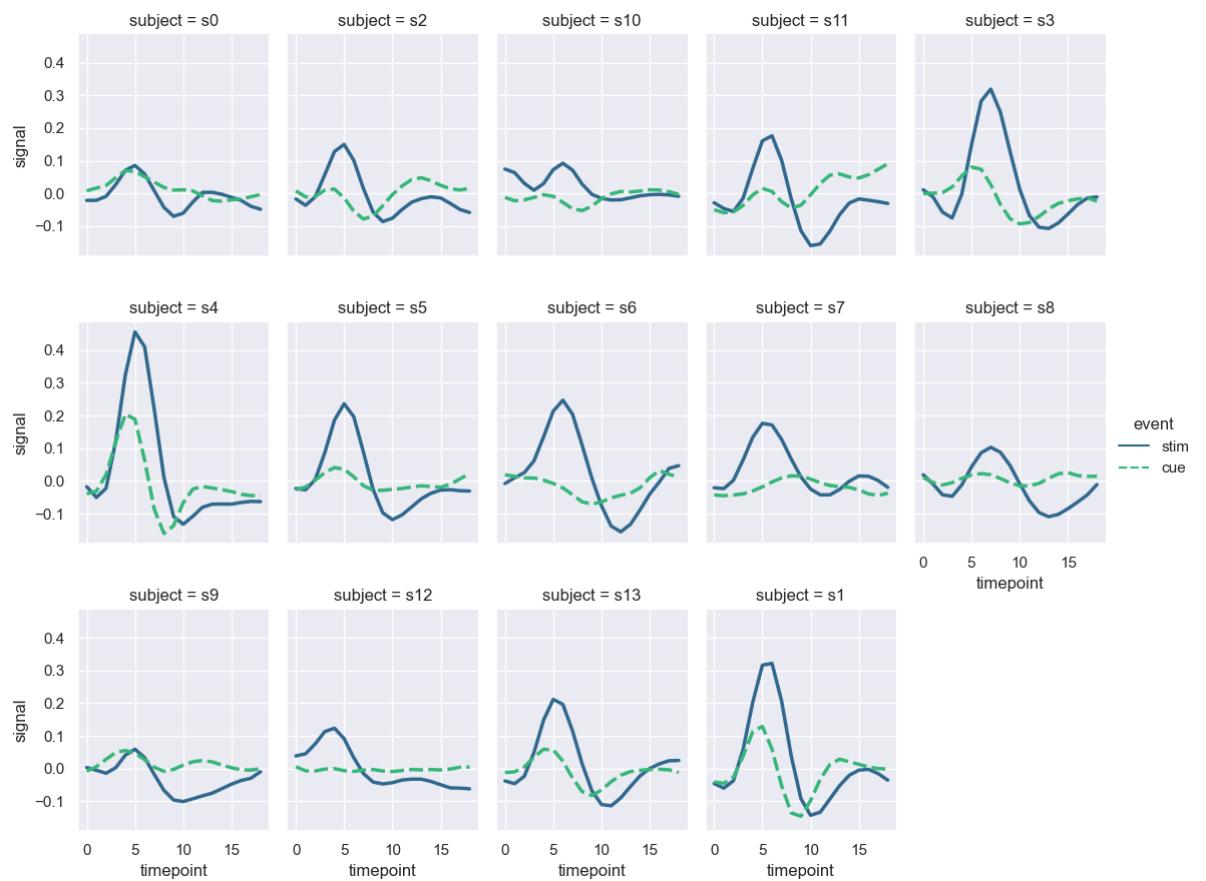
In [34]: `sns.set(style= 'darkgrid')`

```
In [35]: sns.relplot(x= 'timepoint', y= 'signal', hue= 'subject', col= 'region', height= 5, kind= 'line', estimator= None, data= fmri);
```

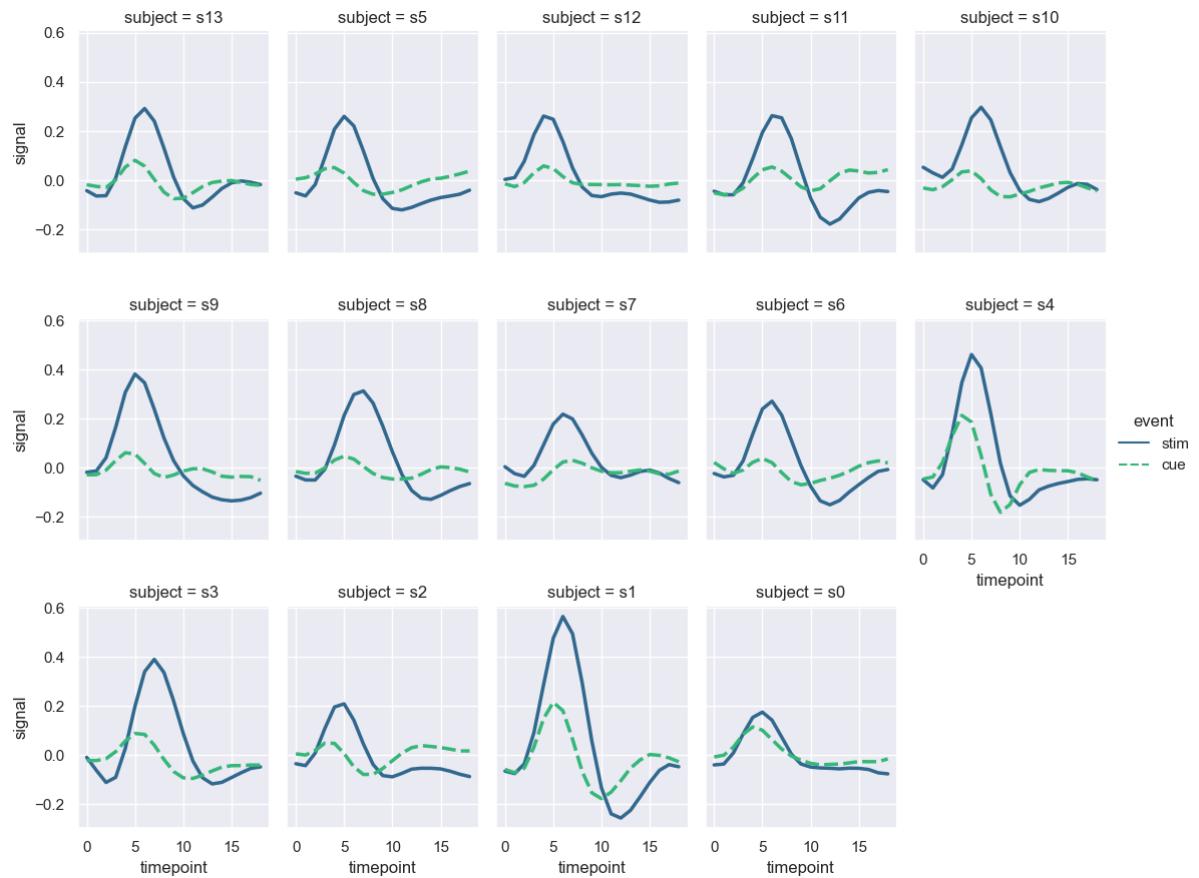


Если вы хотите изучить эффекты на многих уровнях переменной, хорошей идеей может быть разделение этой переменной по столбцам, а затем обертка рисунков в строки:

```
In [36]: sns.relplot(x= 'timepoint', y= 'signal', hue= 'event', style= 'event',
       col_wrap= 5, palette= 'viridis', height= 3, aspect= .75,
       kind= 'line', data= fmri.query(" region== 'frontal' "));
```



```
In [37]: sns.relplot(x= 'timepoint', y= 'signal', hue= 'event', style= 'event',
                     col_wrap= 5, palette= 'viridis', height= 3, aspect= .75,
                     kind= 'line', data= fmri.query(" region== 'parietal' "))
```

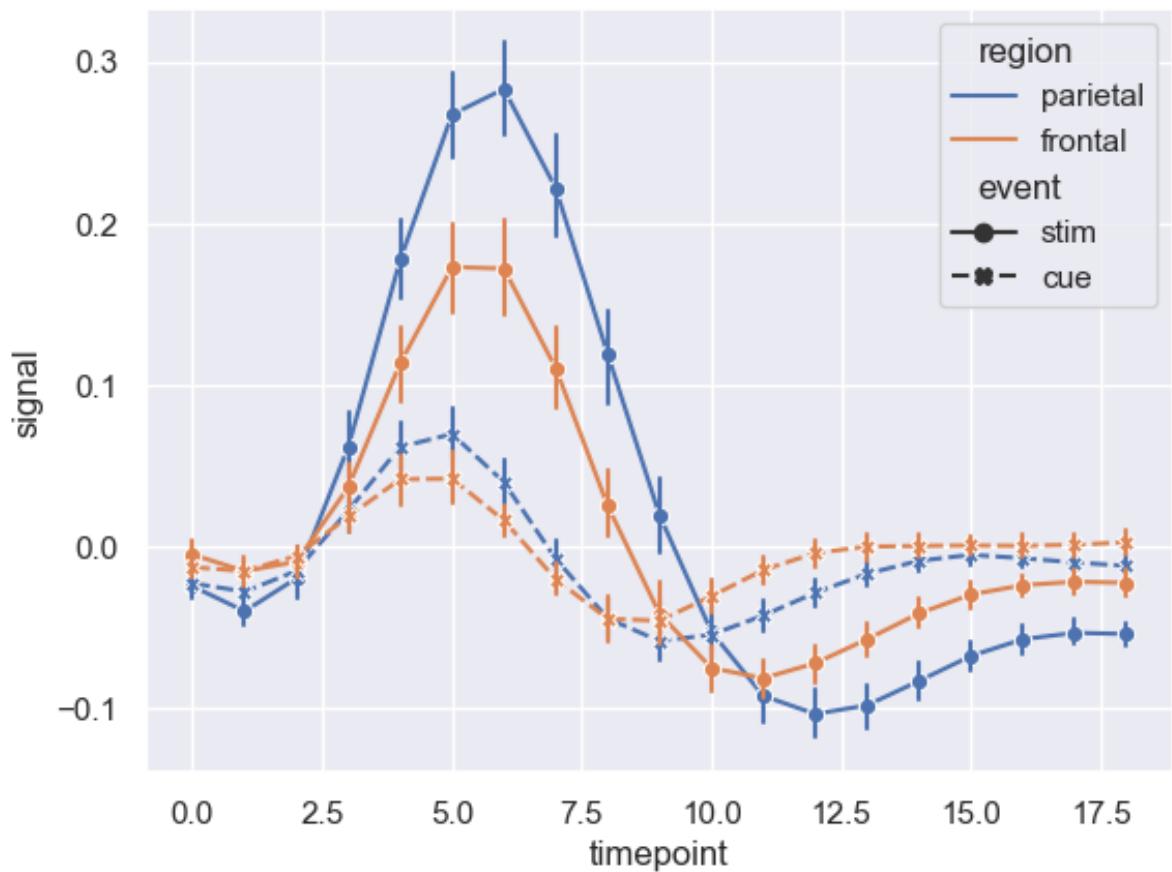


```
In [38]: fmri.head()
```

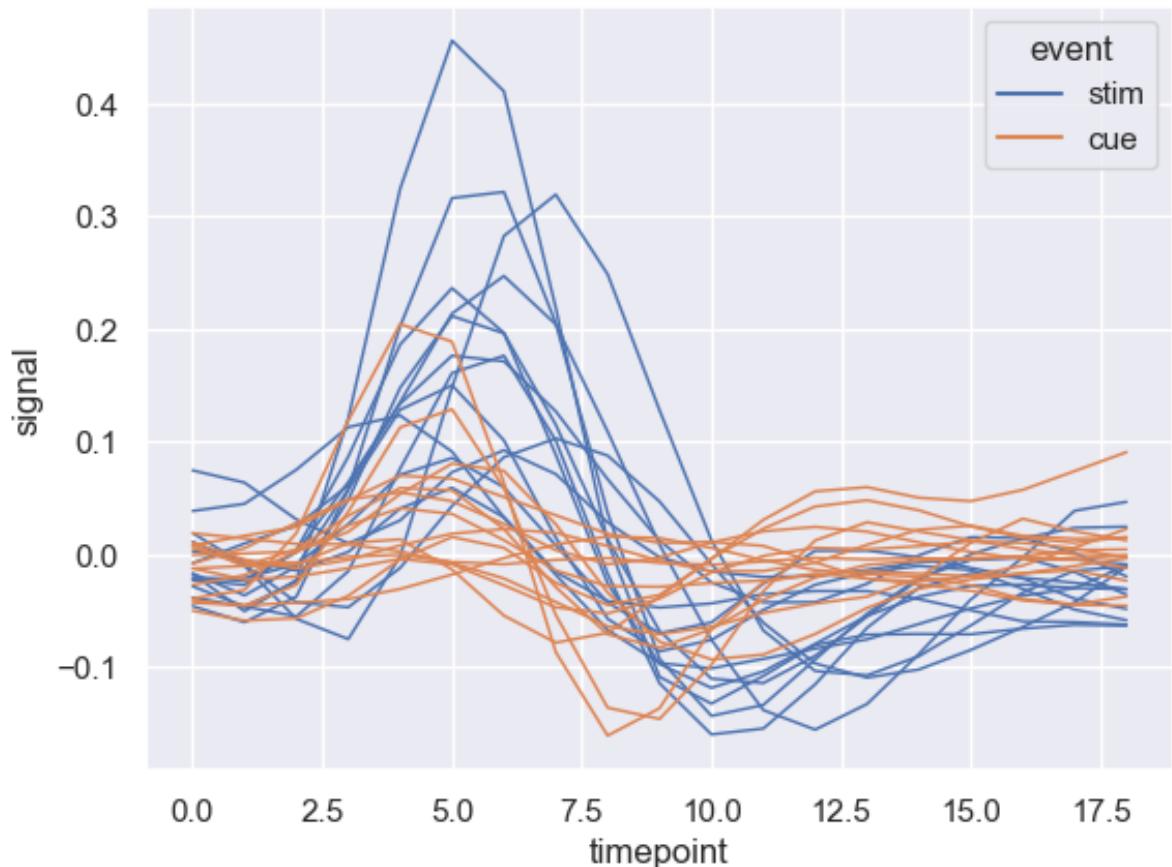
Out [38]:

	subject	timepoint	event	region	signal
0	s13	18	stim	parietal	-0.017552
1	s5	14	stim	parietal	-0.080883
2	s12	18	stim	parietal	-0.081033
3	s11	18	stim	parietal	-0.046134
4	s10	18	stim	parietal	-0.037970

```
In [39]: sns.lineplot(x= 'timepoint', y= 'signal', hue= 'region', style= 'event',
errorbar=('ci', 68), markers= True, err_style= 'bars')
```



```
In [40]: sns.lineplot(x= 'timepoint', y= 'signal', hue= 'event', units= 'sub  
data= fmri.query("region=='frontal'"));
```

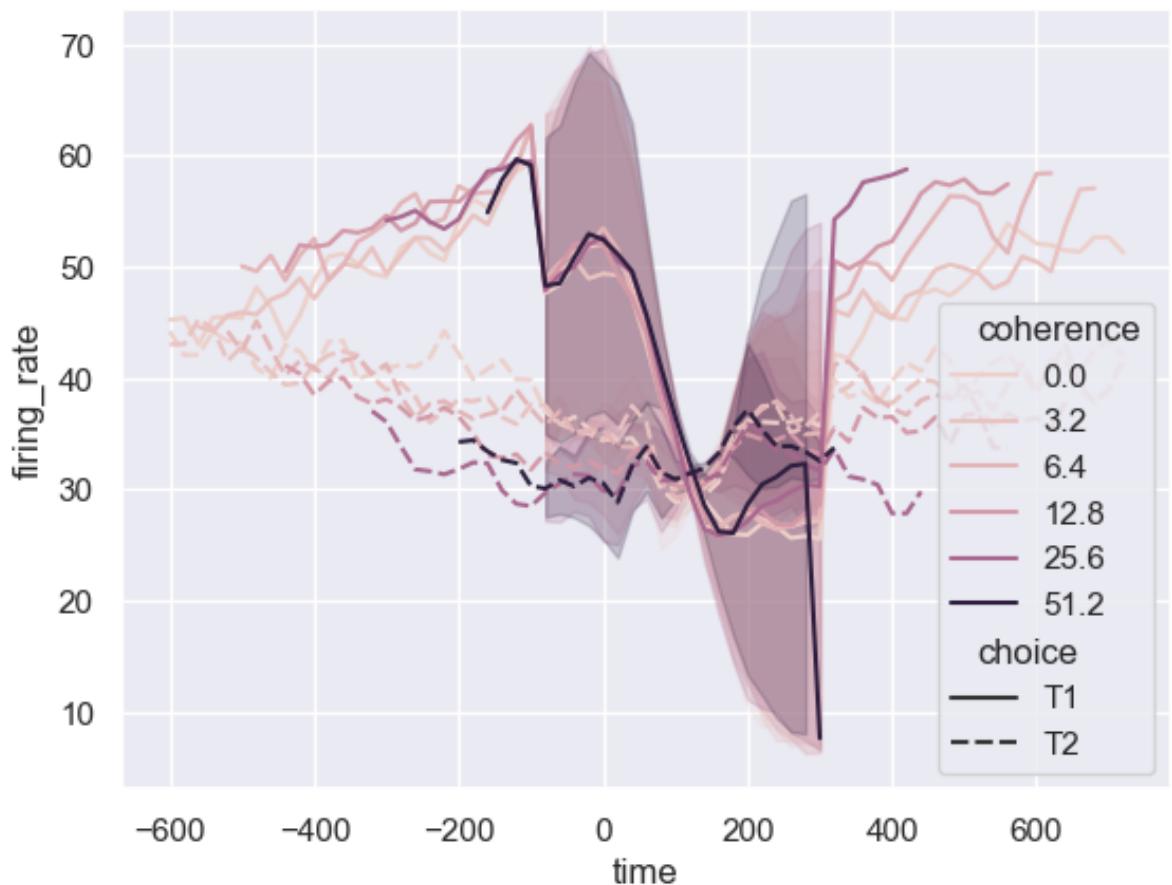


```
In [41]: dots.head()
```

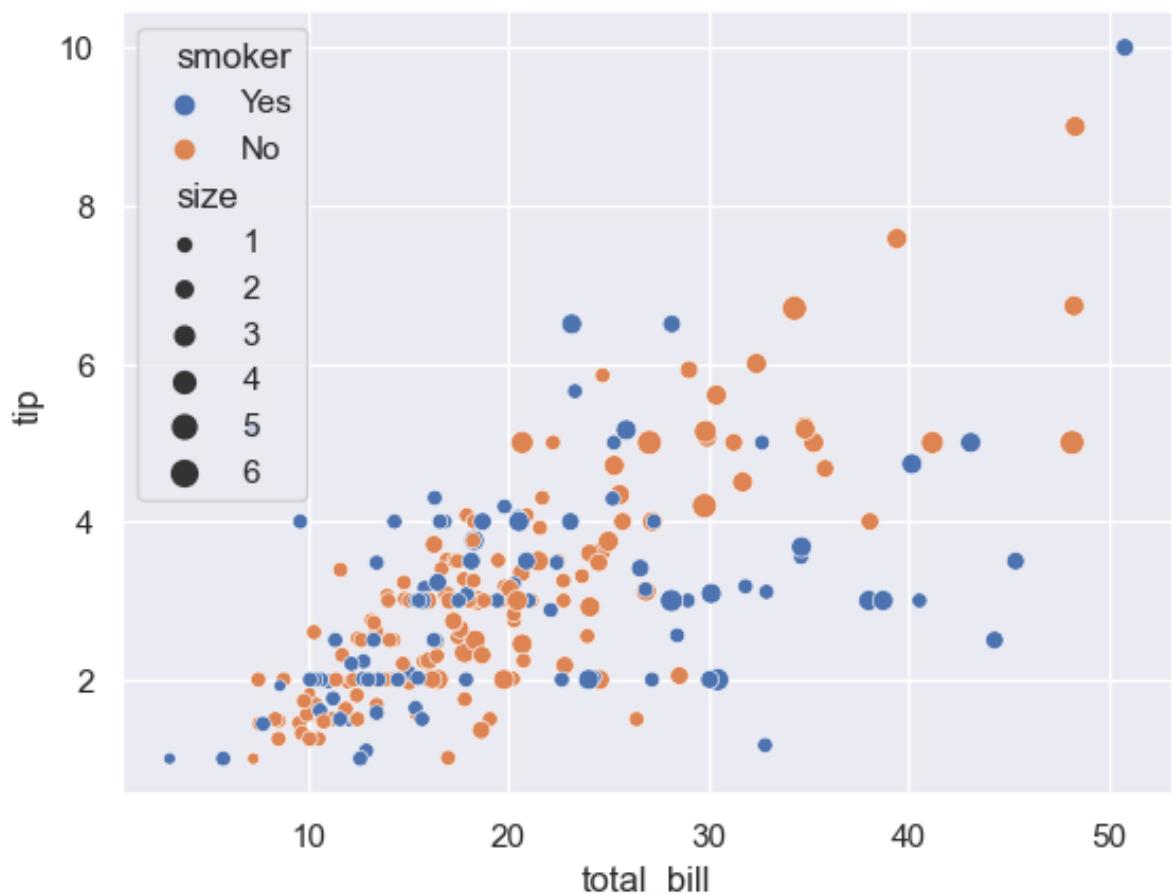
Out [41]:

	align	choice	time	coherence	firing_rate
0	dots	T1	-80	0.0	33.189967
1	dots	T1	-80	3.2	31.691726
2	dots	T1	-80	6.4	34.279840
3	dots	T1	-80	12.8	32.631874
4	dots	T1	-80	25.6	35.060487

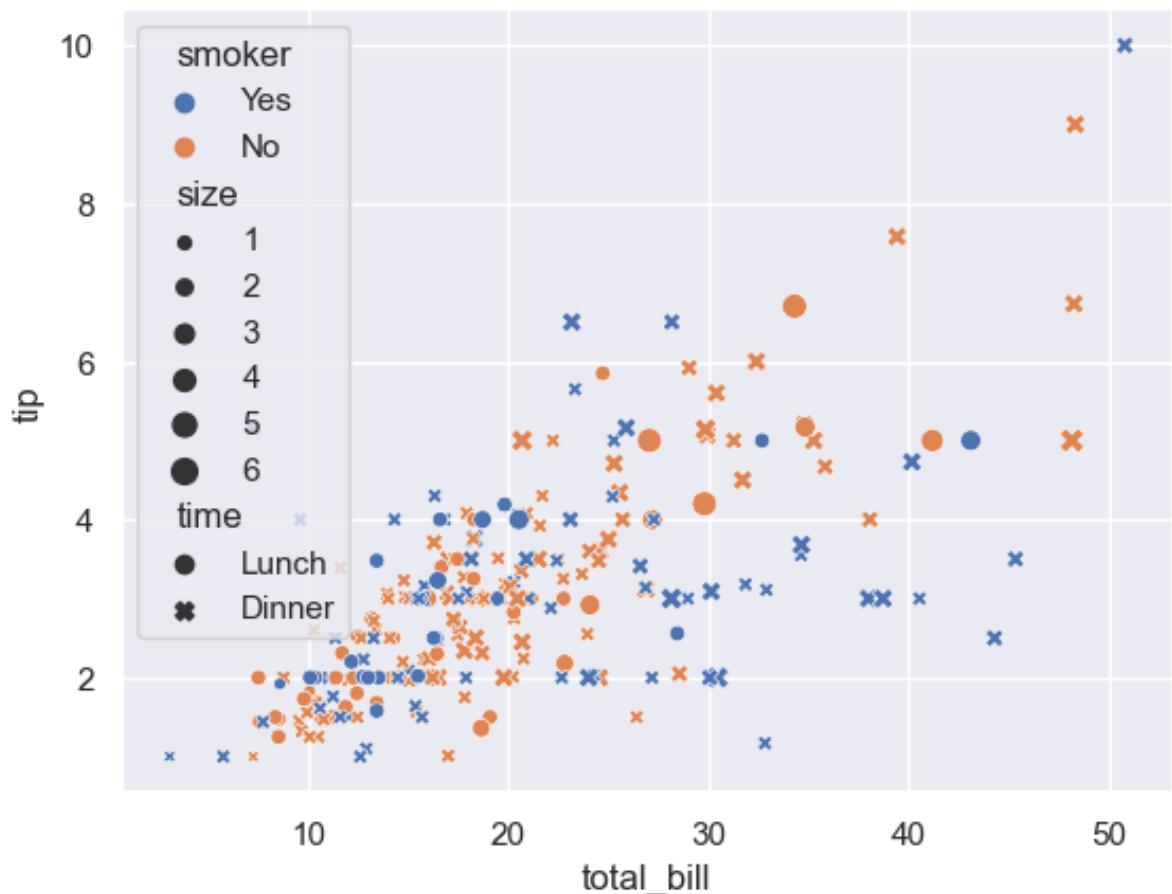
```
In [42]: sns.lineplot(x= 'time', y= 'firing_rate', data= dots, hue= 'coherence',
```



```
In [43]: sns.scatterplot(x= 'total_bill', y= 'tip', hue= 'smoker', size= 'size',
```



```
In [44]: sns.scatterplot(x= 'total_bill', y= 'tip', hue= 'smoker', size= 'si
```

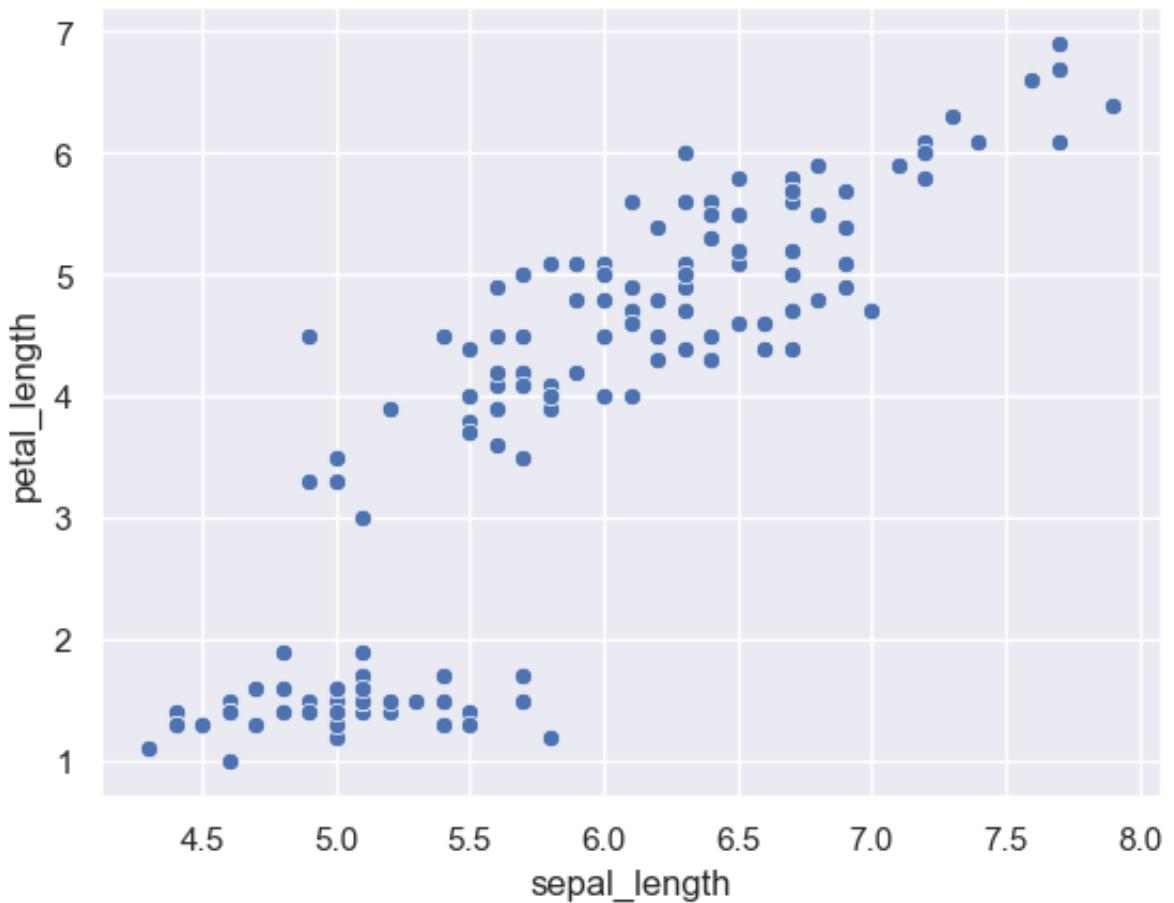


```
In [45]: iris= sns.load_dataset('iris')
iris.head()
```

Out [45]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

```
In [46]: sns.scatterplot(x= 'sepal_length', y= 'petal_length', data= iris);
```



2. Категориальные данные

Если одна из основных переменных является «категориальной» (разделённой на отдельные группы), может быть полезно использовать более специализированный подход к визуализации. Существует два способа построения таких графиков. Существует ряд функций уровня областей рисования (axes) для отображения категориальных данных различными способами, а также интерфейс `catplot()` уровня рисунков (figure), который обеспечивает унифицированный доступ к ним на более высоком уровне.

Рассмотрим различные типы категориальных графиков как принадлежащие к трём разным семействам, которые мы подробно обсудим ниже. Вот они:

Диаграммы категориального рассеяния:

1. `stripplot() (kind="strip";default)`
2. `swarmplot() (kind="swarm")`

Диаграммы категориального распределения:

1. `boxplot() (kind="box")`
2. `boxenplot() (kind="boxen")`
3. `violinplot() (kind="violin")`

Диаграммы категориальной оценки:

1. `pointplot() (kind="point")`
2. `barplot() (kind="bar")`
3. `countplot() (kind="count")`

In [47]: `sns.set(style= 'ticks', color_codes= True)`

Диаграмма категориального рассеяния

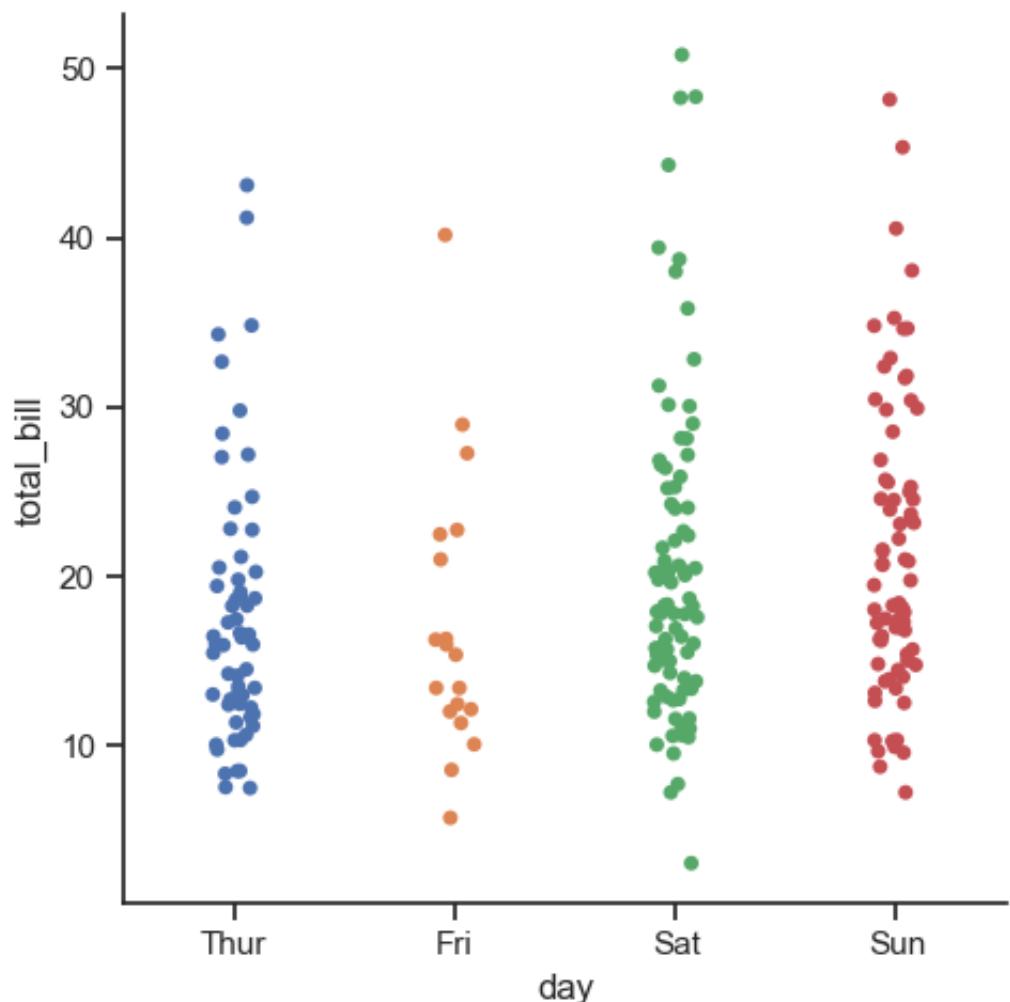
По умолчанию данные в `catplot()` представляются в виде диаграммы рассеяния.

In [48]: `tips.head()`

Out [48]:

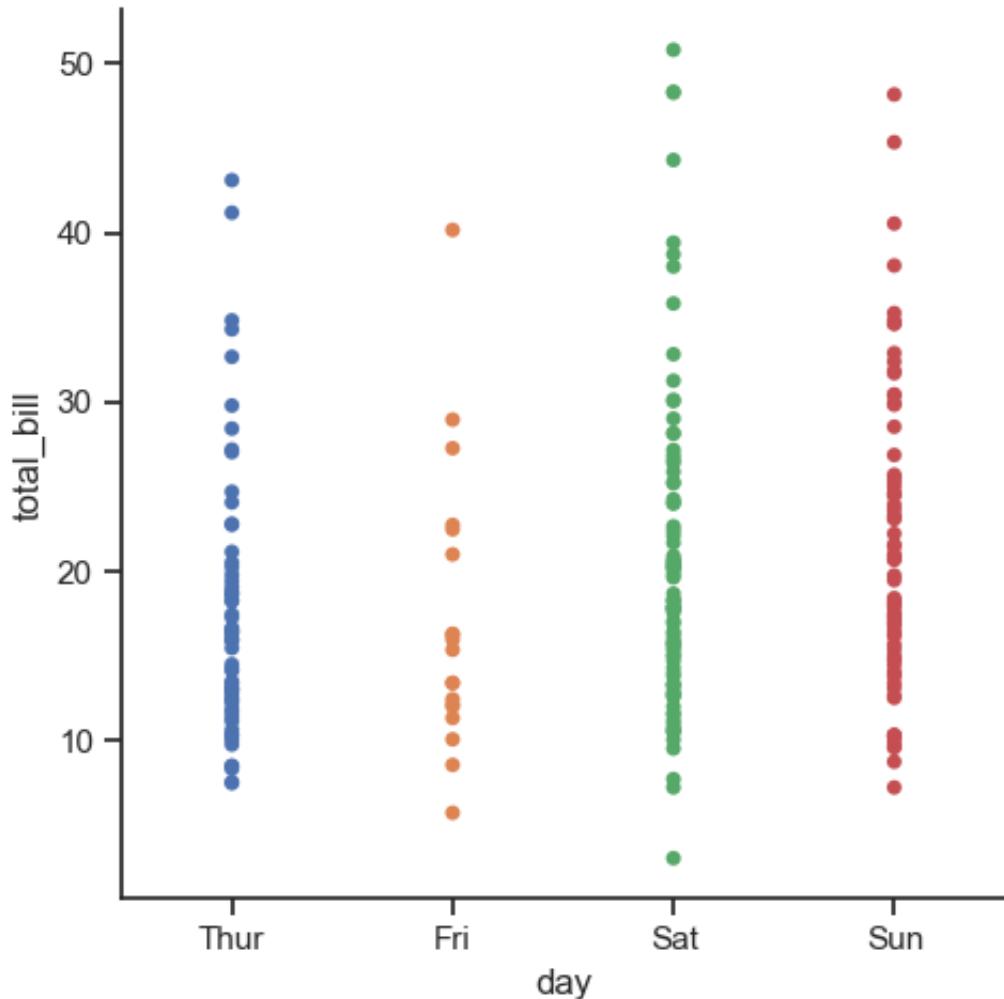
	total_bill	tip	sex	smoker	day	time	size	
0	16.99	1.01	Female		No	Sun	Dinner	2
1	10.34	1.66	Male		No	Sun	Dinner	3
2	21.01	3.50	Male		No	Sun	Dinner	3
3	23.68	3.31	Male		No	Sun	Dinner	2
4	24.59	3.61	Female		No	Sun	Dinner	4

In [49]: `sns.catplot(x= 'day', y= 'total_bill', data= tips, hue='day', jitte`



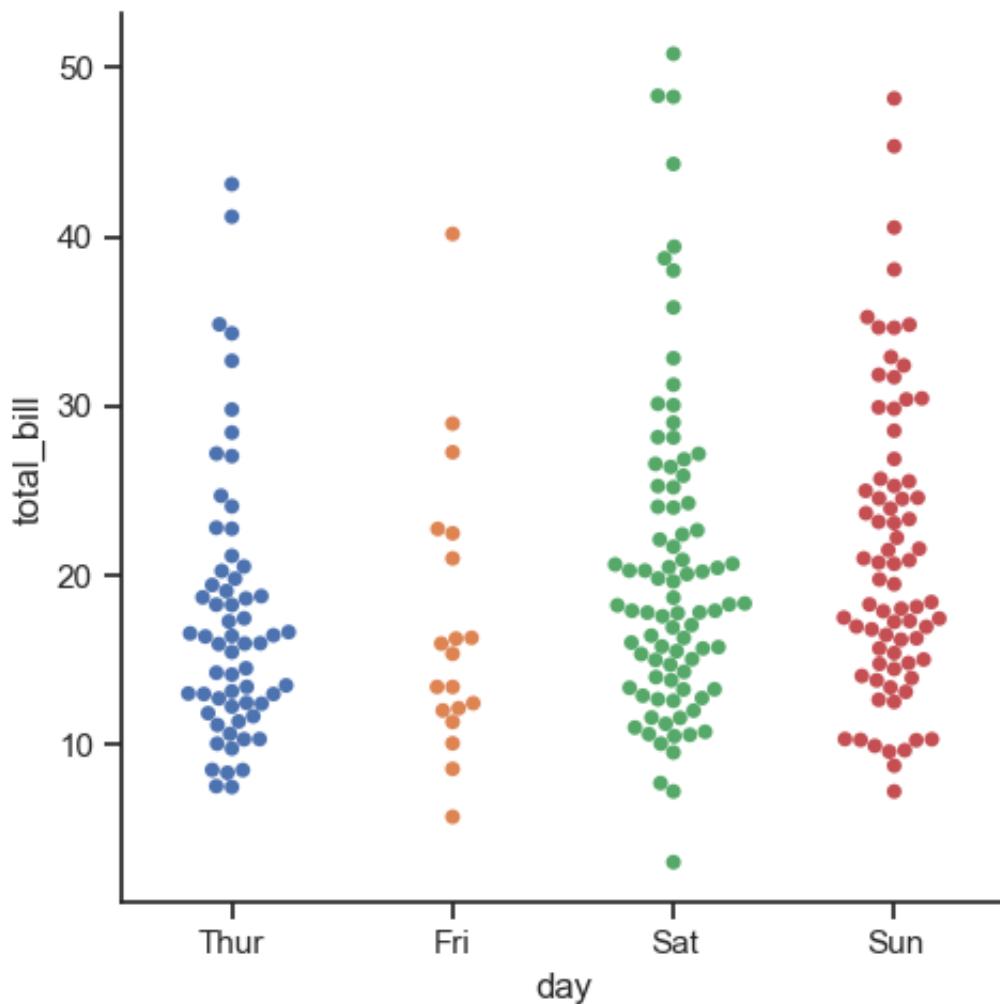
Параметр `jitter` управляет величиной "тряски" или отключает ее полностью:

```
In [50]: sns.catplot(x= 'day', y= 'total_bill', data= tips, hue='day', jitte
```



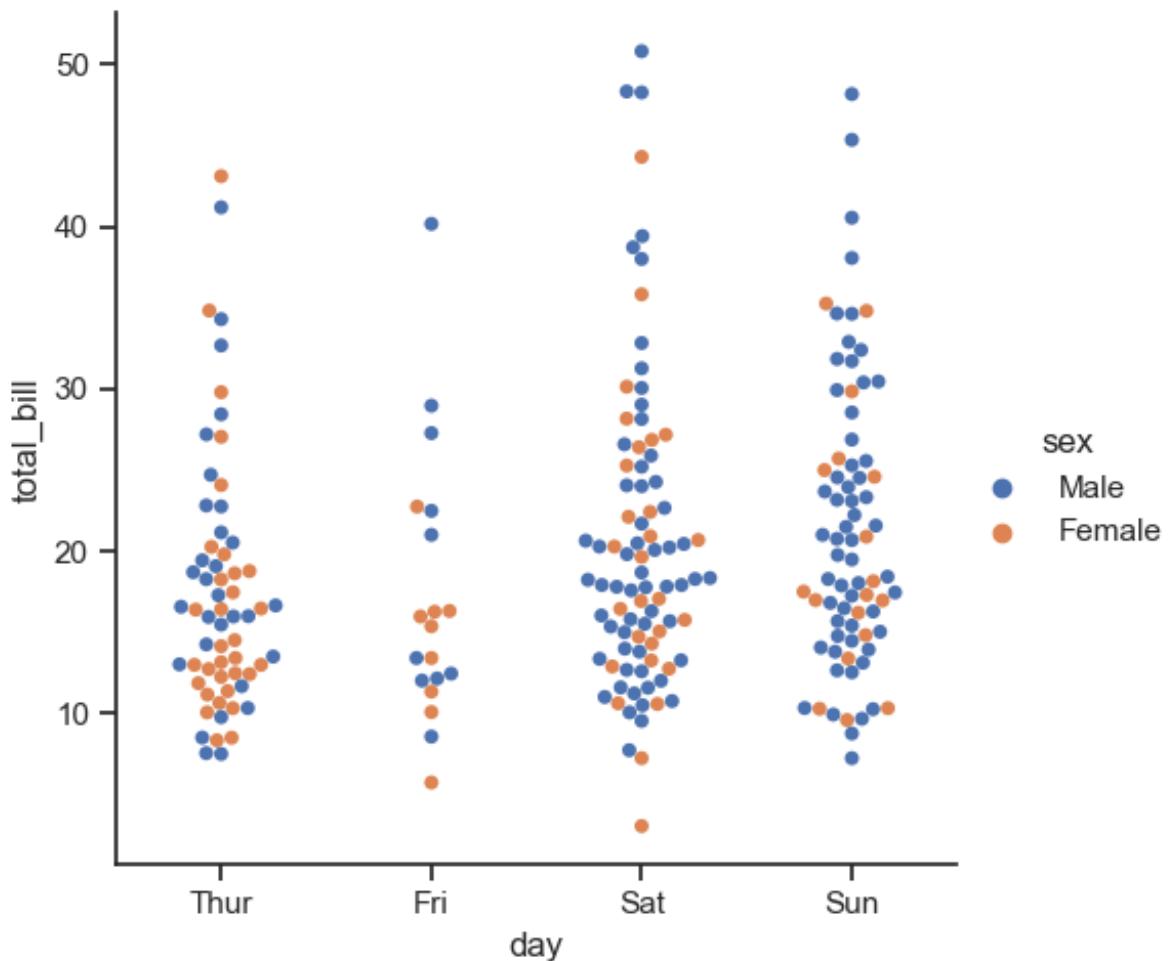
Второй подход корректирует точки вдоль категориальной оси с помощью алгоритма, предотвращающего их перекрытие. Это может дать лучшее представление о распределении наблюдений, хотя хорошо работает только для относительно небольших наборов данных. Такой тип графика иногда называют «пчелиным роем» (`beeswarm`), и он строится в Seaborn с помощью функции `swarmplot()`, которая активируется установкой `kind="swarm"` в `catplot()`:

```
In [51]: sns.catplot(x= 'day', y= 'total_bill', data= tips, hue='day', kind=
```



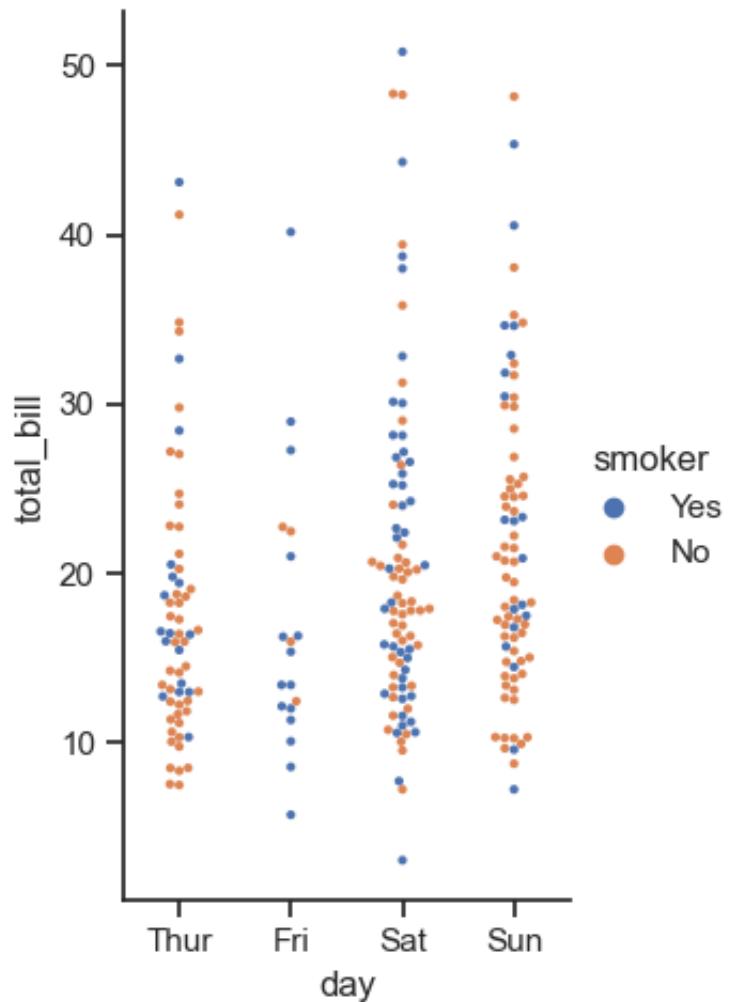
Подобно реляционным графикам, к категориальному графику можно добавить ещё одно измерение, используя семантику оттенка (hue). Категориальные графики не поддерживают семантику размера и стиля. Каждая функция построения категориального графика обрабатывает семантику оттенка по-разному. Для диаграмм рассеяния достаточно изменить только цвет точек:

```
In [52]: sns.catplot(x= 'day', y= 'total_bill', kind= 'swarm', hue= 'sex', d
```

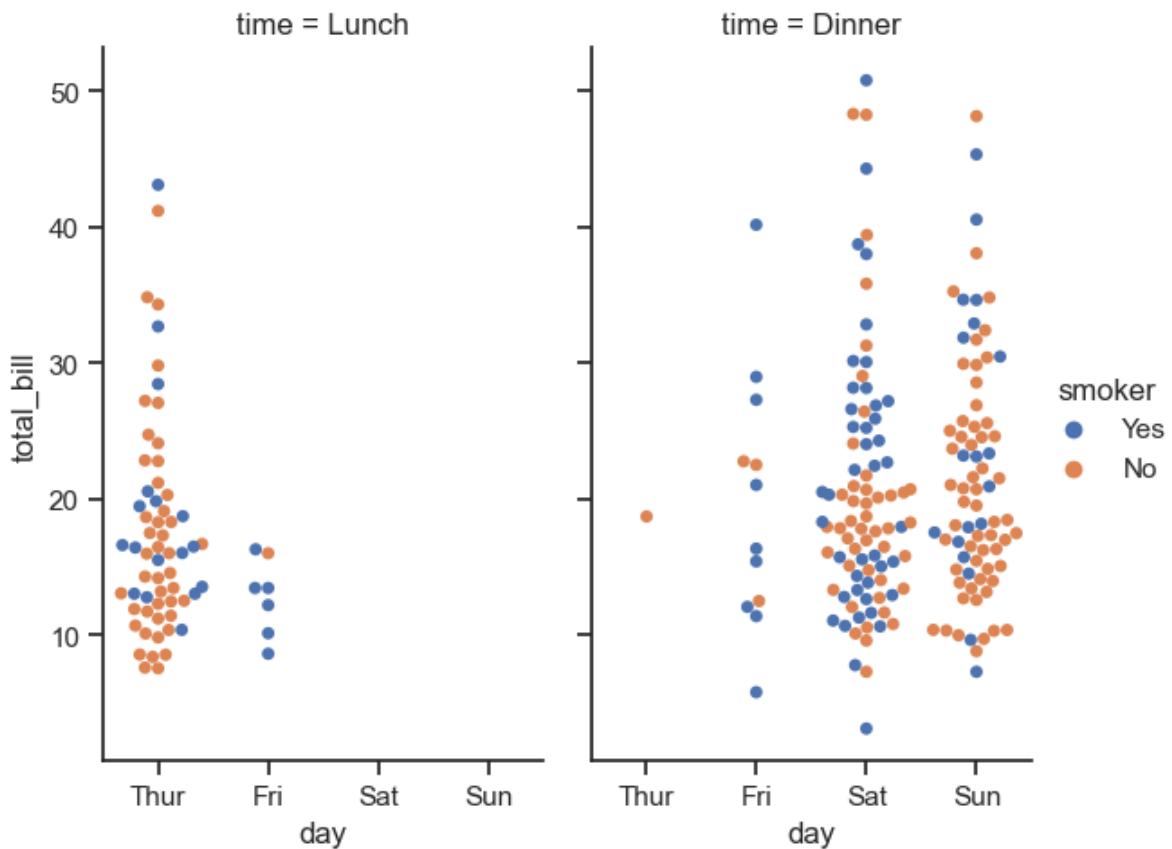


Отображение нескольких взаимосвязей с помощью областей рисования. Как и `relplot()`, функция `catplot()` построена на `FacetGrid`, что позволяет легко добавлять переменные областей рисования для визуализации многомерных взаимосвязей:

```
In [53]: sns.catplot(x="day", y="total_bill", hue="smoker",
                     aspect=.6, kind="swarm", size=3, data=tips);
```

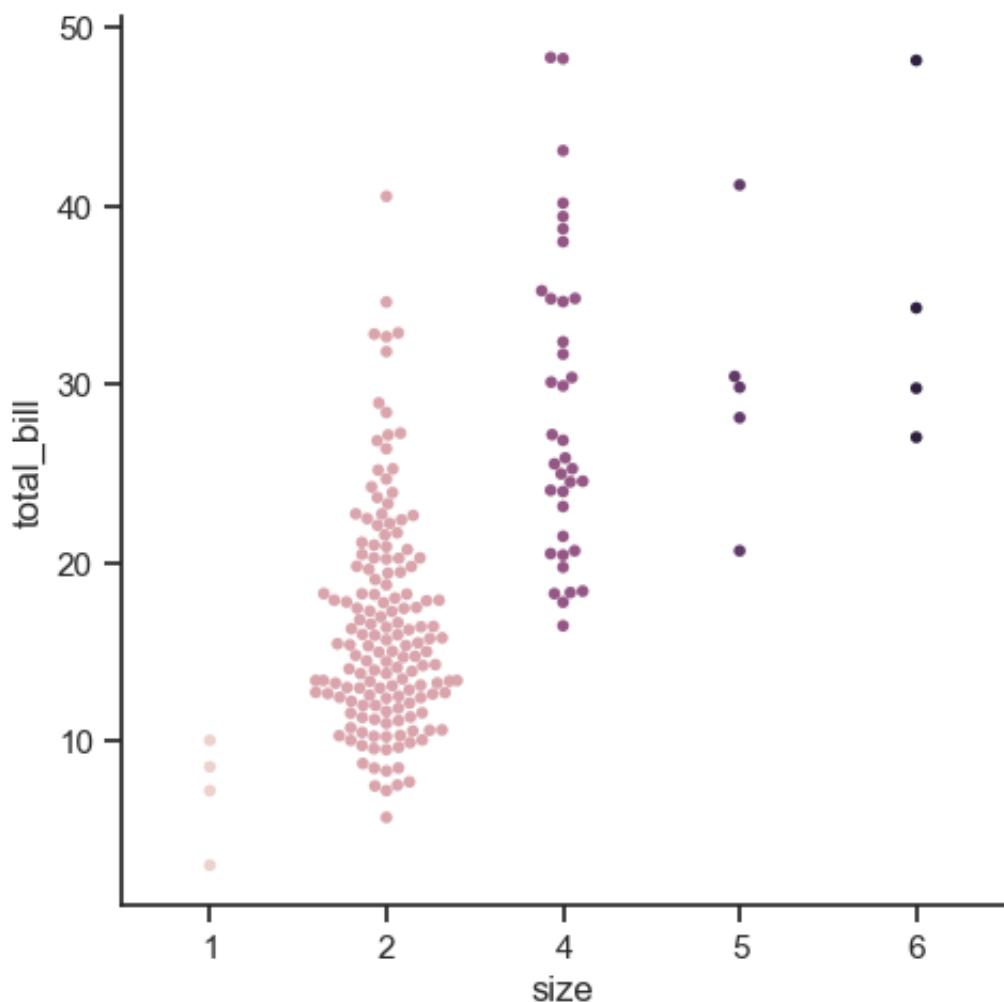


```
In [54]: sns.catplot(x="day", y="total_bill", hue="smoker",
                    col="time", aspect=.6,
                    kind="swarm", data=tips);
```



В отличие от числовых данных, не всегда очевидно, как упорядочить уровни категориальной переменной вдоль её оси. Как правило, функции построения категориальных графиков Seaborn пытаются определить порядок категорий на основе данных. Если ваши данные относятся к категориальному типу данных Pandas, то порядок категорий по умолчанию можно задать в нём. Если переменная, переданная на категориальную ось, выглядит числовой, уровни будут отсортированы. Однако данные по-прежнему рассматриваются как категориальные и отображаются в порядковых позициях на категориальных осях (в частности, в точках 0, 1, ...), даже если для их обозначения используются числа:

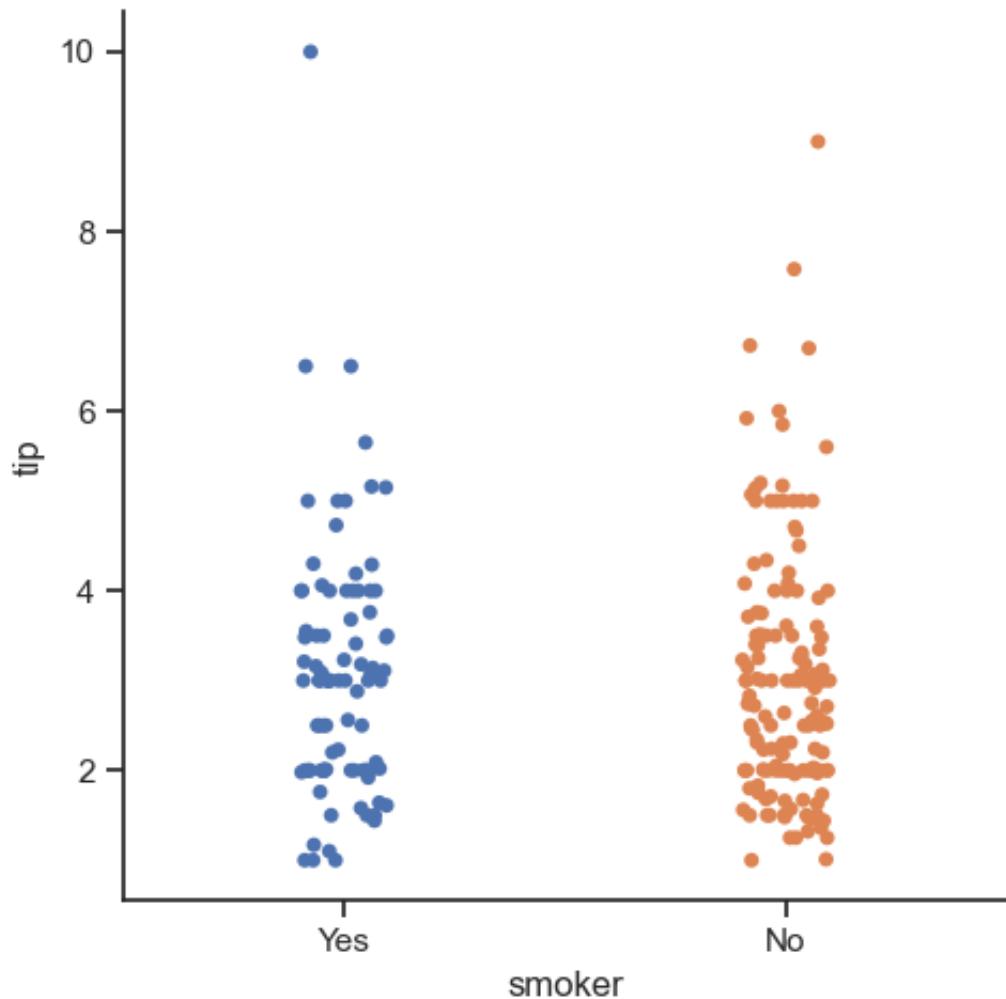
```
In [55]: sns.catplot(x= 'size', y= 'total_bill', kind= 'swarm', hue='size',
```



Другой вариант выбора порядка по умолчанию — использовать уровни категории в том виде, в котором они представлены в наборе данных. Порядок также можно контролировать для каждого участка с помощью параметра `order`.

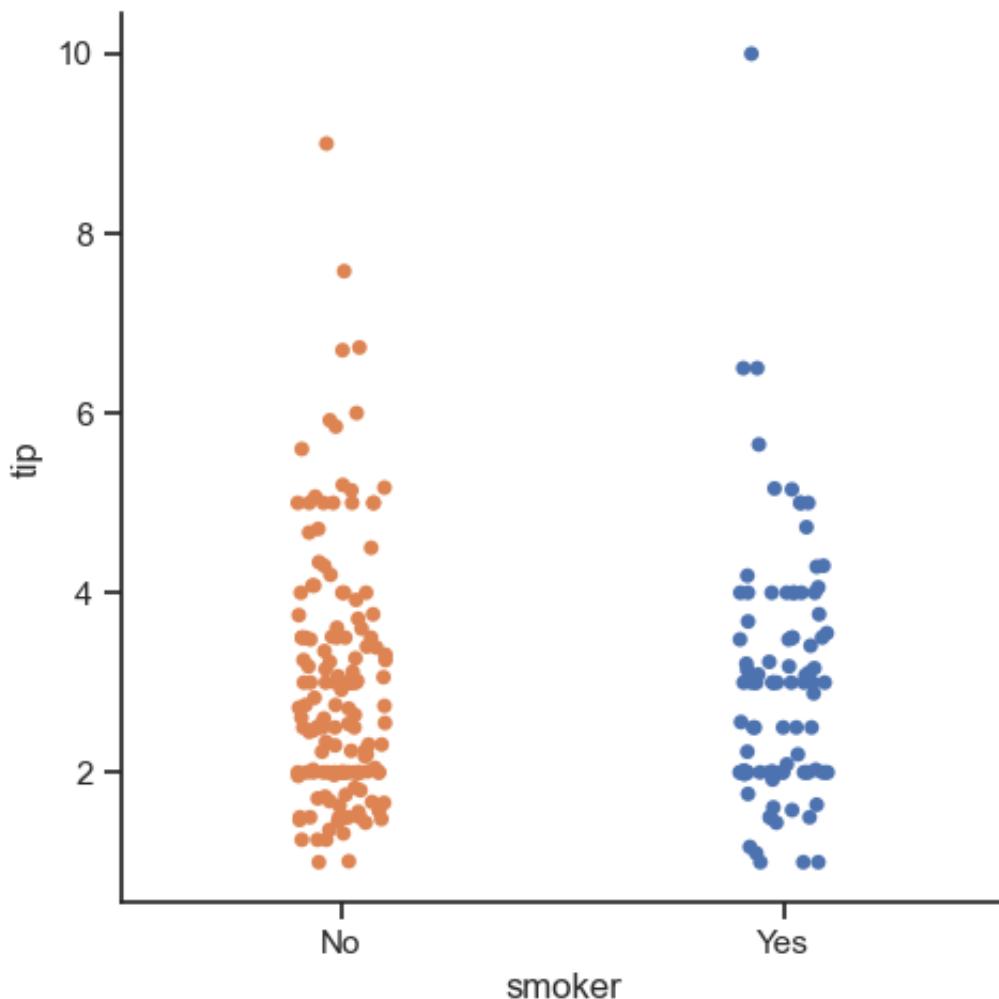
1. без параметра `order`

```
In [56]: sns.catplot( x= 'smoker', y= 'tip', hue='smoker', data= tips);
```



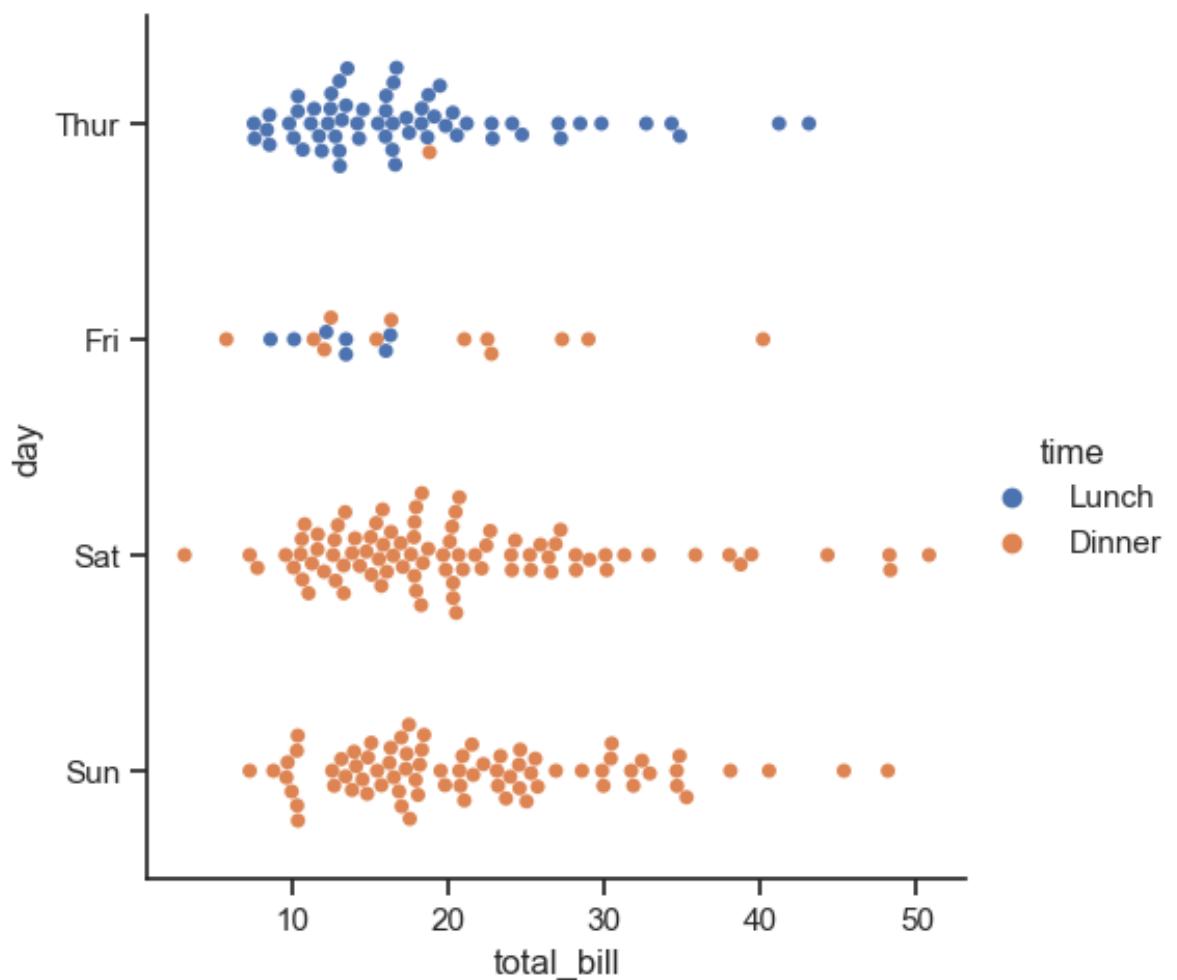
```
2. с параметром order
```

```
In [57]: sns.catplot( x= 'smoker', y= 'tip', order= ['No', 'Yes'] , hue='smo
```



Мы упомянули понятие «категориальной оси». В этих примерах она всегда соответствует горизонтальной оси. Но часто бывает полезно разместить категориальную переменную на вертикальной оси (особенно когда названия категорий относительно длинные или их много).

```
In [58]: sns.catplot(x="total_bill", y="day", hue="time", kind="swarm", data=
```



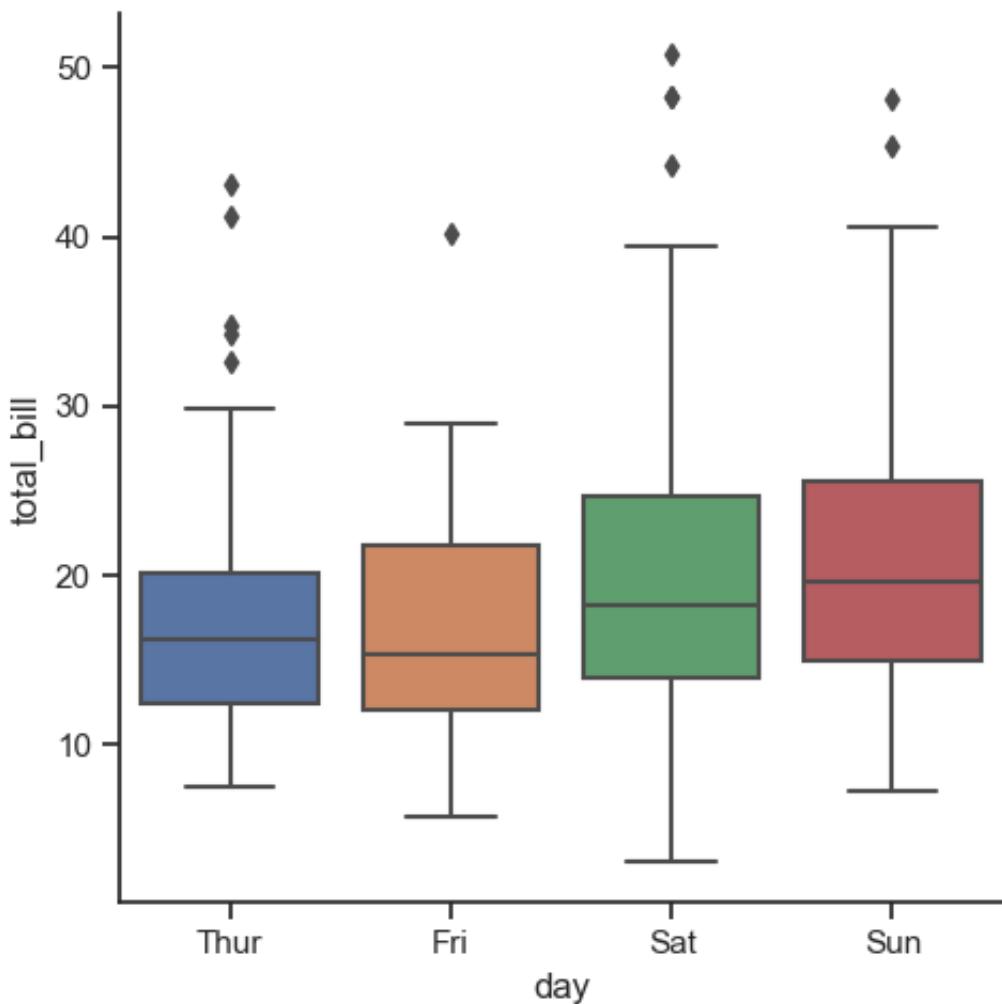
Распределение наблюдений внутри категорий

По мере роста размера набора данных категориальные диаграммы рассеяния становятся ограниченны в плане информации о распределении значений внутри каждой категории. В таких случаях существует несколько подходов к обобщению информации о распределении, упрощающих сравнение данных по уровням категорий.

Диаграммы размаха ("ящик с усами", box plot)

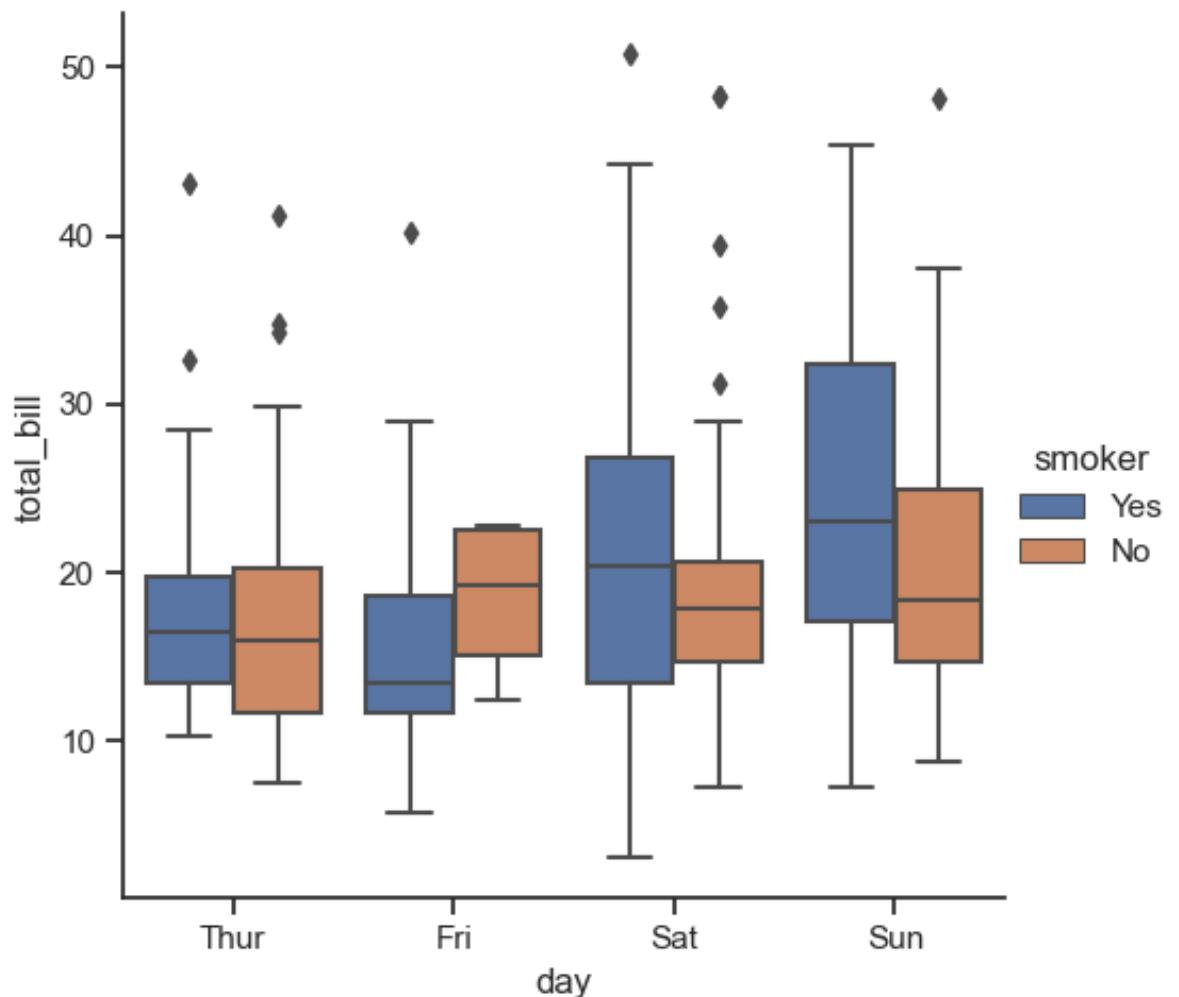
Этот тип диаграммы отображает три квартильных значения распределения вместе с экстремальными значениями. «Усы» доходят до точек, лежащих в пределах 1,5 межквартильных интервалов (IQR) от нижнего и верхнего квартилей, а затем наблюдения, выходящие за эти пределы, отображаются отдельно. Это означает, что каждое значение на диаграмме ящиков соответствует фактическому наблюдению в данных.

```
In [59]: sns.catplot(x= 'day', y= 'total_bill', kind= 'box', data= tips);
```



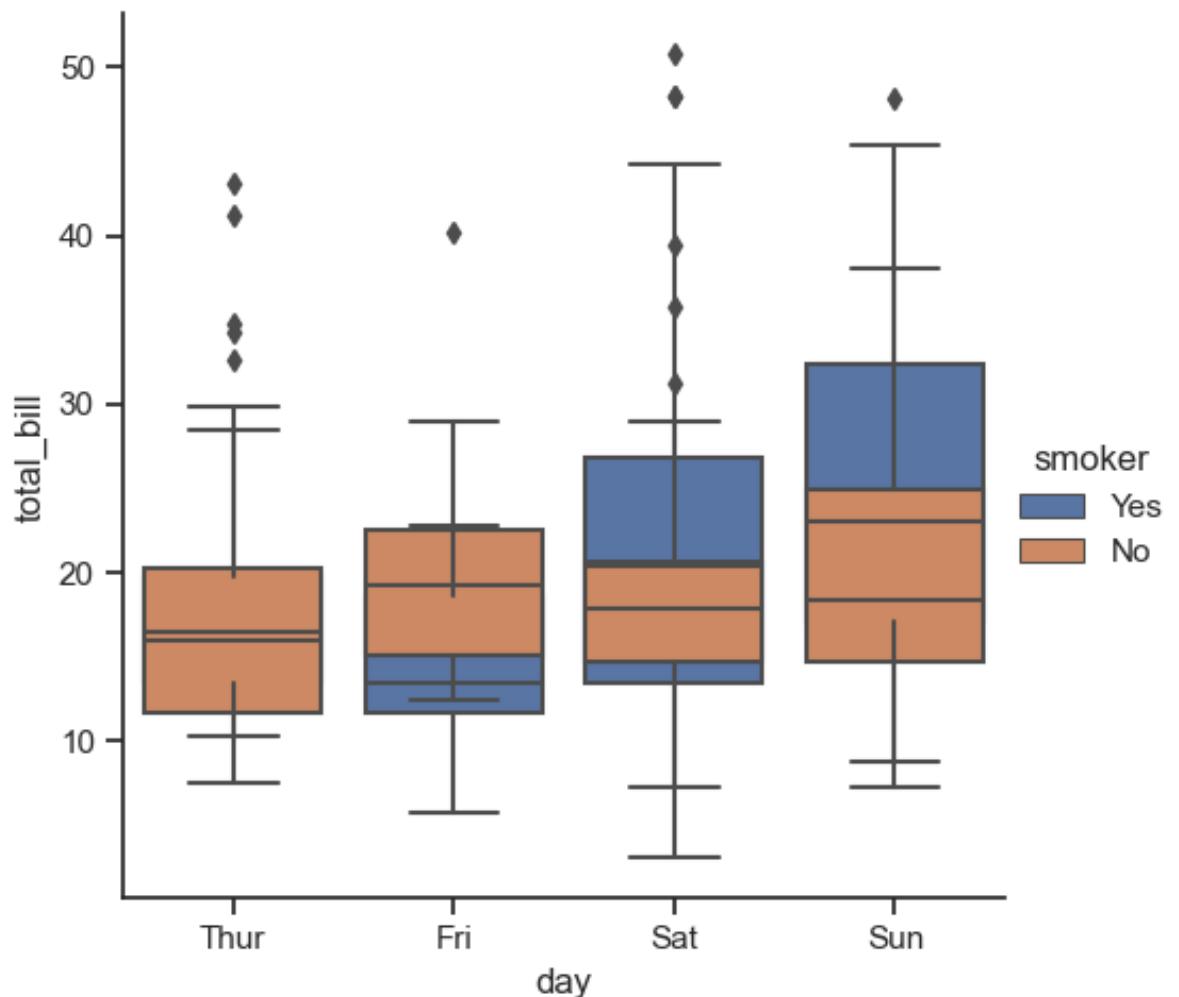
При добавлении семантики оттенка (hue) поле для каждого уровня семантической переменной перемещается вдоль категориальной оси, чтобы они не перекрывались:

```
In [60]: sns.catplot(x= 'day', y= 'total_bill', kind= 'box', data= tips, hue=
```



Такое поведение называется «уклонением» (“dodging”) и включено по умолчанию, поскольку предполагается, что семантическая переменная вложена в основную категориальную переменную. Если это не так, уклонение можно отключить:

```
In [61]: sns.catplot(x= 'day', y= 'total_bill', kind= 'box', data= tips, hue=
```



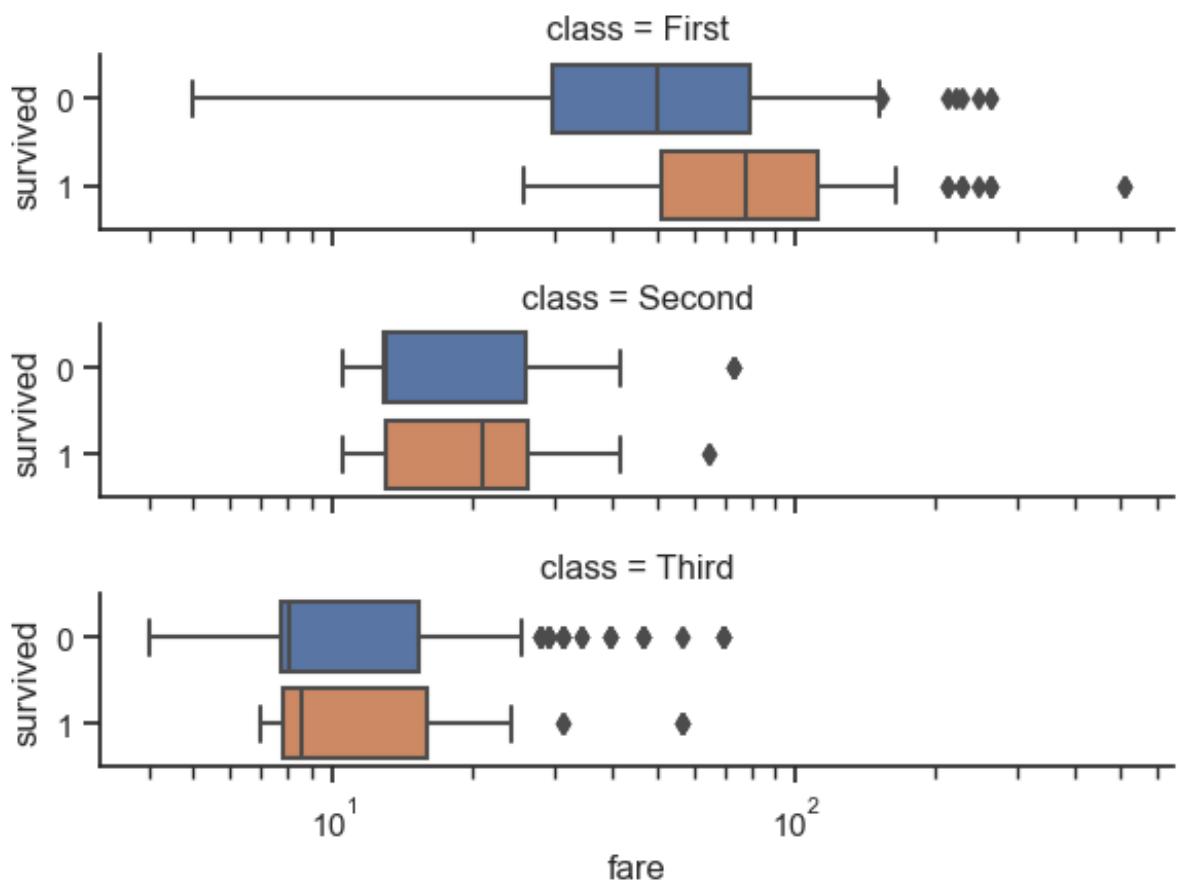
```
In [62]: titanic= sns.load_dataset('titanic')
```

```
In [63]: titanic.head()
```

Out [63]:

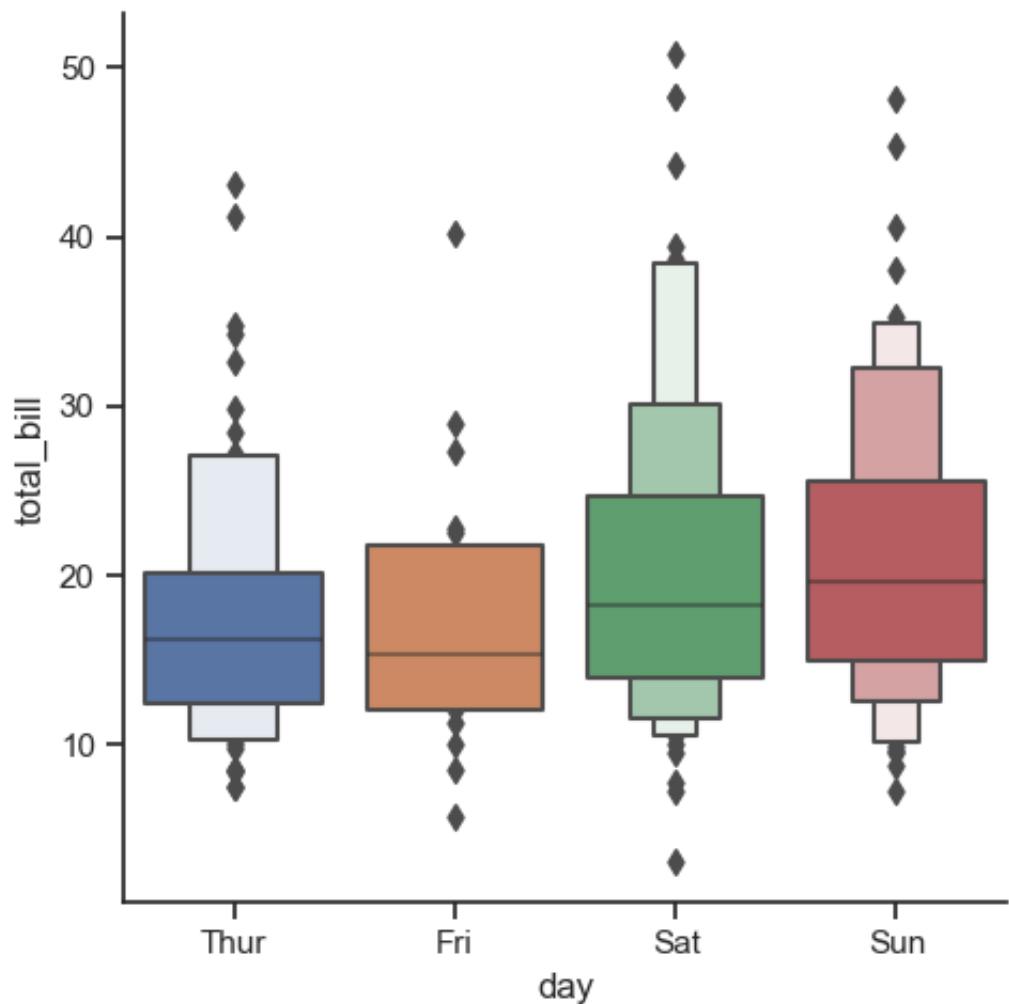
	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_m
0	0	3	male	22.0	1	0	7.2500	S	Third	man	T
1	1	1	female	38.0	1	0	71.2833	C	First	woman	F&
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	F&
3	1	1	female	35.0	1	0	53.1000	S	First	woman	F&
4	0	3	male	35.0	0	0	8.0500	S	Third	man	T

```
In [64]: g = sns.catplot(x="fare", y="survived", row="class",
                      kind="box", orient="h", height=1.5, aspect=4,
                      data=titanic.query("fare > 0"))
g.set(xscale="log");
```

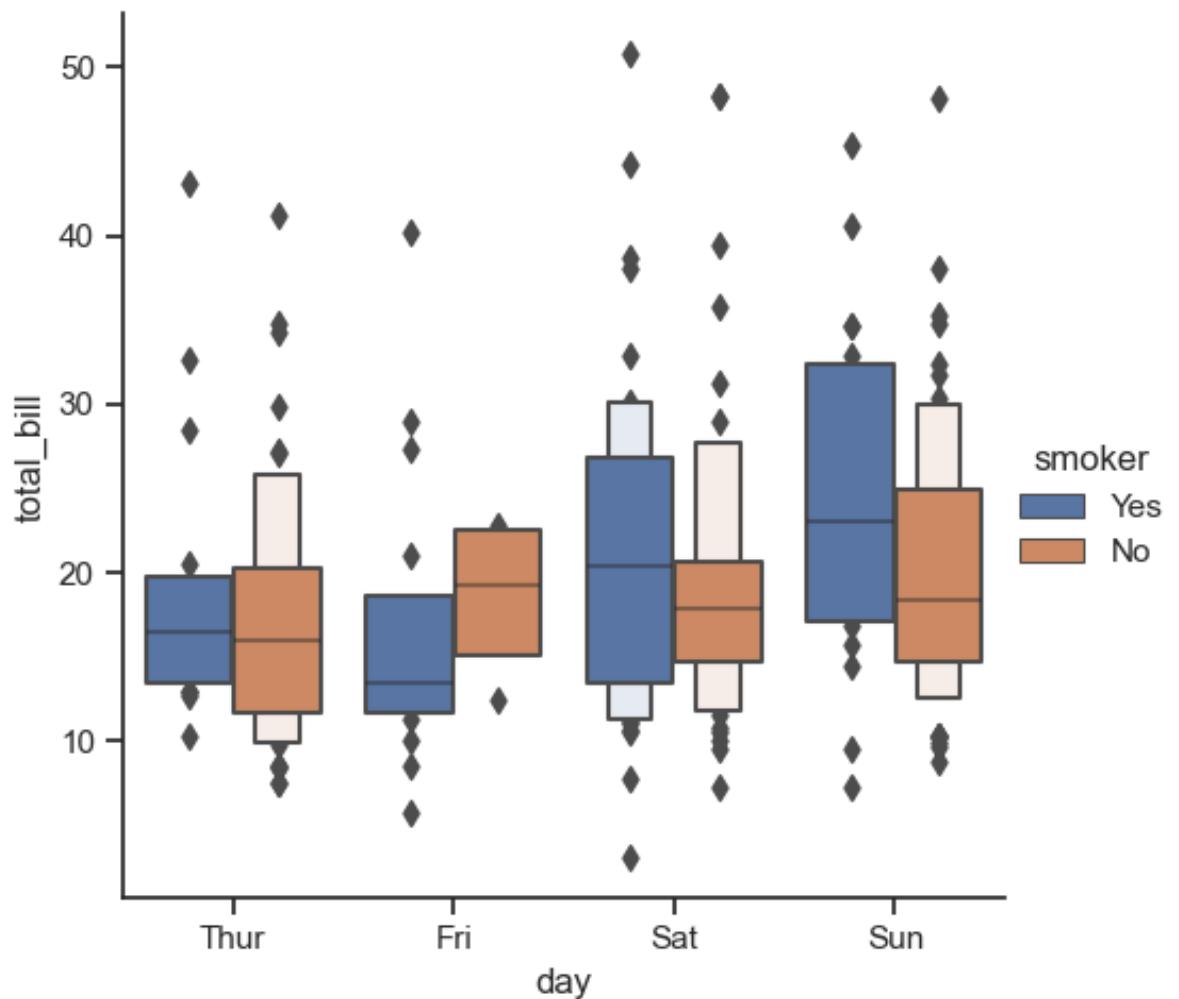


Связанная функция `boxenplot()` строит диаграмму, похожую на диаграмму размаха, но оптимизированную для отображения более подробной информации о форме распределения. Она лучше всего подходит для больших наборов данных:

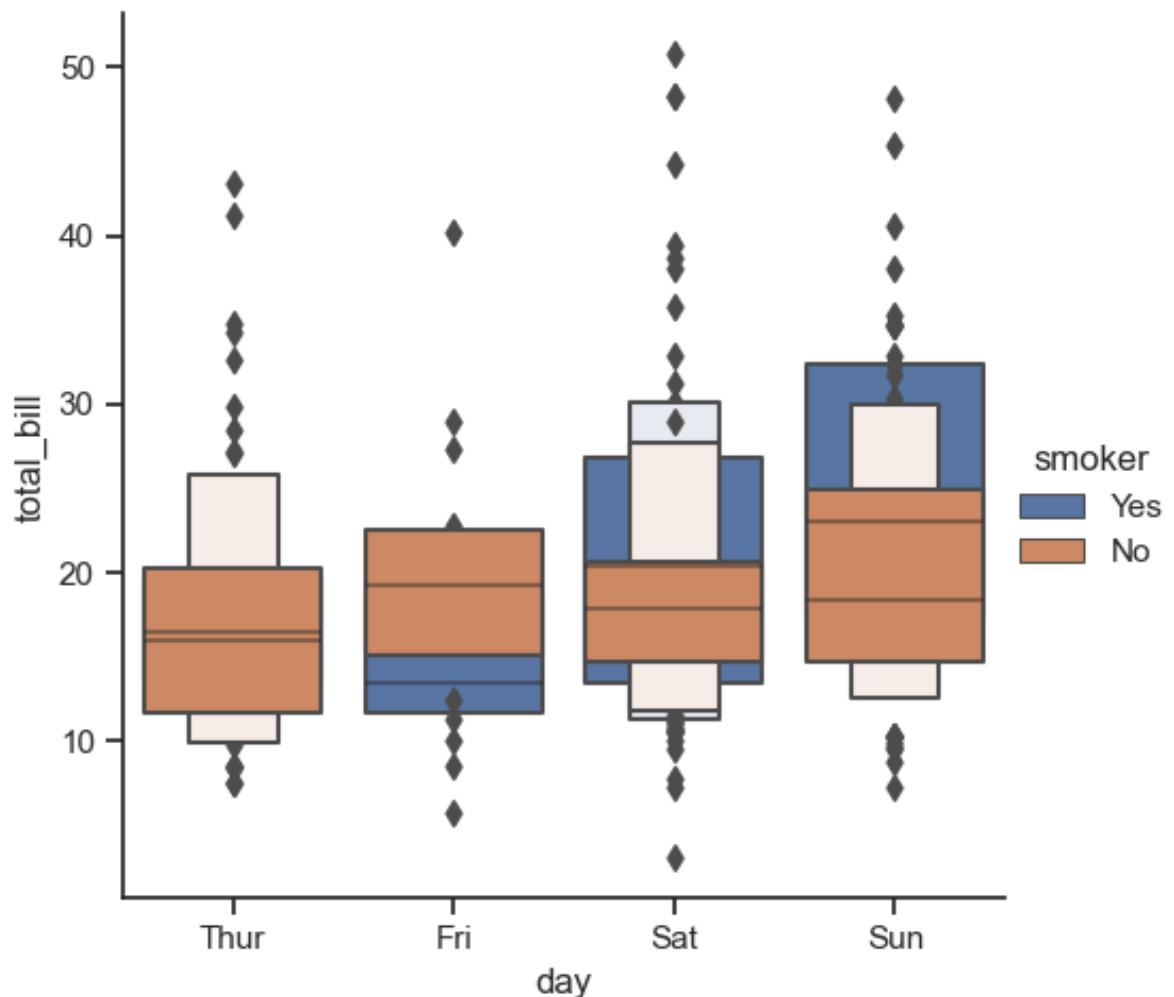
```
In [65]: sns.catplot(x= 'day', y= 'total_bill', kind= 'boxen', data= tips);
```



```
In [66]: sns.catplot(x= 'day', y= 'total_bill', kind= 'boxen', data= tips, h
```



```
In [67]: sns.catplot(x= 'day', y= 'total_bill', kind= 'boxen', data= tips, h
```



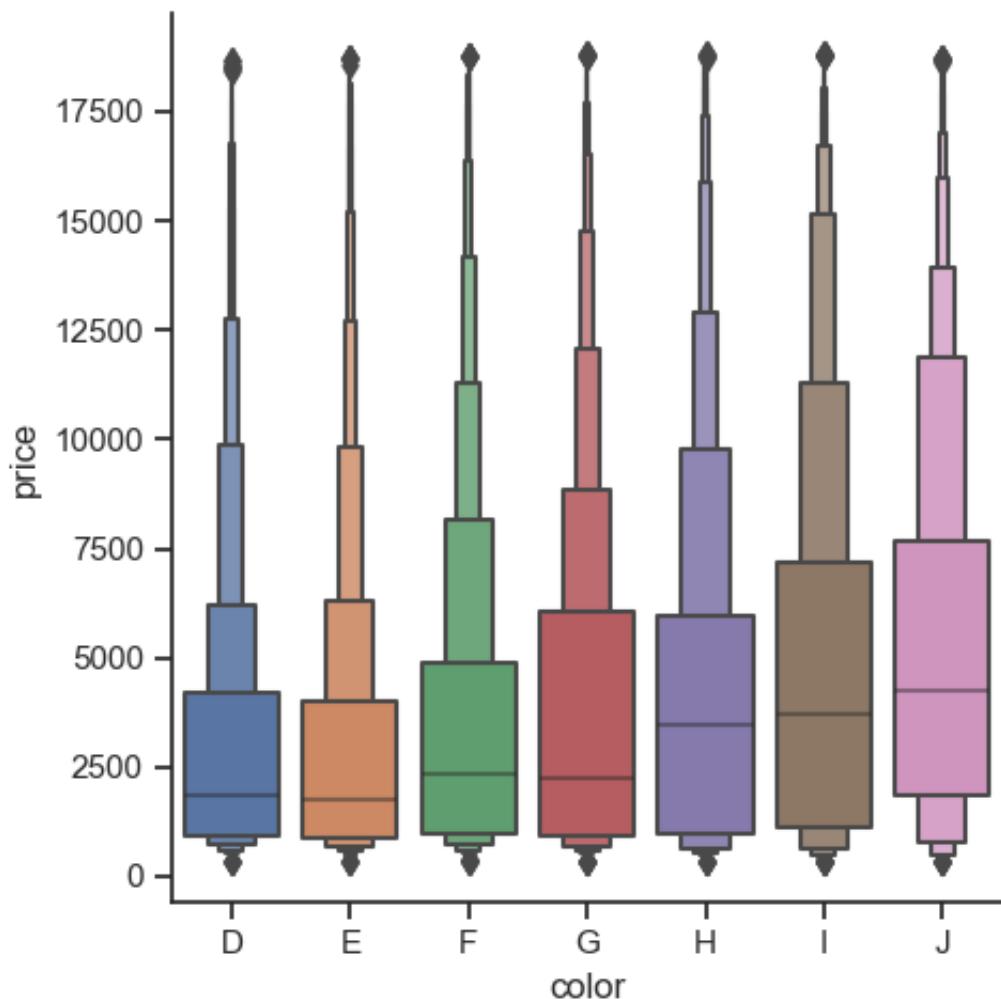
Лучше всего подходит для больших наборов данных

```
In [68]: diamonds= sns.load_dataset('diamonds')
diamonds.head()
```

Out [68]:

	carat	cut	color	clarity	depth	table	price	x	y	z
0	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
1	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
2	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
3	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
4	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75

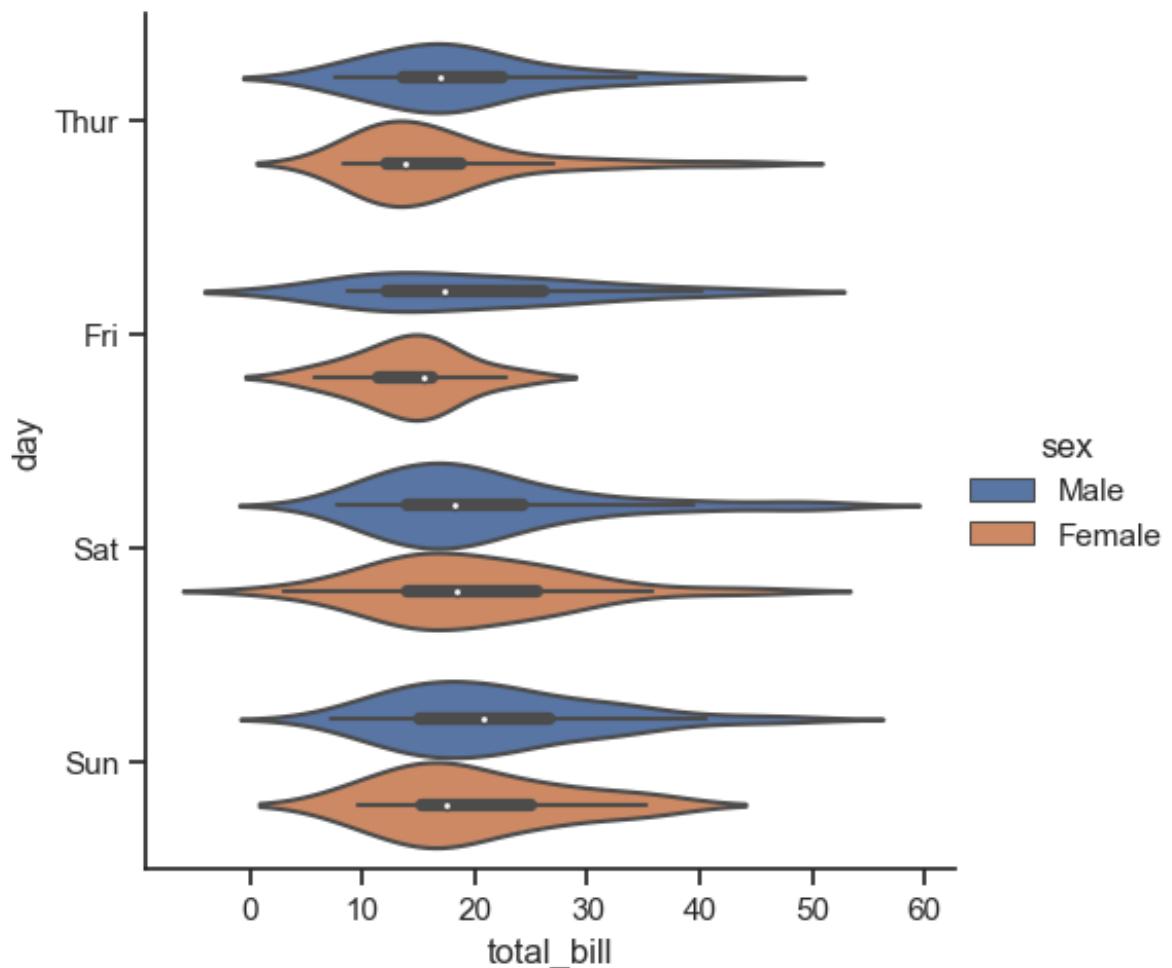
```
In [69]: sns.catplot(x= 'color', y= 'price', kind= 'boxen', data= diamonds.s
```



Скрипичные диаграммы (Violinplots)

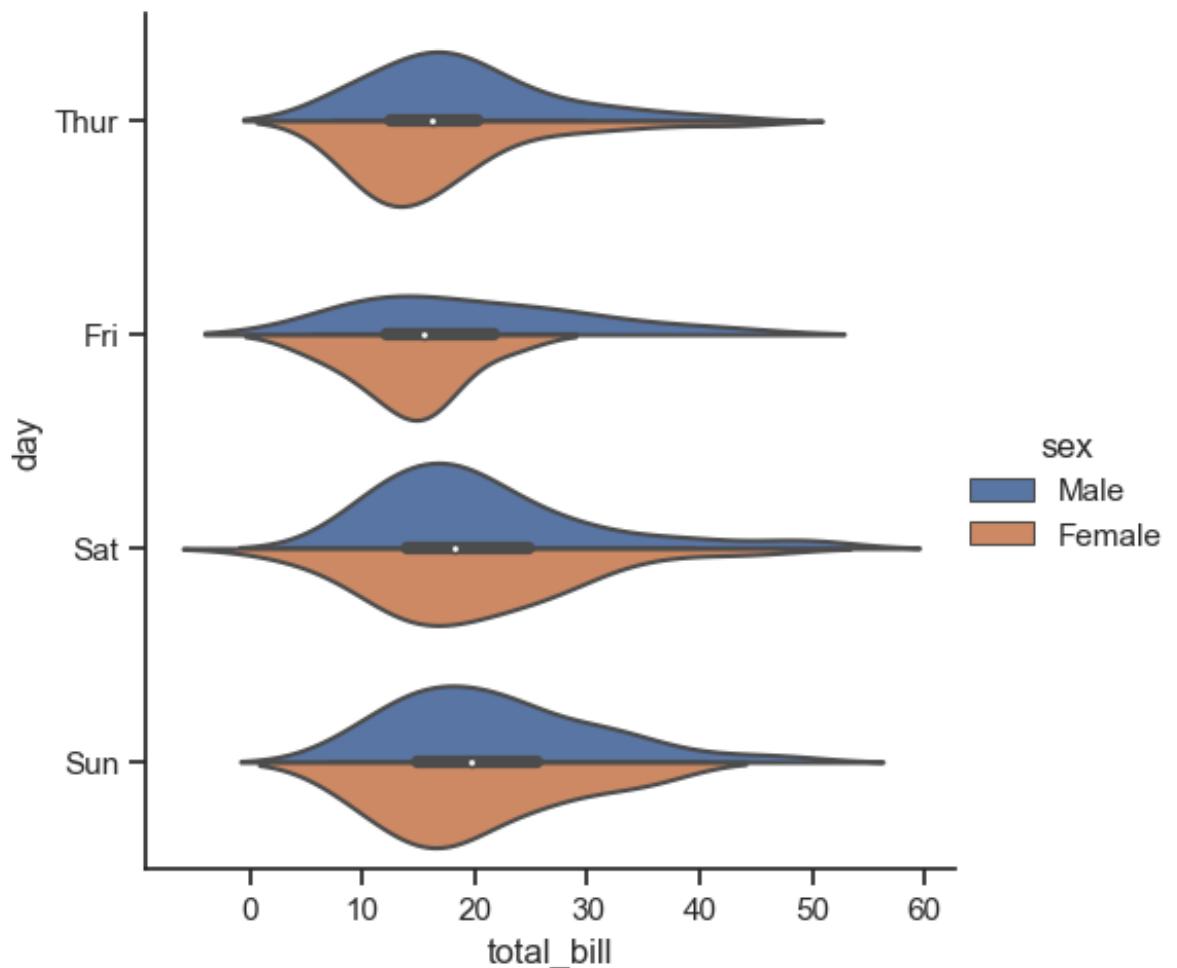
Другой подход — это функция `violinplot()`, которая сочетает в себе диаграмму размаха с процедурой оценки плотности распределения:

```
In [70]: sns.catplot(x="total_bill", y="day", hue="sex",
                    kind="violin", data=tips);
```

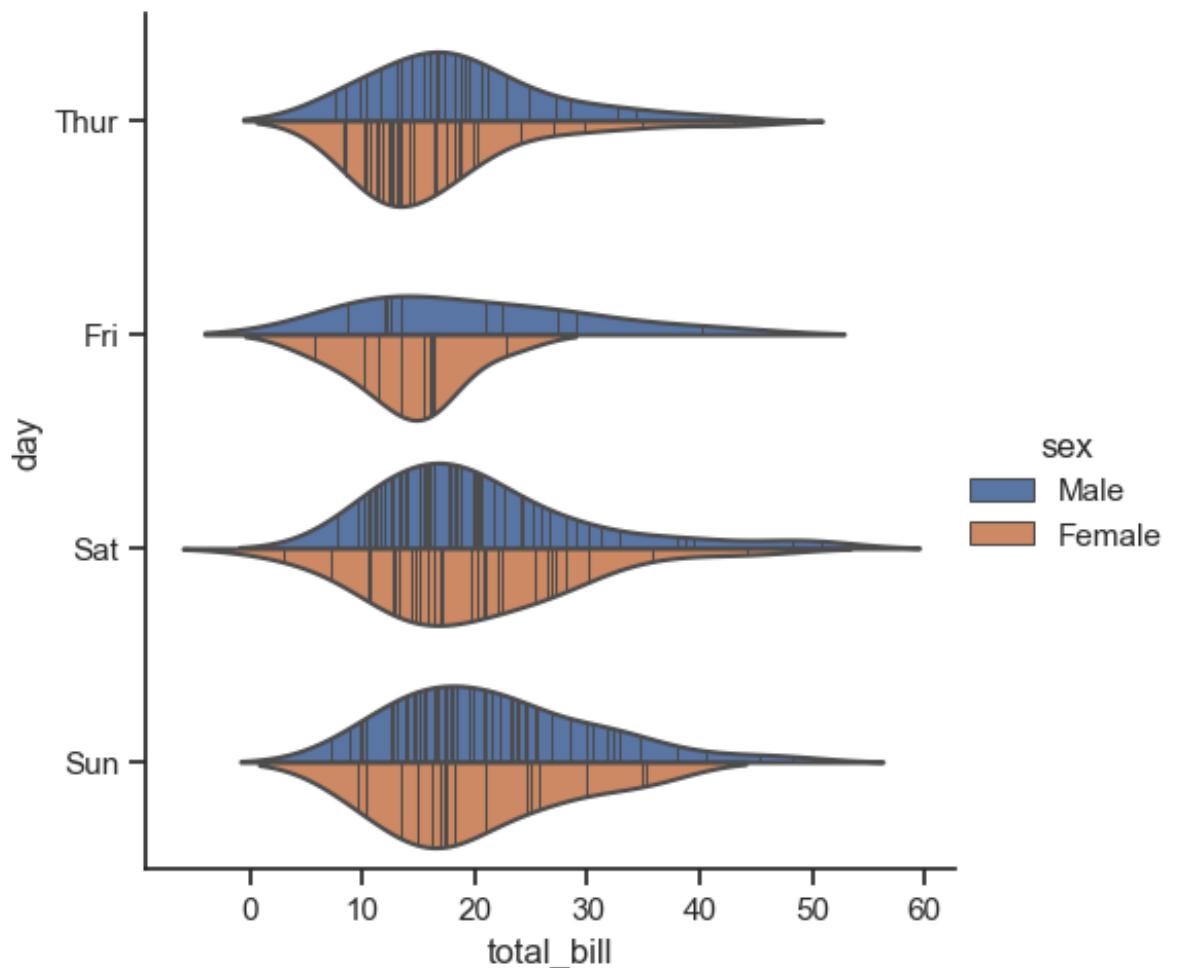


Также можно «разделить» скрипки (violins), если параметр оттенка имеет только два уровня, что может позволить более эффективно использовать пространство:

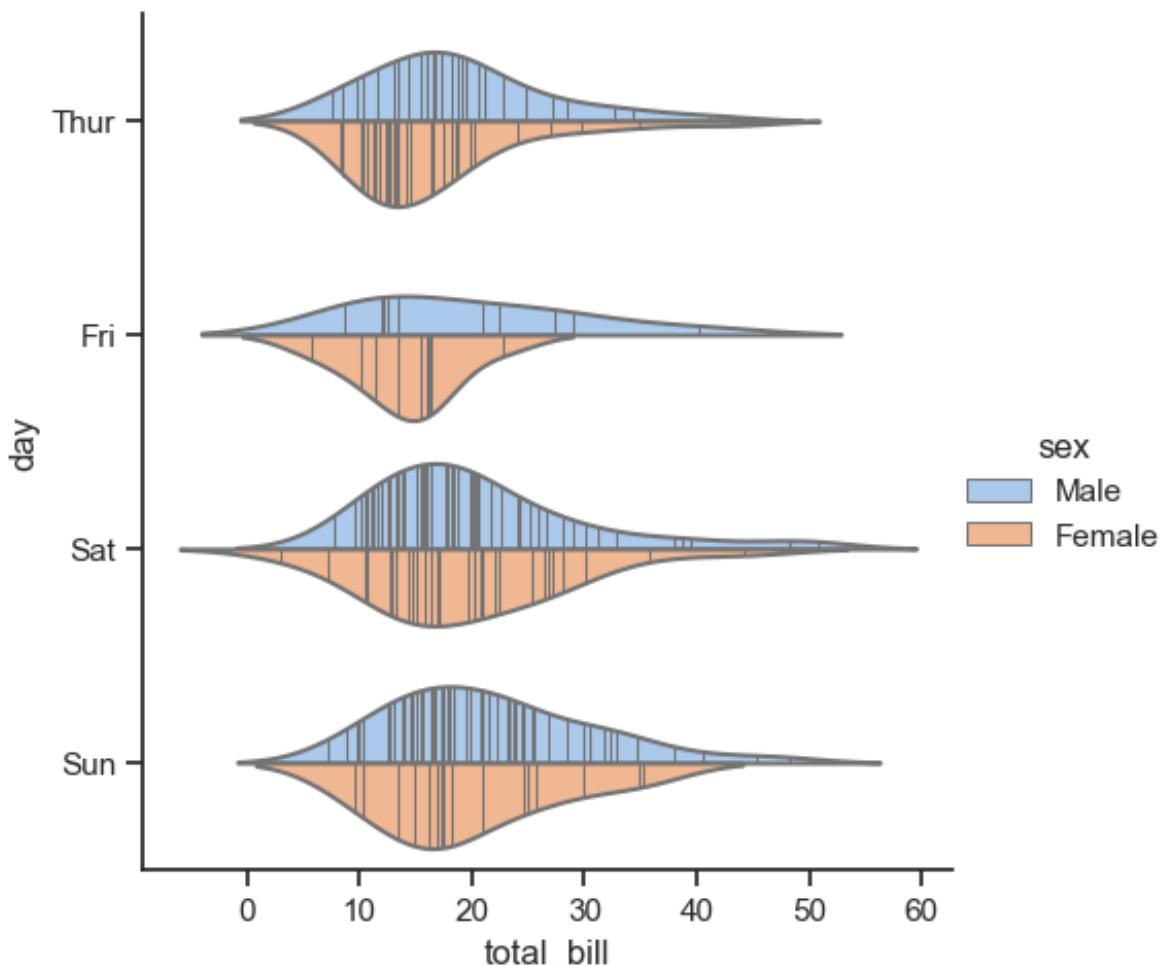
```
In [71]: sns.catplot(x="total_bill", y="day", hue="sex",
                    kind="violin", data=tips, split=True);
```



```
In [72]: sns.catplot(x="total_bill", y="day", hue="sex",
                     kind="violin", data=tips, split= True, inner= "stick");
```

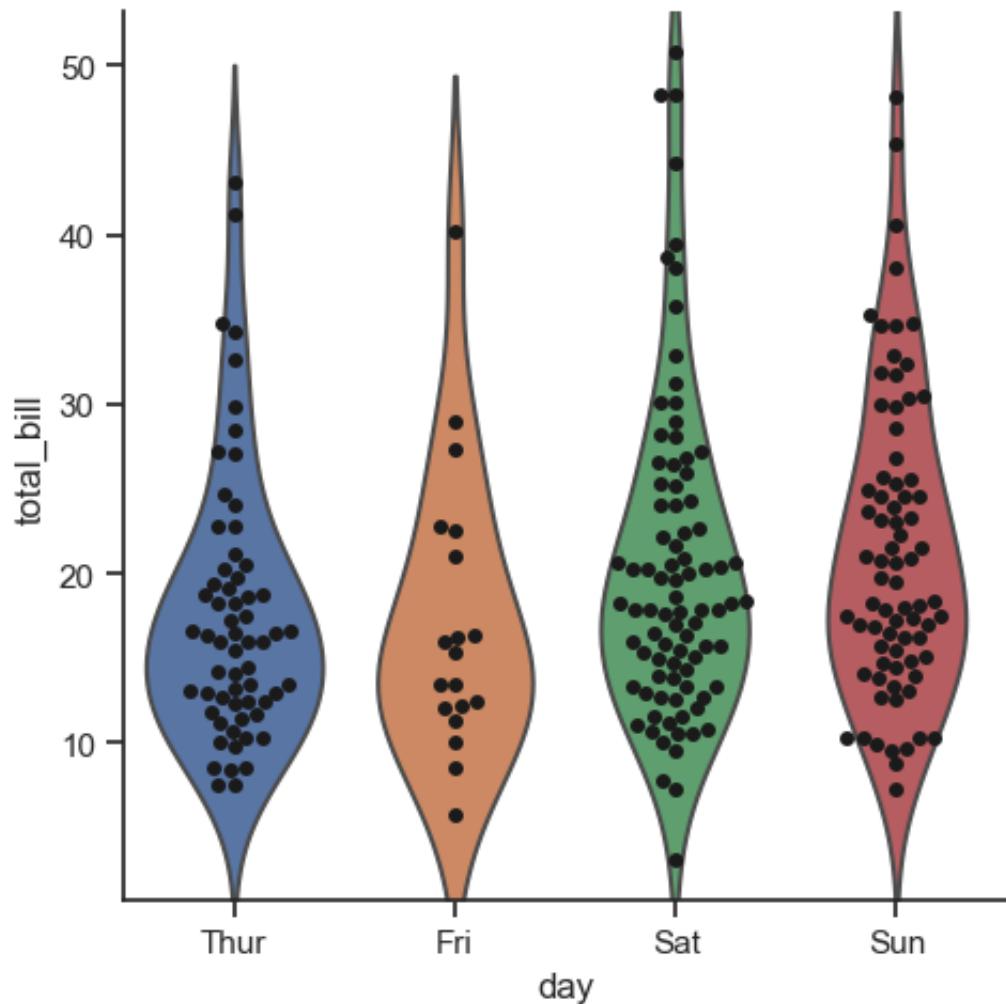


```
In [73]: sns.catplot(x="total_bill", y="day", hue="sex",
                    kind="violin", data=tips, split= True, inner= "stick",
```

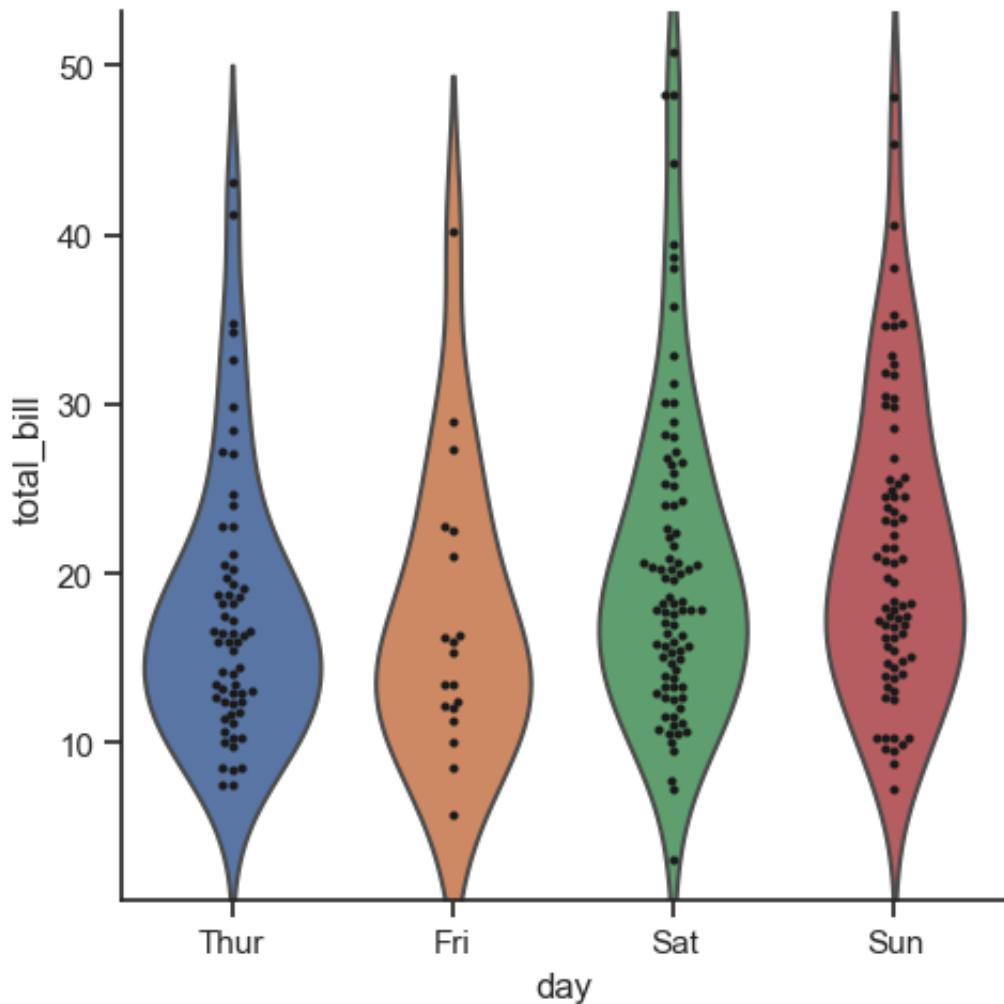


Также может быть полезно объединить `swarmplot()` или `stripplot()` с диаграммой ящиков или скрипичной диаграммой, чтобы показать каждое наблюдение вместе с кратким изложением распределения:

```
In [74]: g= sns.catplot( x= 'day', y= 'total_bill', kind= 'violin', inner= None  
sns.swarmplot(x= 'day', y= 'total_bill', color= 'k', data= tips, ax=
```



```
In [75]: g= sns.catplot( x= 'day', y= 'total_bill', kind= 'violin', inner= None  
sns.swarmplot(x= 'day', y= 'total_bill', size=3, color= 'k', data= df)
```



Статистическая оценка внутри категорий

В других случаях вместо отображения распределения внутри каждой категории может потребоваться оценка центральной тенденции значений.

Столбчатые диаграммы

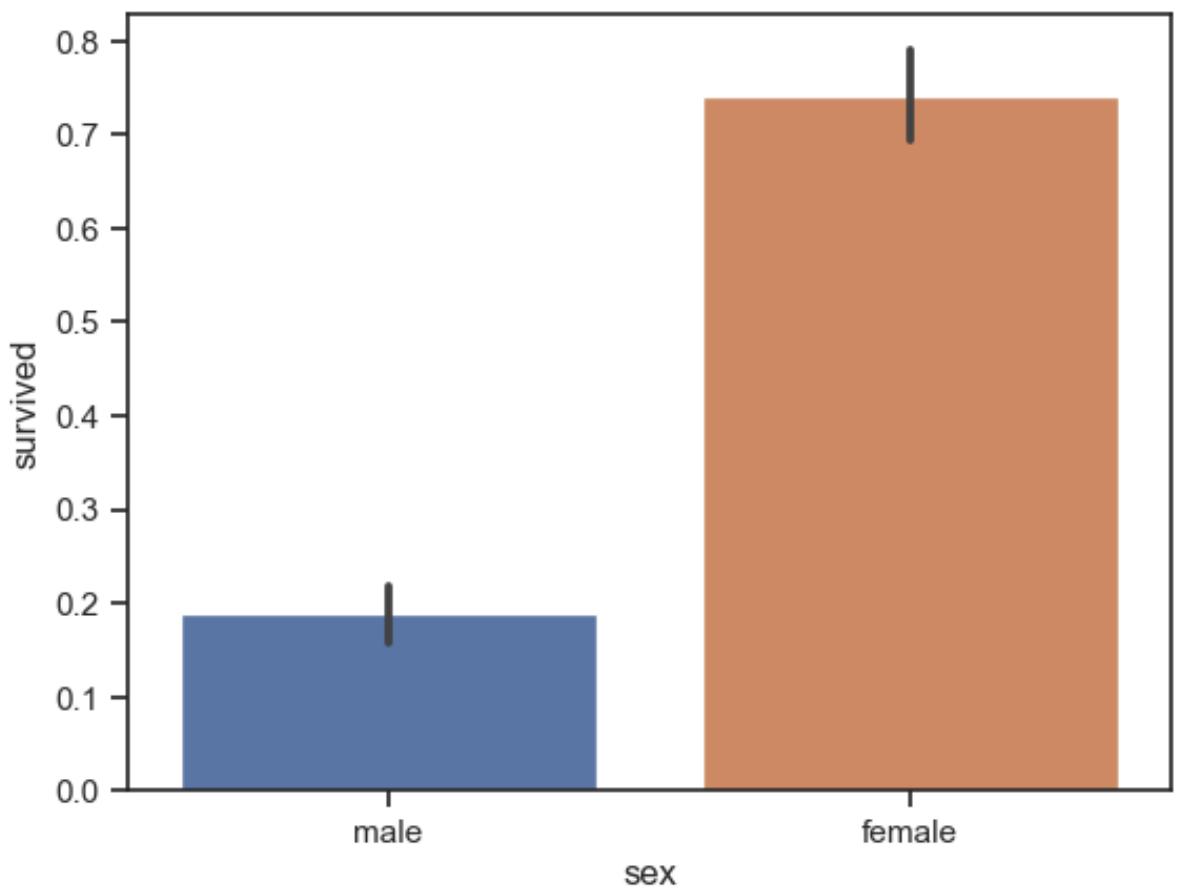
Распространённый стиль построения графиков, позволяющий достичь этой цели, – столбчатая диаграмма. В Seaborn функция barplot() работает с полным набором данных и применяет функцию для получения оценки (по умолчанию принимается среднее значение). При наличии нескольких наблюдений в каждой категории также вычисляется доверительный интервал вокруг оценки, который строится с использованием планок погрешностей (error bars):

```
In [76]: titanic.tail()
```

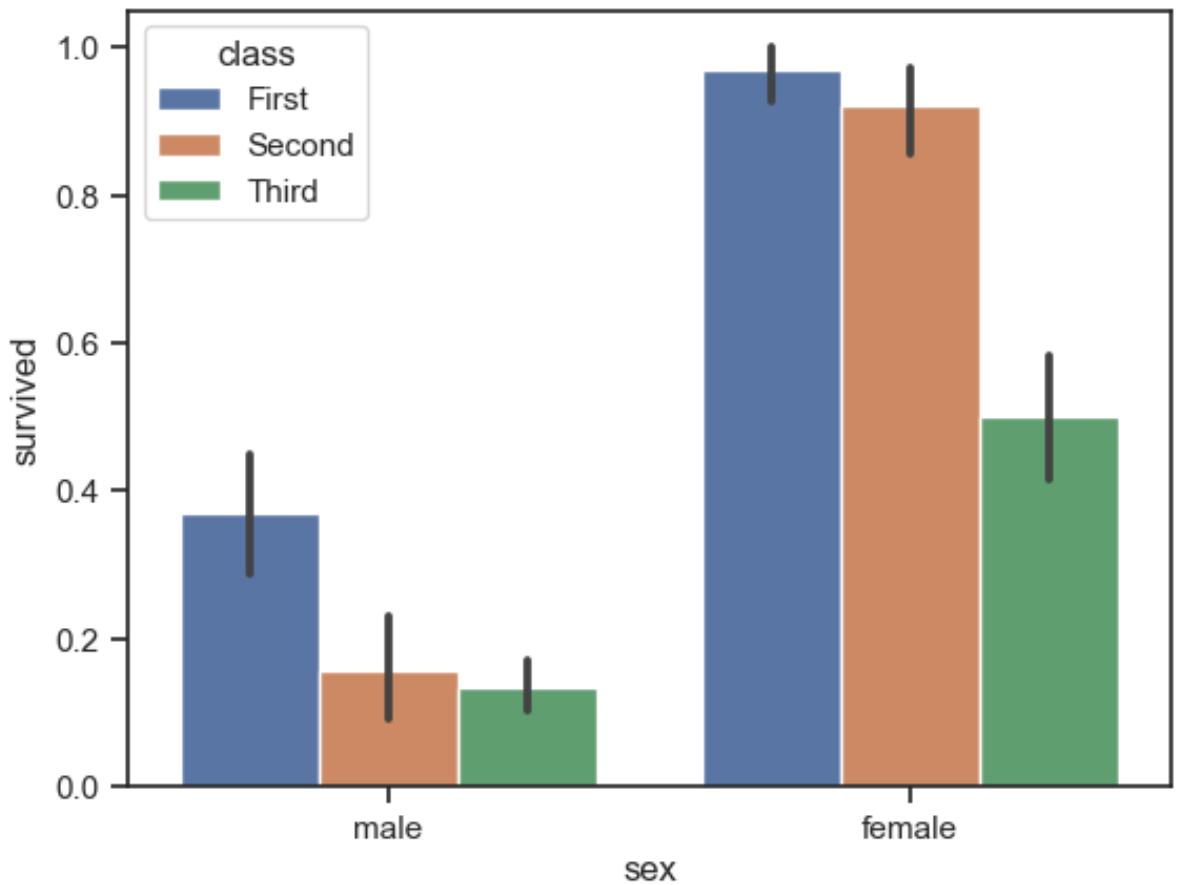
Out[76]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_
886	0	2	male	27.0	0	0	13.00	S	Second	man	
887	1	1	female	19.0	0	0	30.00	S	First	woman	
888	0	3	female	NaN	1	2	23.45	S	Third	woman	
889	1	1	male	26.0	0	0	30.00	C	First	man	
890	0	3	male	32.0	0	0	7.75	Q	Third	man	

```
In [77]: sns.barplot(x= 'sex', y= 'survived', data= titanic);
```

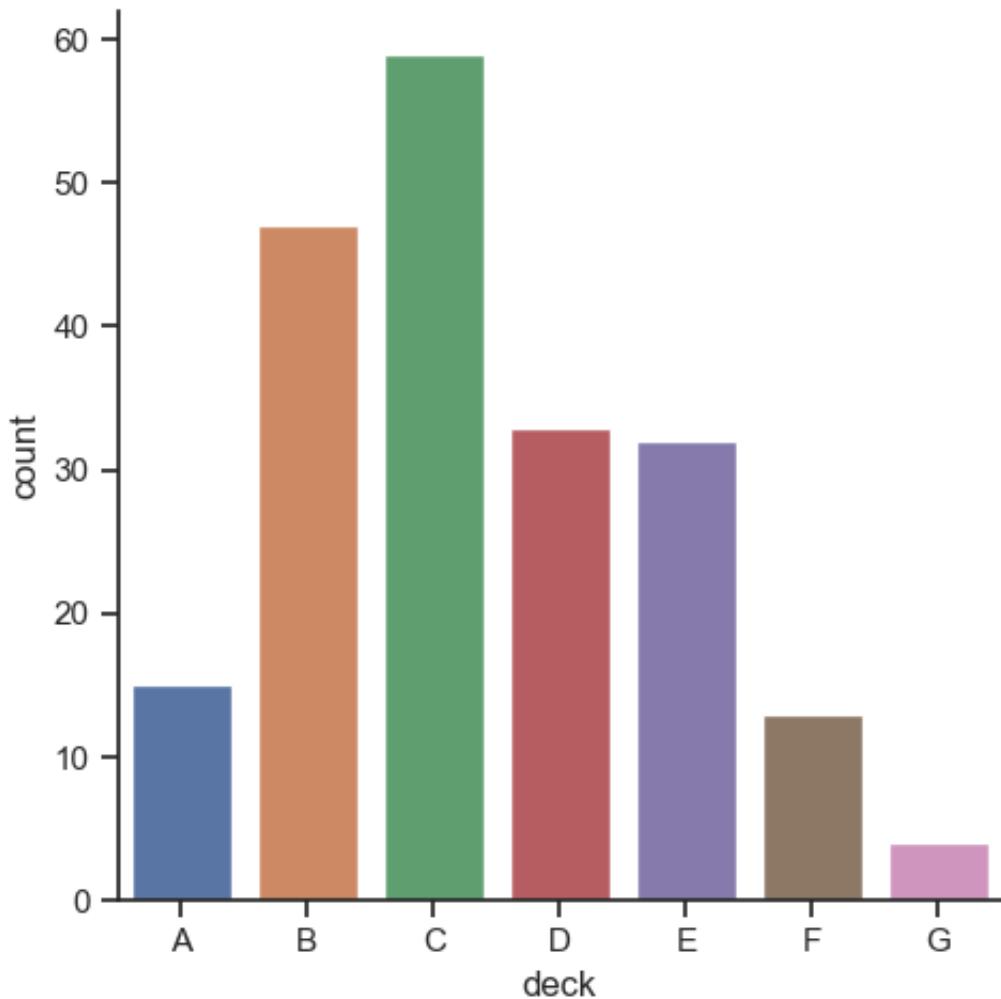


```
In [78]: sns.barplot(x= 'sex', y= 'survived',hue= 'class', data= titanic);
```

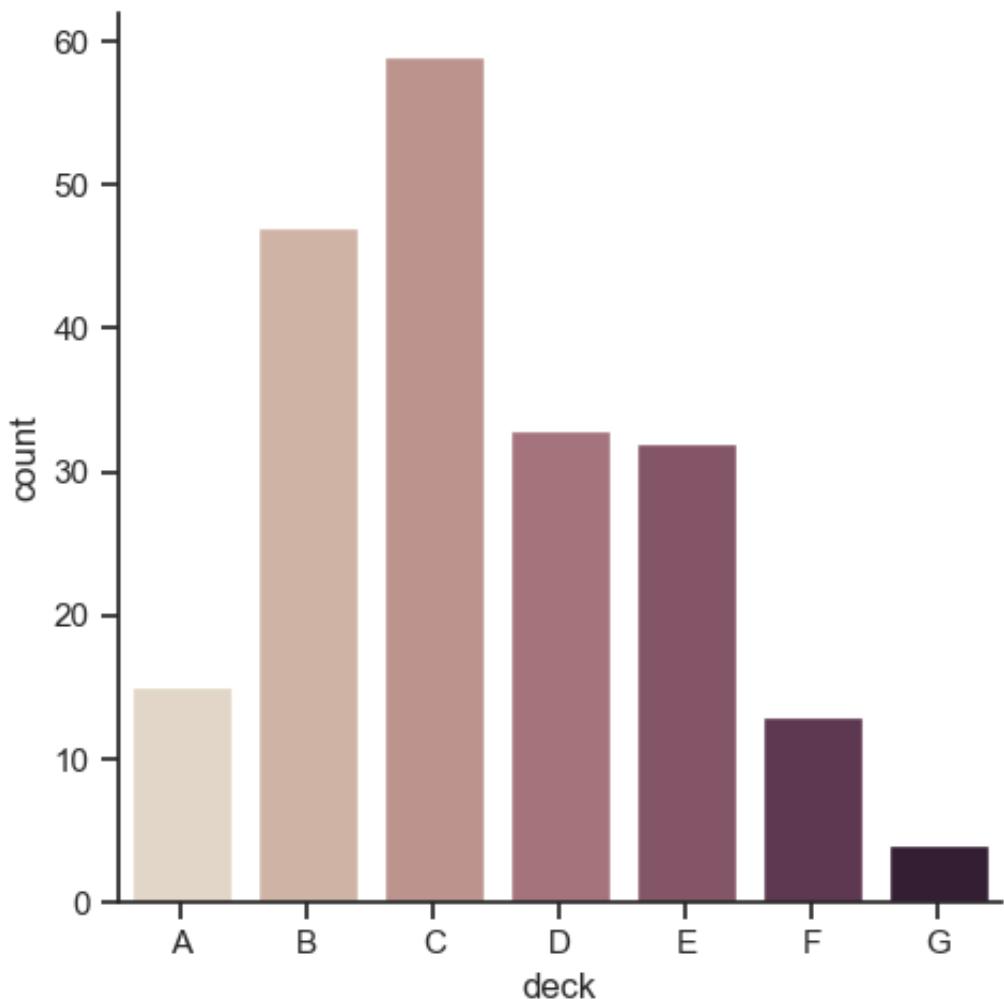


Особый случай столбчатой диаграммы — когда требуется отобразить количество наблюдений в каждой категории, а не вычислить статистику для второй переменной. Это похоже на гистограмму для категориальной, а не количественной переменной. В Seaborn это легко сделать с помощью функции `countplot()` :

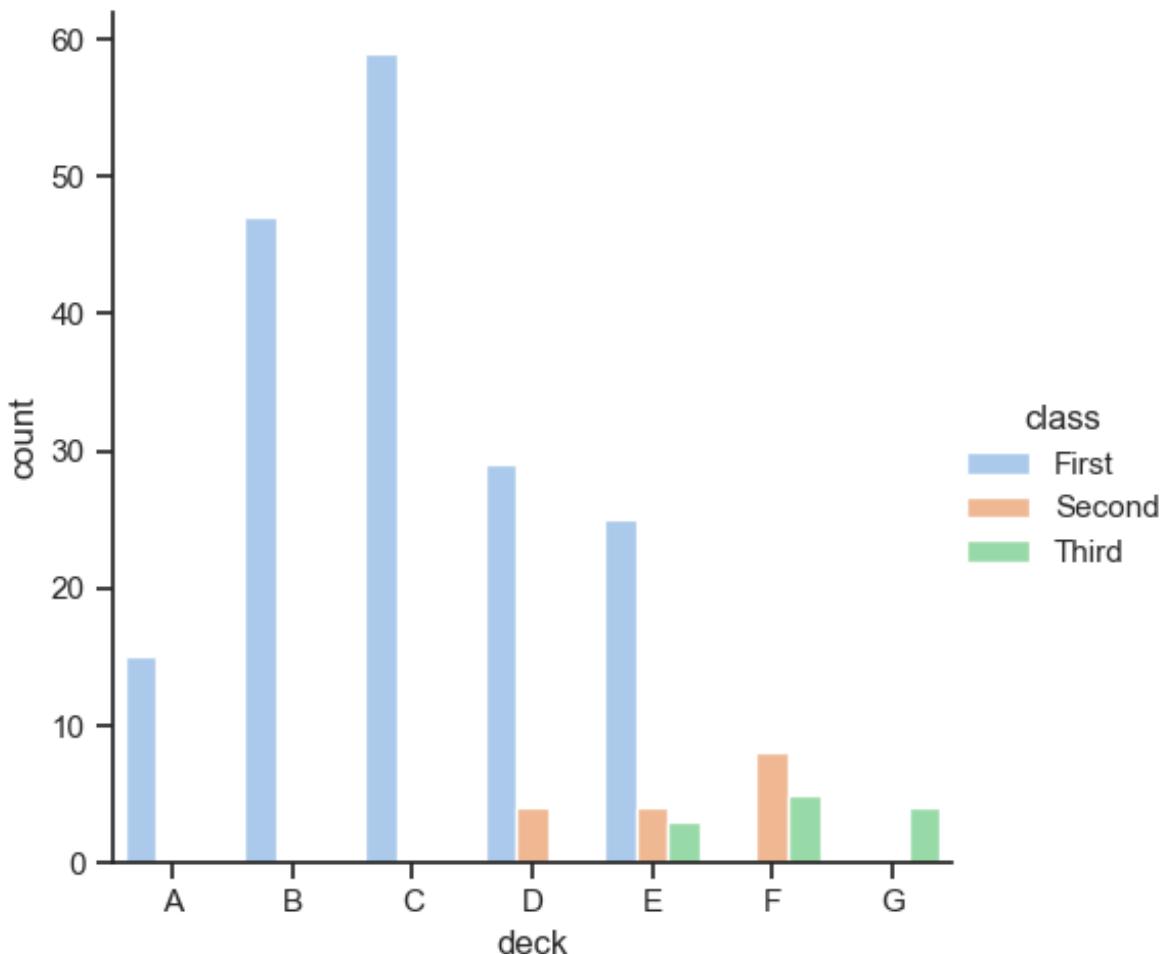
```
In [79]: sns.catplot(x= 'deck', kind= 'count', data= titanic);
```



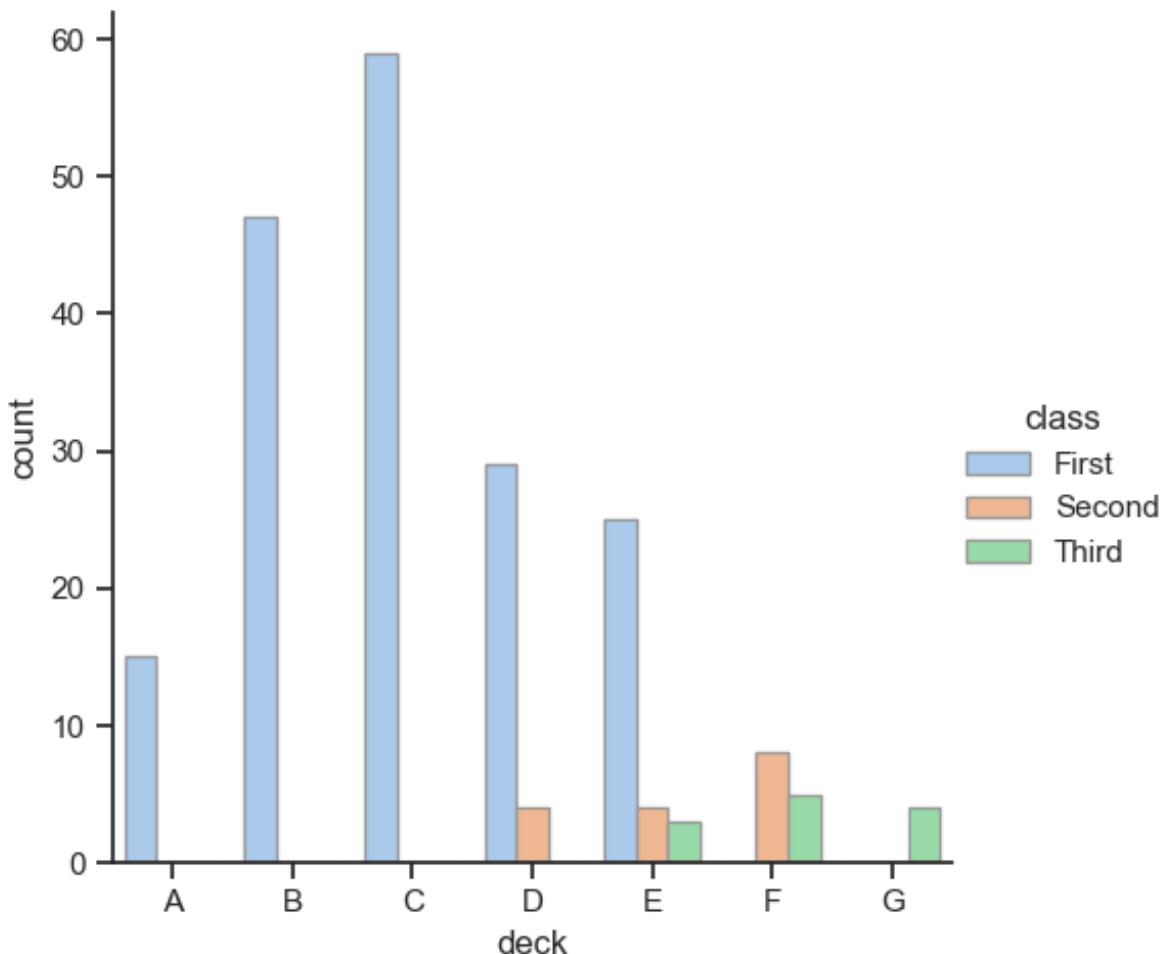
```
In [80]: sns.catplot(x= 'deck', kind= 'count', data= titanic, palette= "ch..")
```



```
In [81]: sns.catplot(x= 'deck', kind= 'count', data= titanic, palette= "pastel1")
```

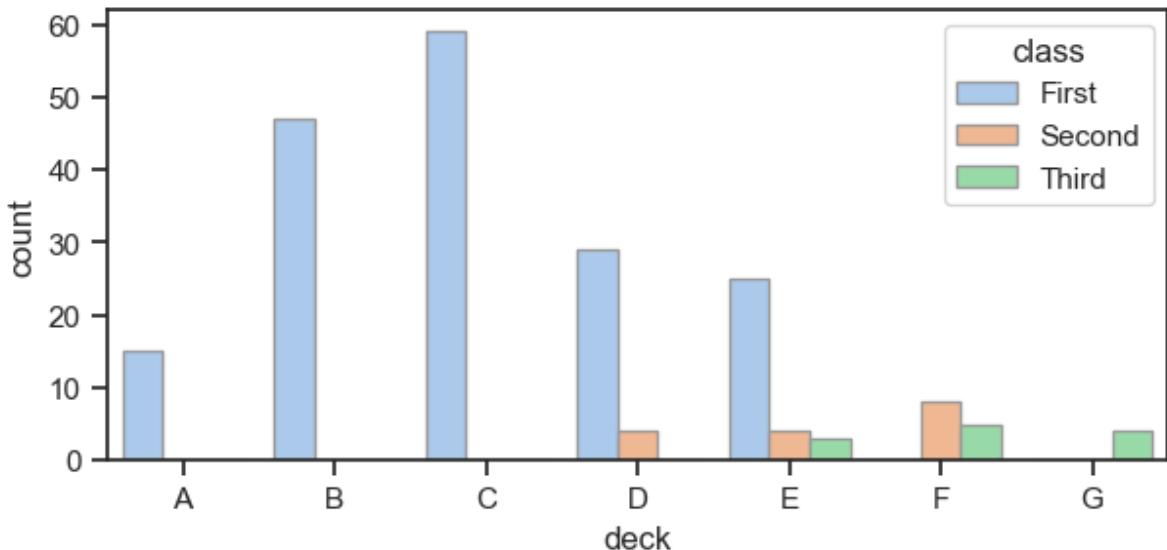


```
In [82]: sns.catplot(x= 'deck', kind= 'count', data= titanic, palette= "pastel")
```



Чтобы управлять размером и формой графиков, создаваемых функциями, обсуждаемыми выше, необходимо самостоятельно настроить рисунок с помощью команд matplotlib:

```
In [83]: f, ax = plt.subplots(figsize=(7, 3))
sns.countplot(x= 'deck', data= titanic, palette= "pastel", hue= 'class')
```



Точечные графики

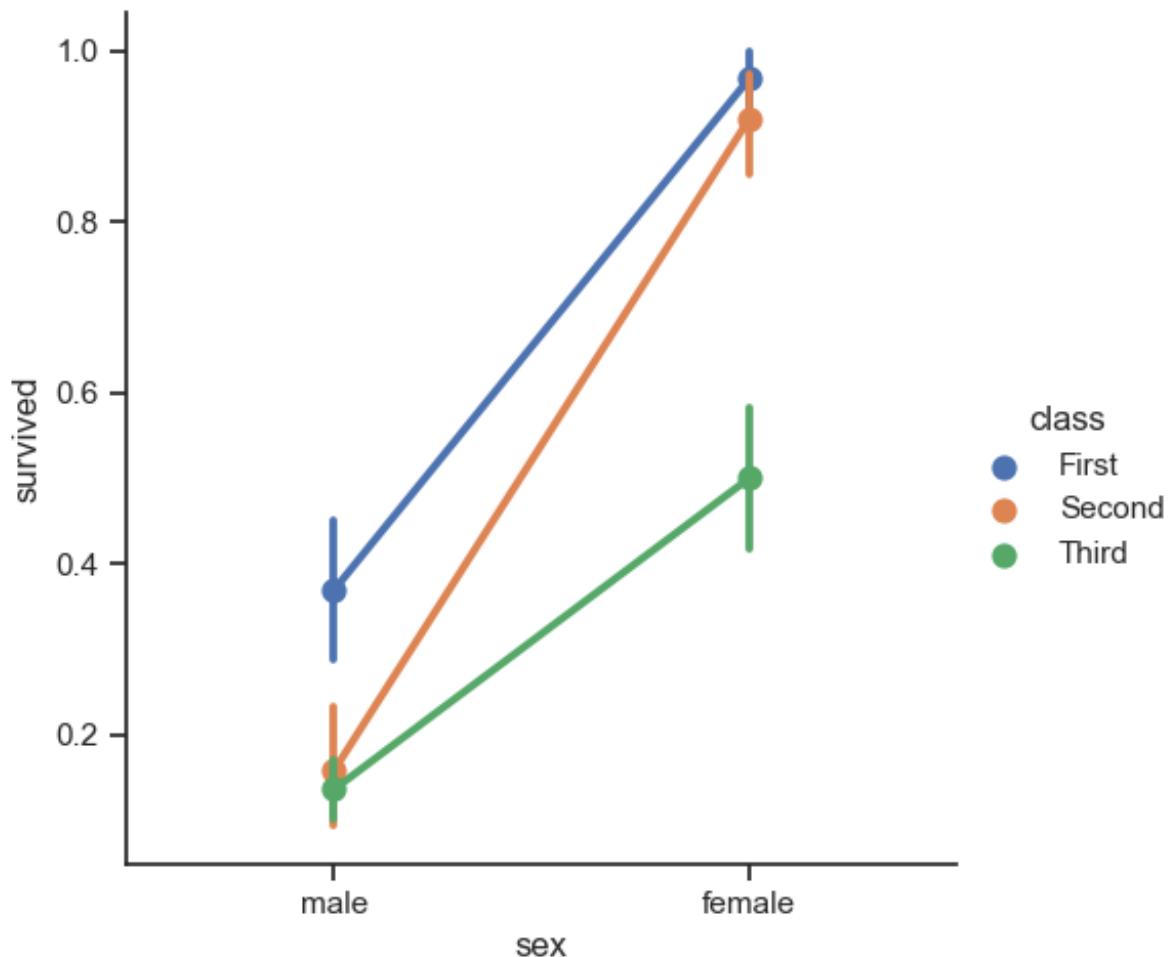
Альтернативный стиль визуализации той же информации предлагает функция `pointplot()`. Эта функция также кодирует значение оценки с высотой на другой оси, но вместо отображения сплошной полосы она отображает точечную оценку и доверительный интервал. Кроме того, функция `pointplot()` соединяет точки из одной категории оттенка. Это позволяет легко увидеть, как основная взаимосвязь меняется в зависимости от семантики оттенка, поскольку ваши глаза довольно хорошо улавливают разницу в наклонах:

```
In [84]: titanic.head()
```

Out [84]:

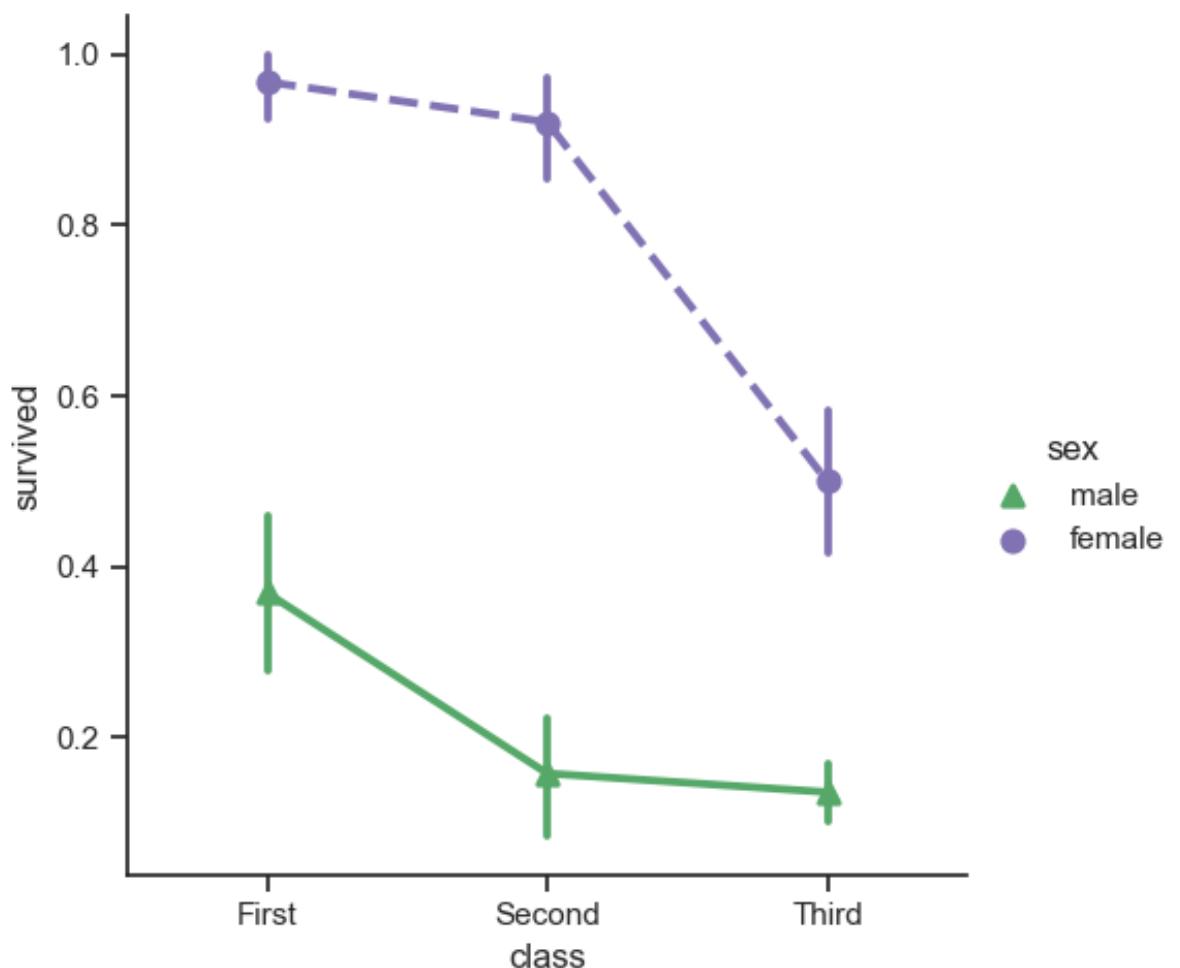
	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_m
0	0	3	male	22.0	1	0	7.2500	S	Third	man	T
1	1	1	female	38.0	1	0	71.2833	C	First	woman	F
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	F
3	1	1	female	35.0	1	0	53.1000	S	First	woman	F
4	0	3	male	35.0	0	0	8.0500	S	Third	man	T

```
In [85]: sns.catplot(x= 'sex', y= 'survived', kind= 'point', data= titanic,
```



Может быть хорошей идеей варьировать маркер и/или стиль линии вместе с оттенком, чтобы сделать рисунки максимально доступными и хорошо воспроизводимыми в черно-белом варианте:

```
In [86]: sns.catplot(x= 'class', y= 'survived', hue= 'sex',
                    palette= {'male':'g', 'female':'m'},
                    markers= ['^', 'o'], linestyles=[ '--', '-'],
                    kind= 'point', data= titanic);
```



Визуализация распределения данных:

- displot()
- kdeplot()
- jointplot()
- rugplot()

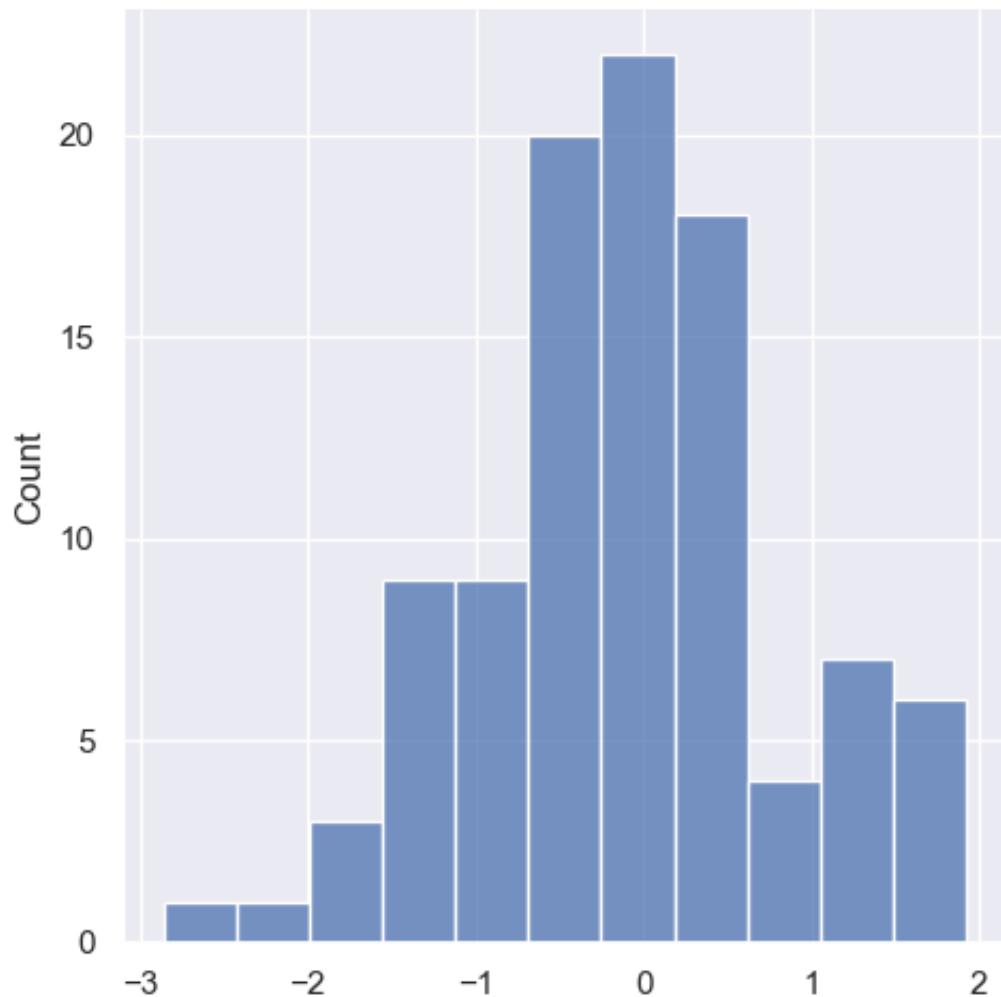
Displot:

1. одномерные распределения

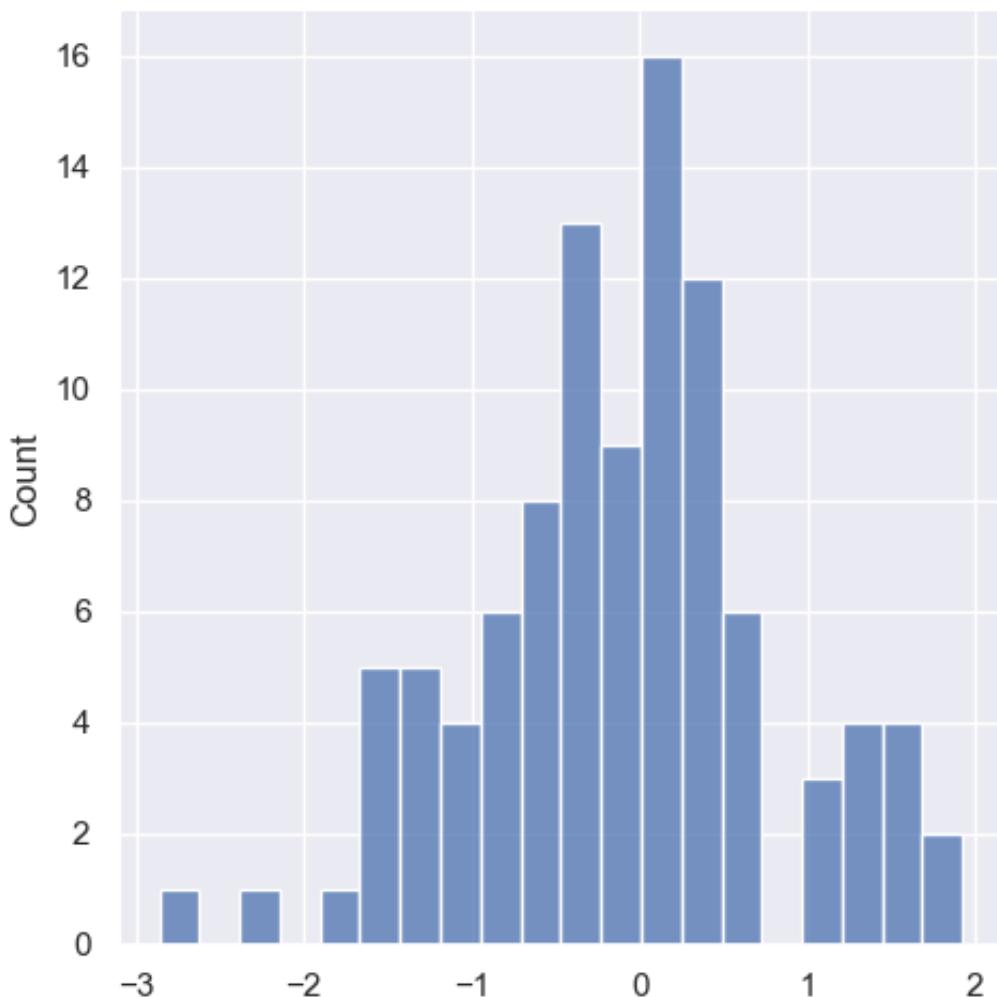
По умолчанию строится гистограмма и подбирается оценка плотности ядра (KDE).

```
In [87]: sns.set()
```

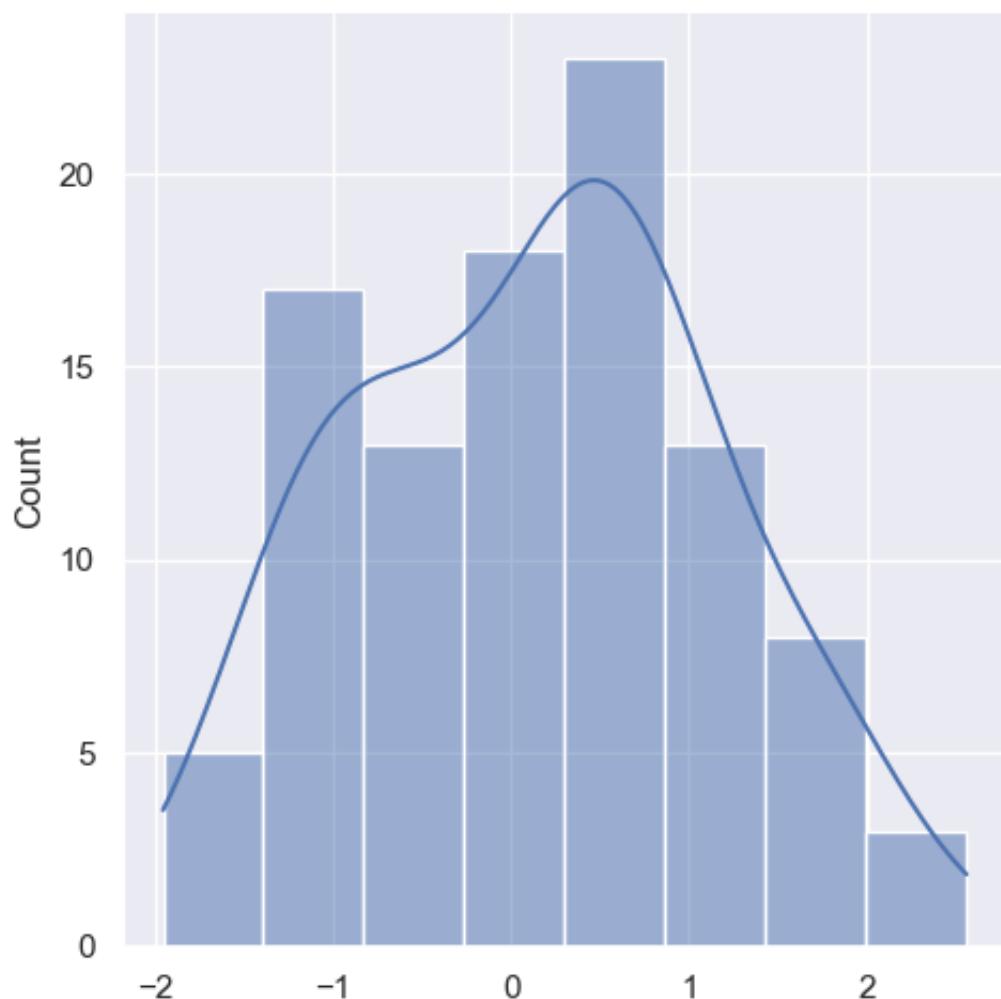
```
In [88]: x = np.random.normal(size=100)  
sns.displot(x);
```



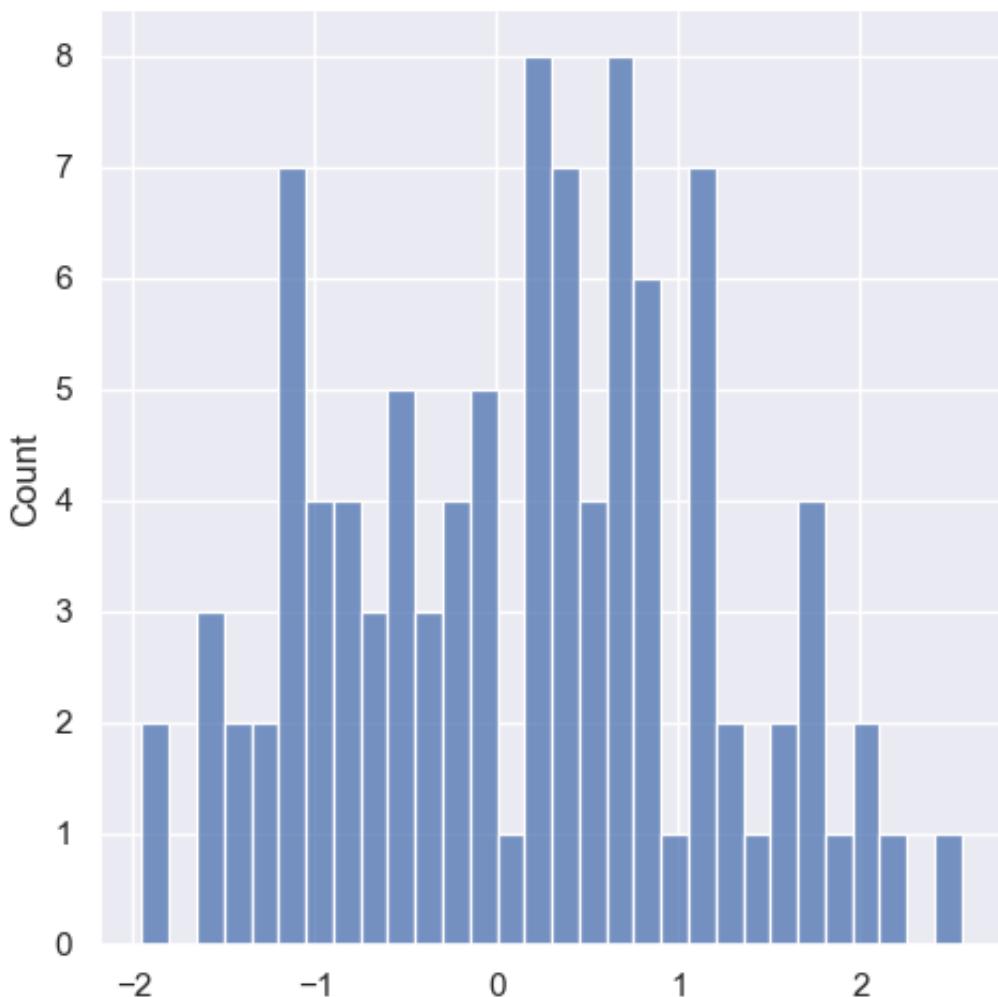
```
In [89]: sns.displot(x, bins= 20);
```



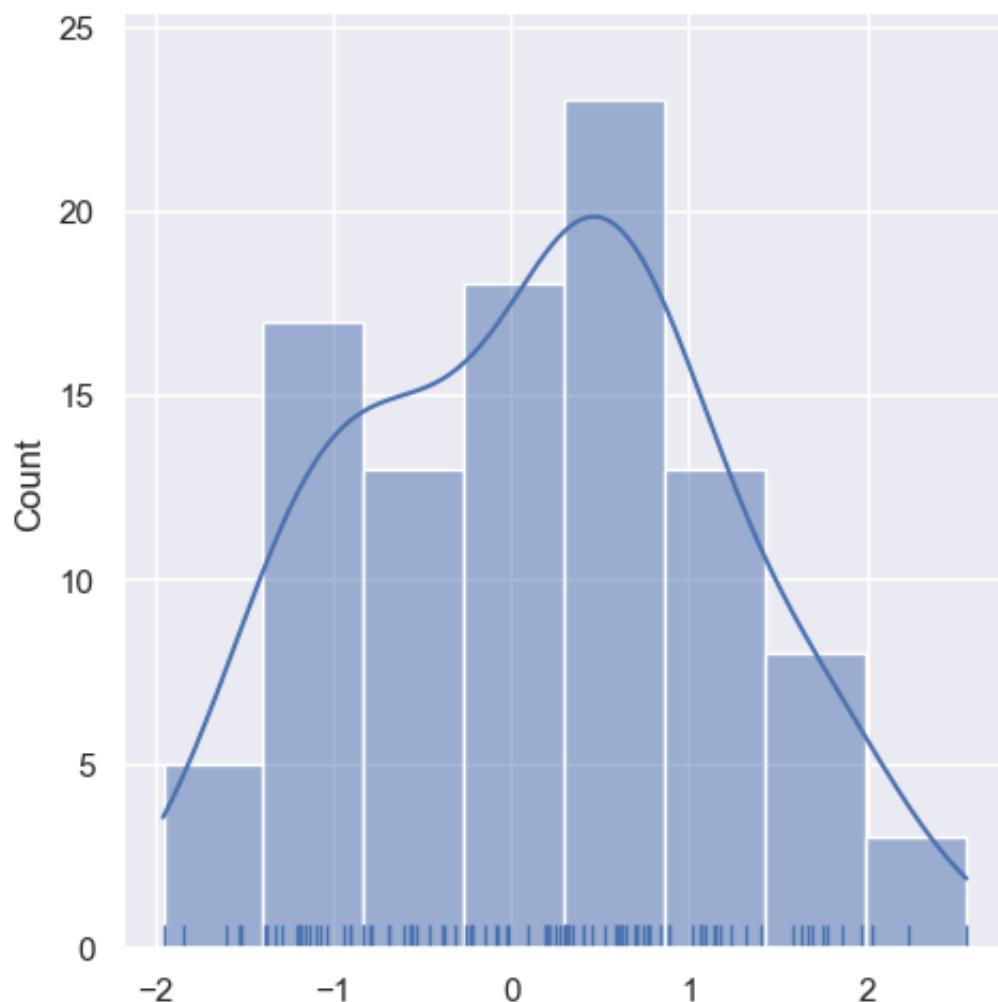
```
In [90]: x= np.random.randn(100)  
sns.displot(x, kde=True);
```



```
In [91]: sns.displot(x, bins= 30);
```



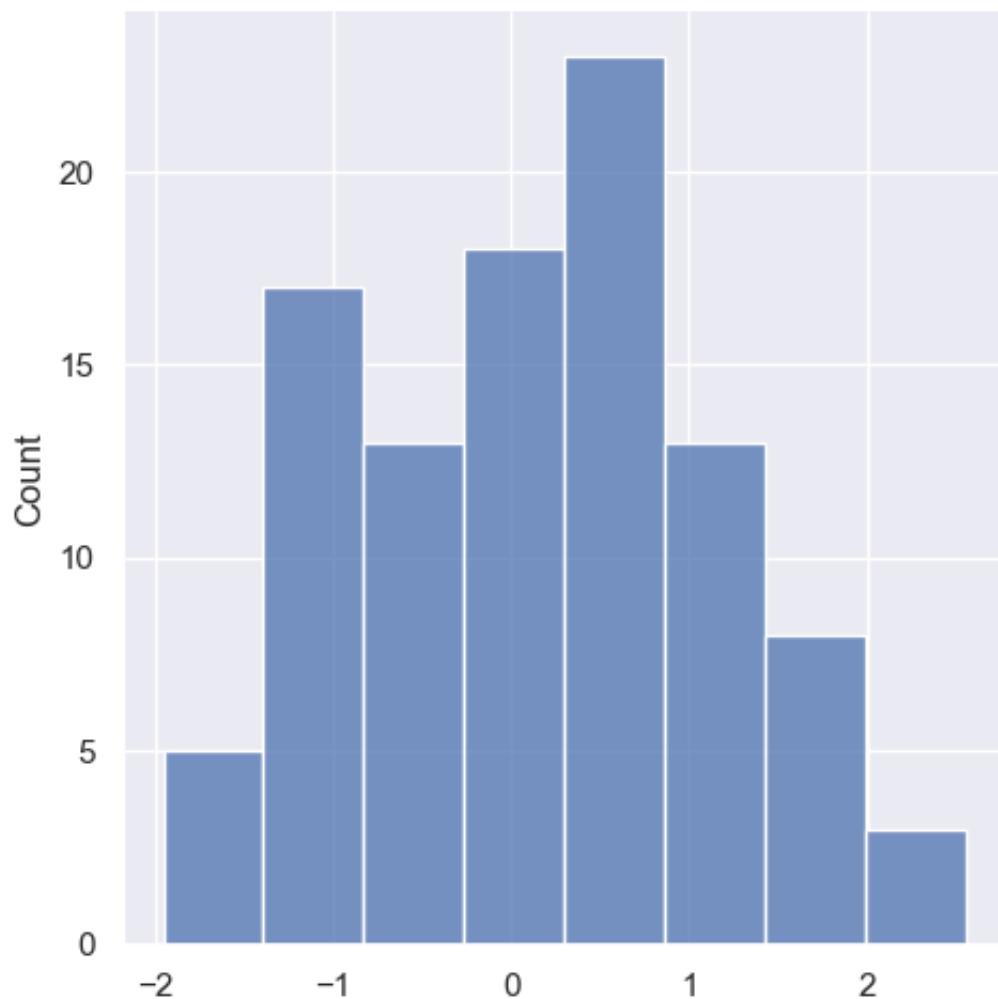
```
In [92]: sns.displot(x, kde=True, rug=True);
```



Гистограмма:

Гистограмма представляет собой распределение данных путём формирования интервалов вдоль диапазона данных с последующим отображением столбцов, показывающих количество наблюдений, попадающих в каждый интервал. В matplotlib она также называется hist.

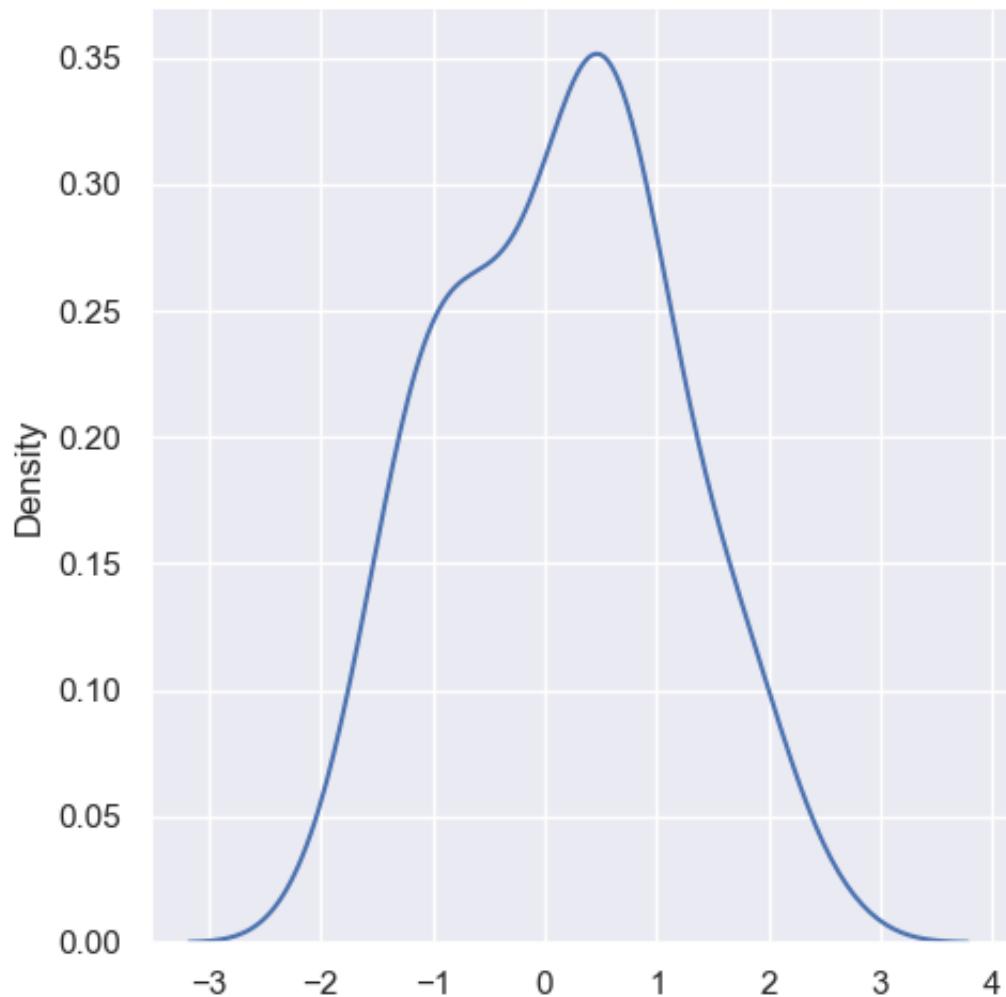
```
In [93]: sns.displot(x, kde=False, rug=False);
```



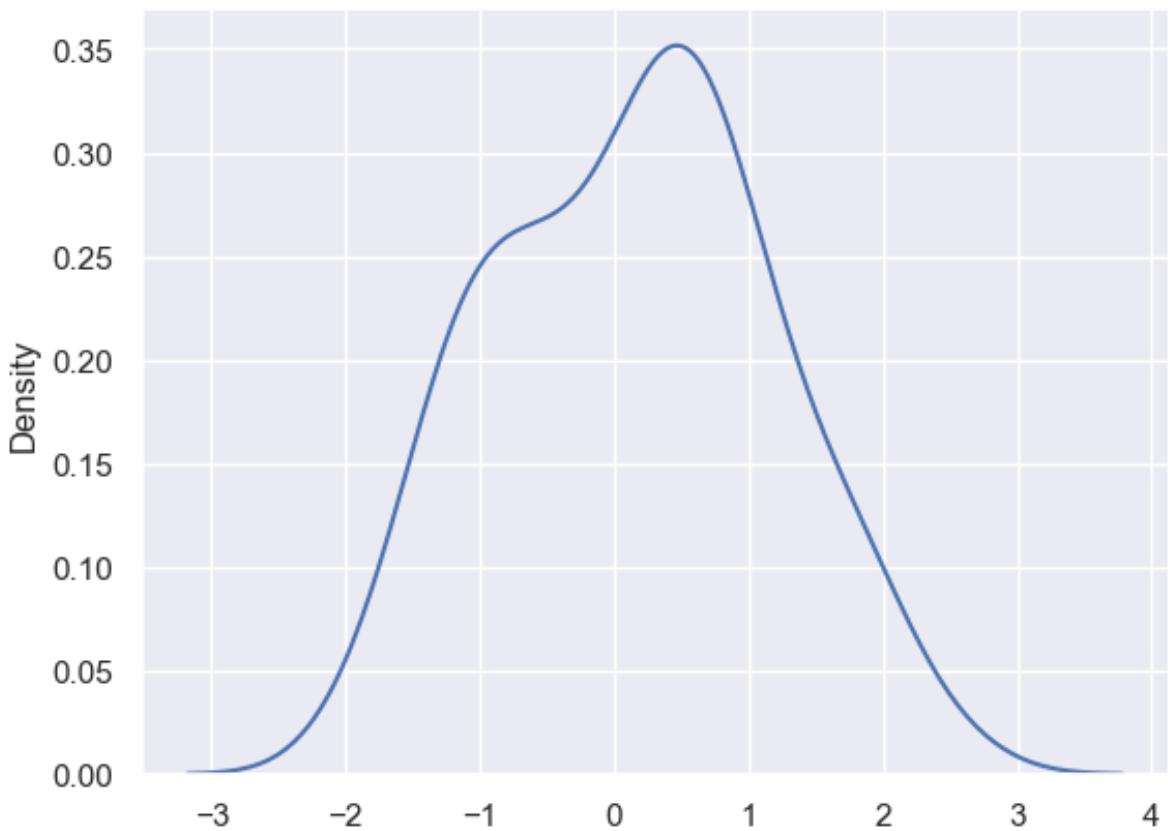
kde (kernel density estimation, оценка плотности ядра)

Как и гистограмма, графики KDE отображают плотность распределения по одной оси и переменную по другой оси:

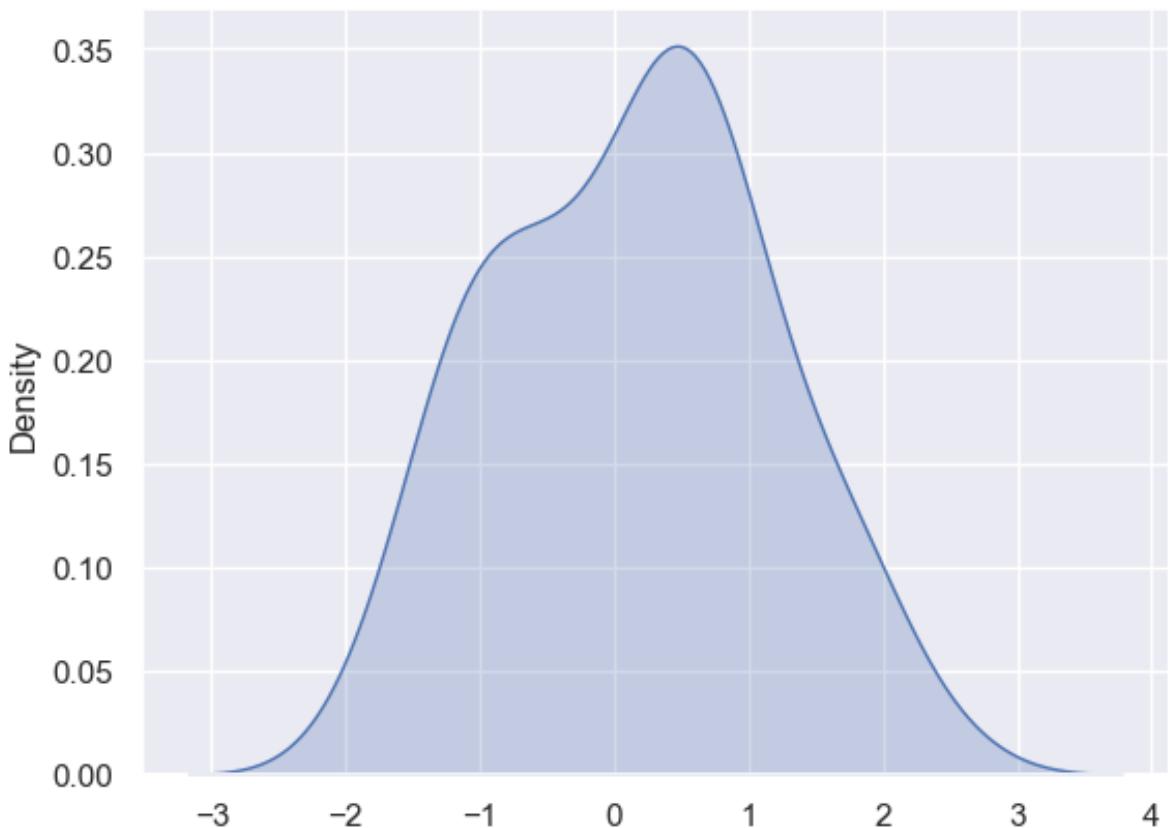
```
In [94]: sns.displot(x, kind='kde');
```



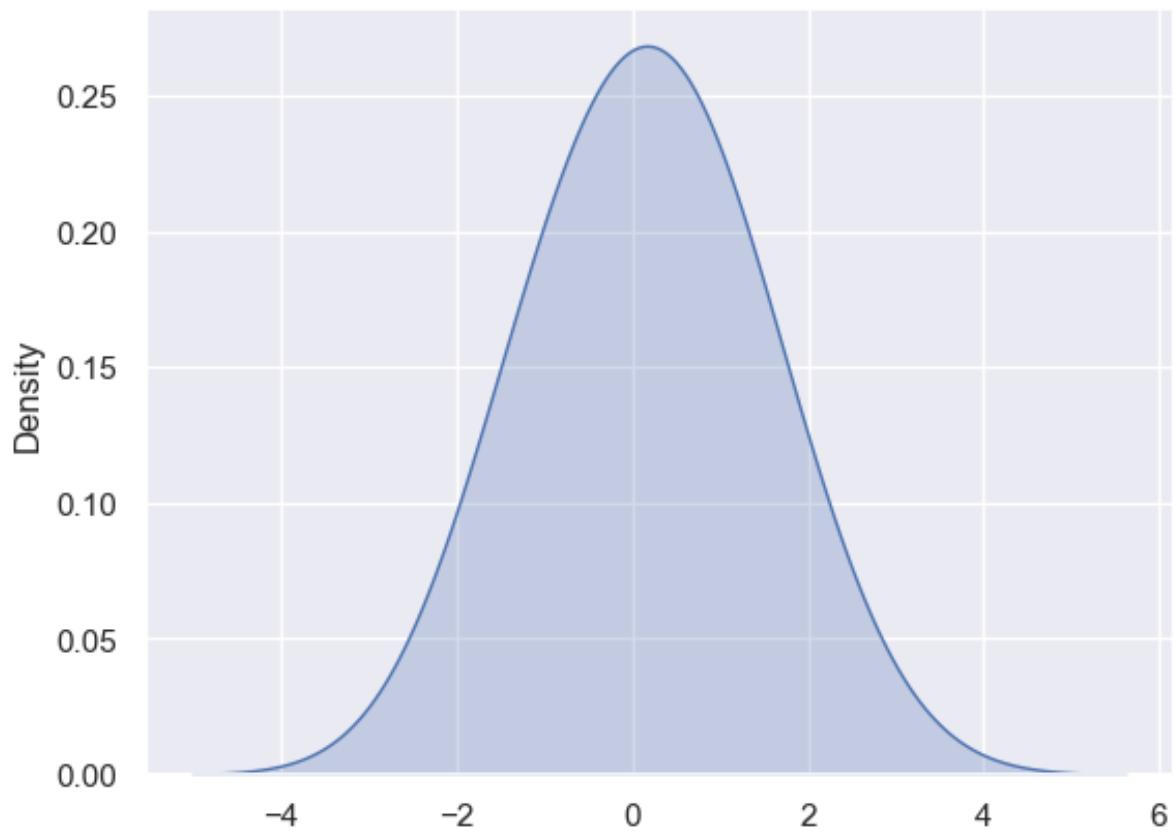
```
In [95]: sns.kdeplot(x);
```



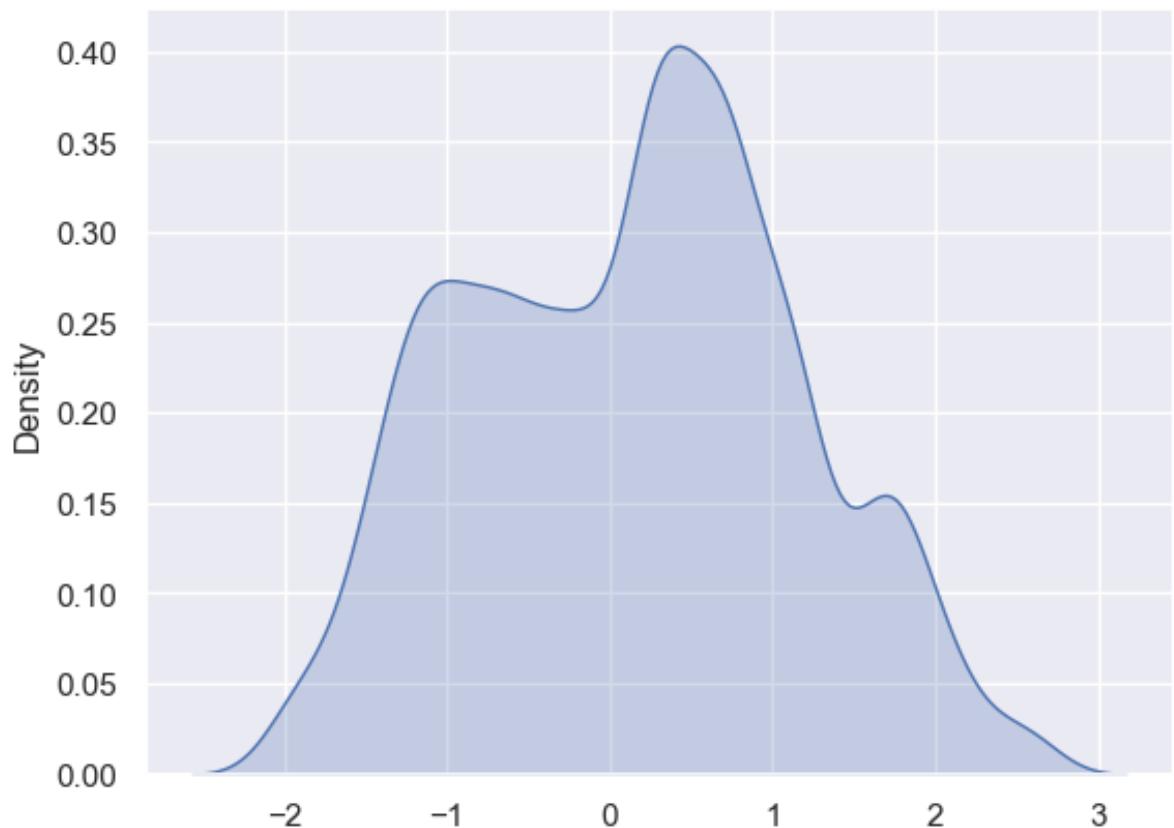
```
In [96]: sns.kdeplot(x, fill=True);
```



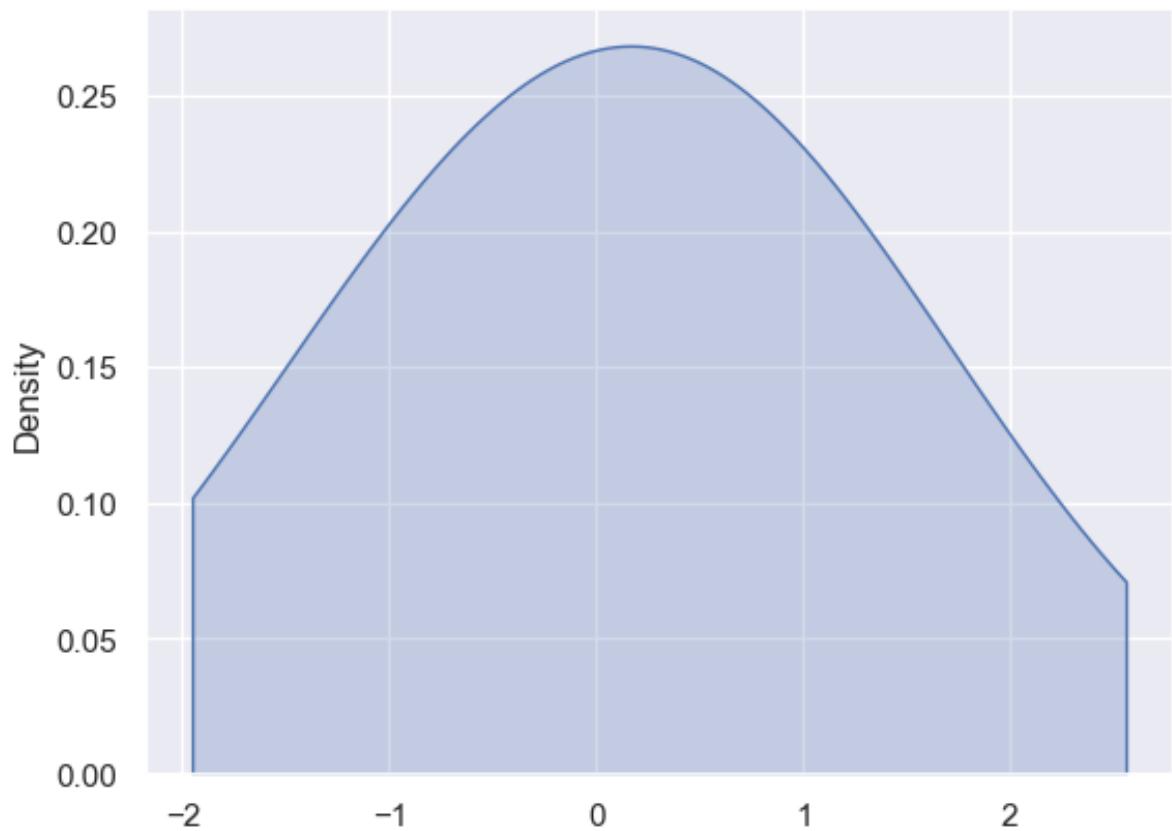
```
In [97]: sns.kdeplot(x, fill=True, bw_method=1);
```



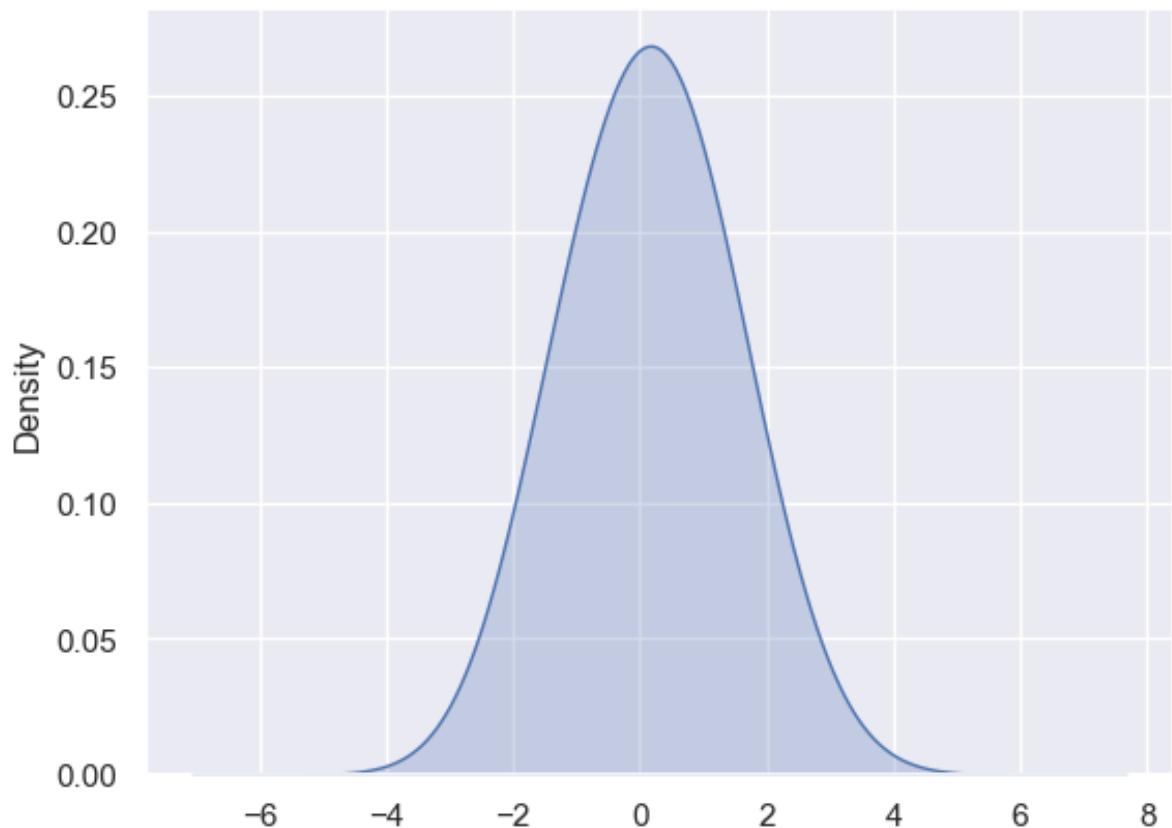
```
In [98]: sns.kdeplot(x, fill=True, bw_method=0.2);
```



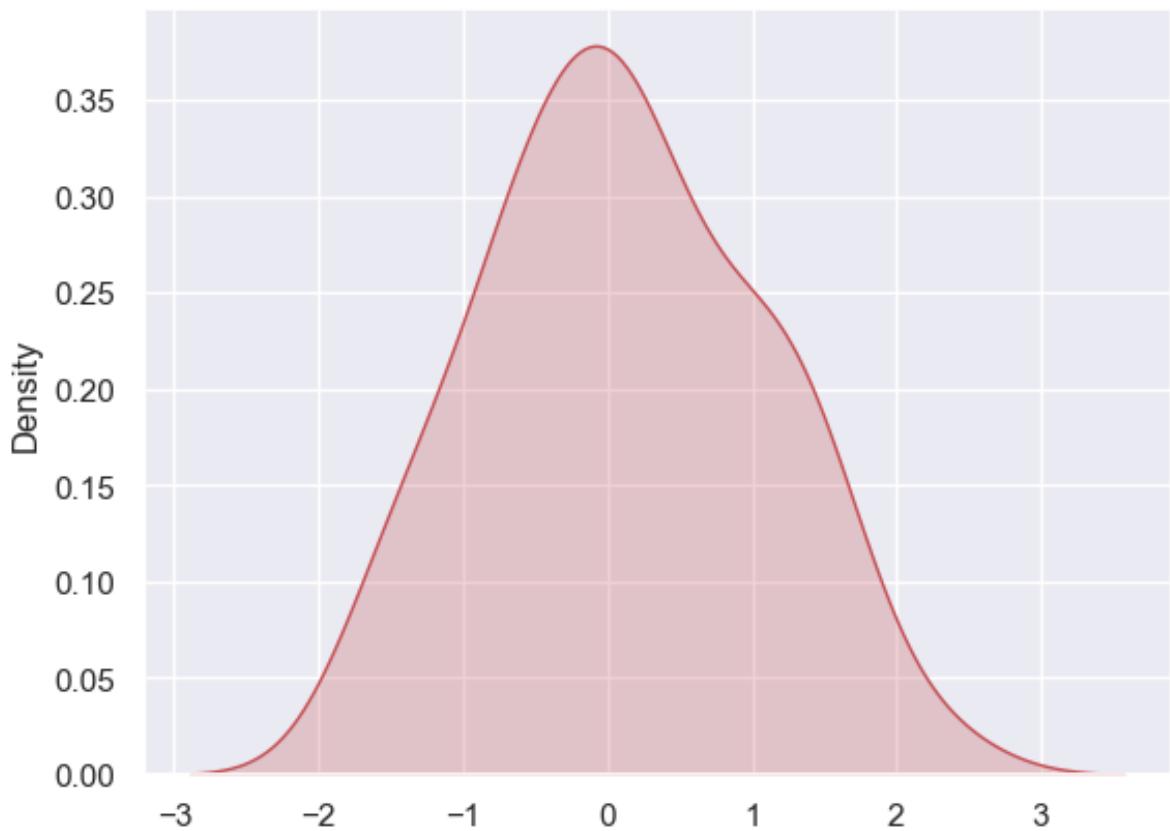
```
In [99]: sns.kdeplot(x, fill=True, bw_method=1, cut=0);
```



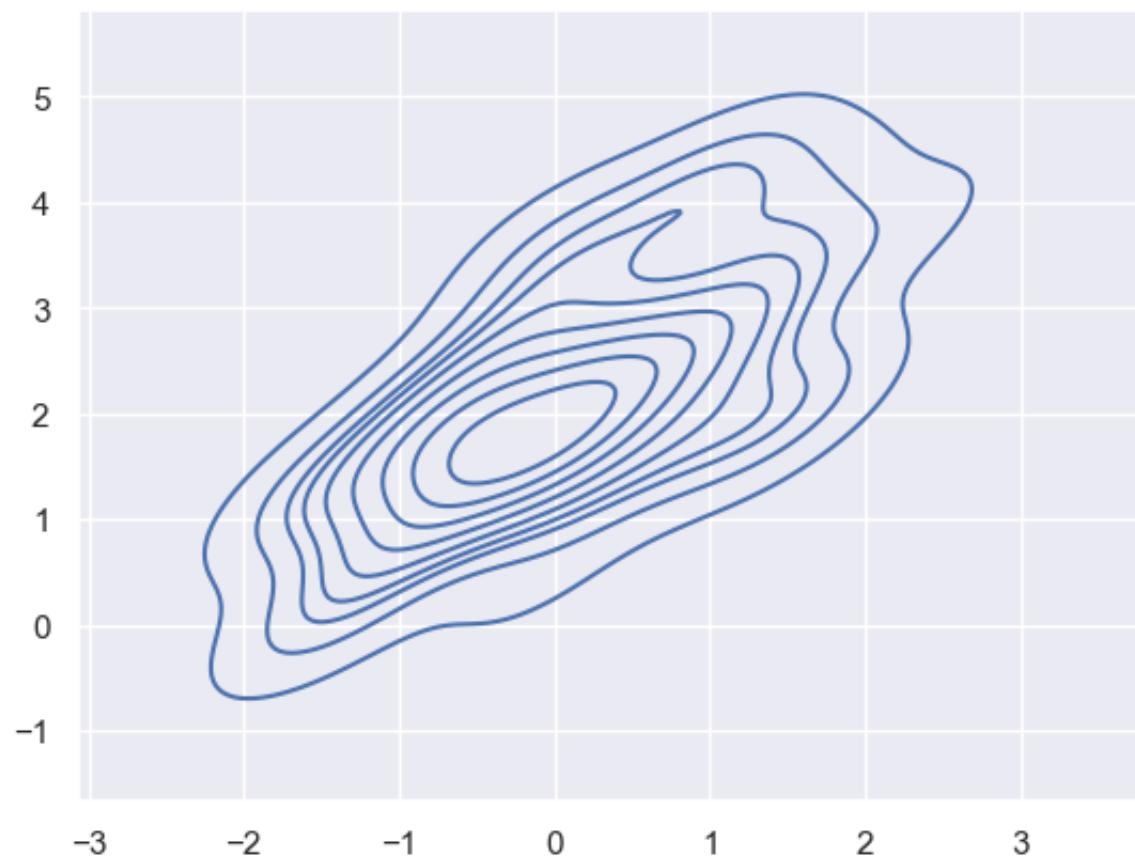
```
In [100]: sns.kdeplot(x, fill=True, bw_method=1, cut=5);
```



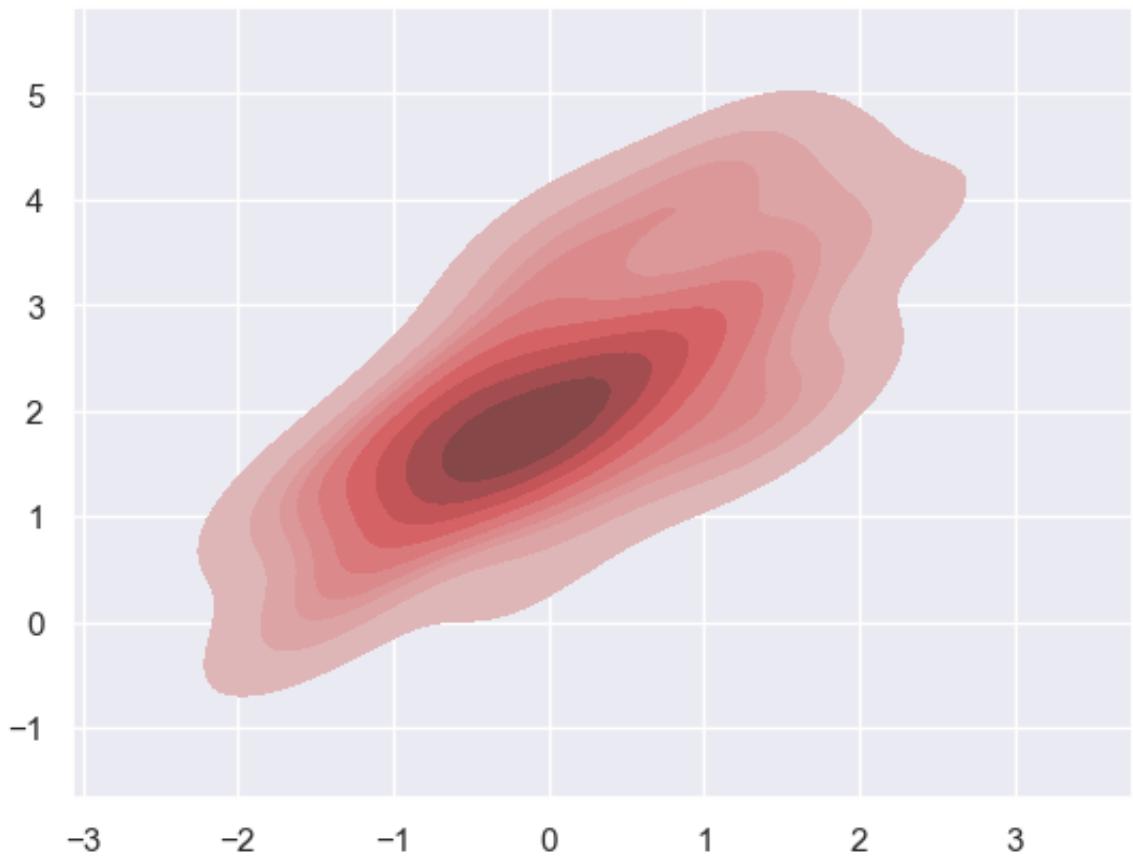
```
In [101]: mean, cov = [0, 2], [(1, .5), (.5, 1)]
x, y = np.random.multivariate_normal(mean, cov, size=50).T
ax = sns.kdeplot(x, color='r', fill=True)
```



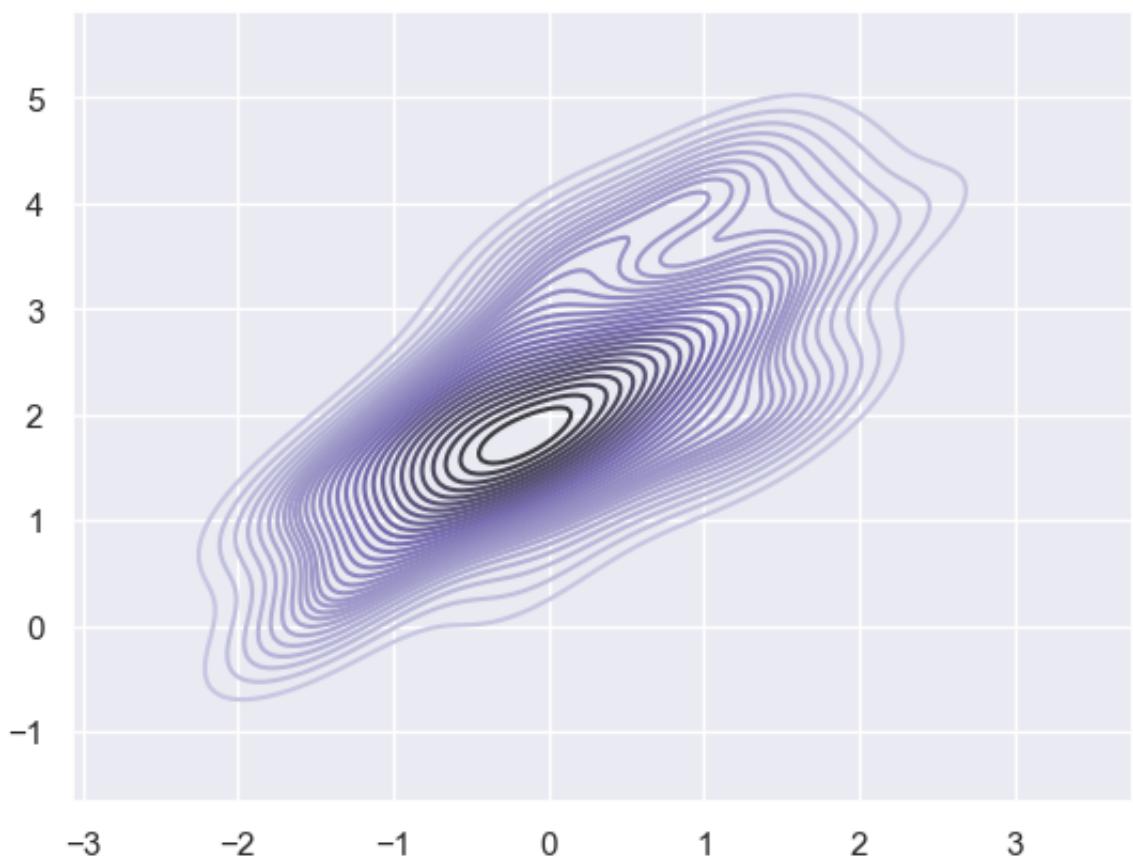
```
In [102]: ax = sns.kdeplot(x=x, y=y)
```



```
In [103]: ax = sns.kdeplot(x=x, y=y, color= 'r', fill= True)
```

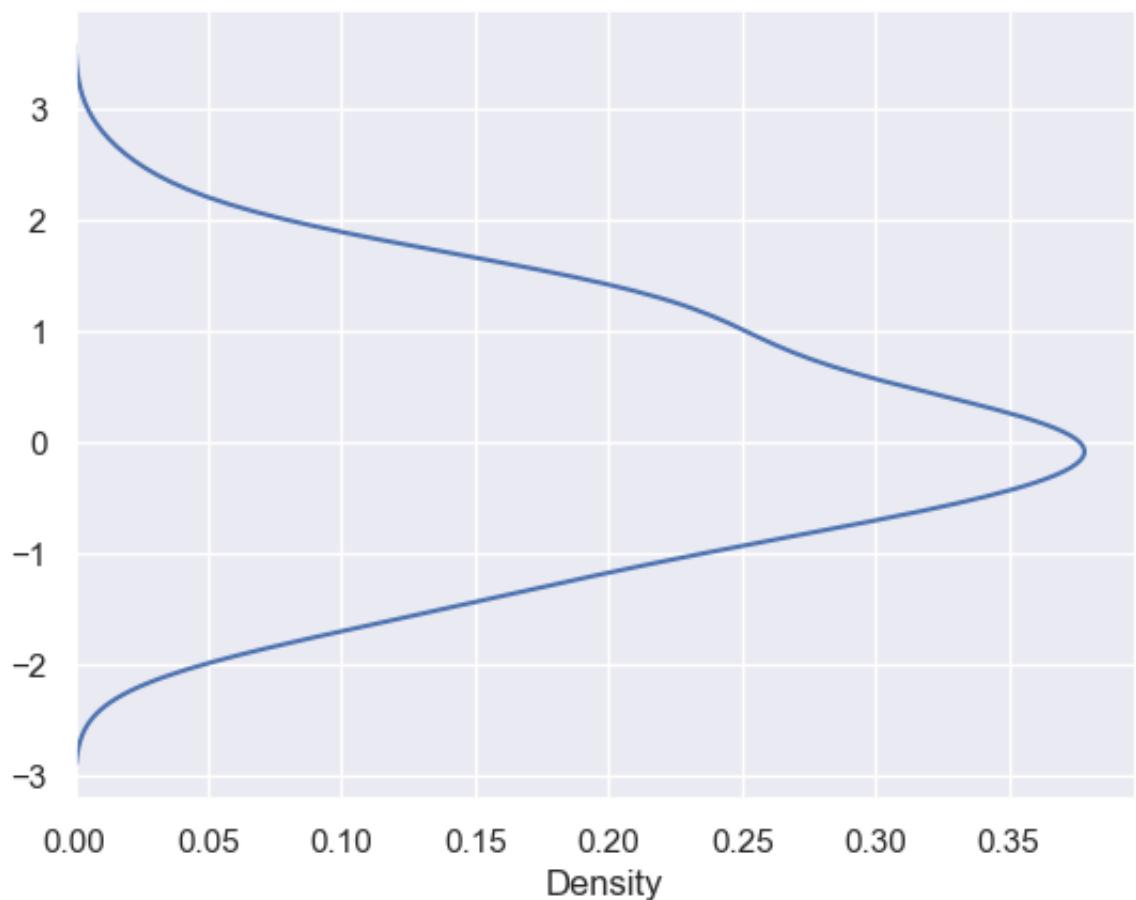


```
In [104]: ax = sns.kdeplot(x=x, y=y, n_levels=30, cmap="Purples_d")
```



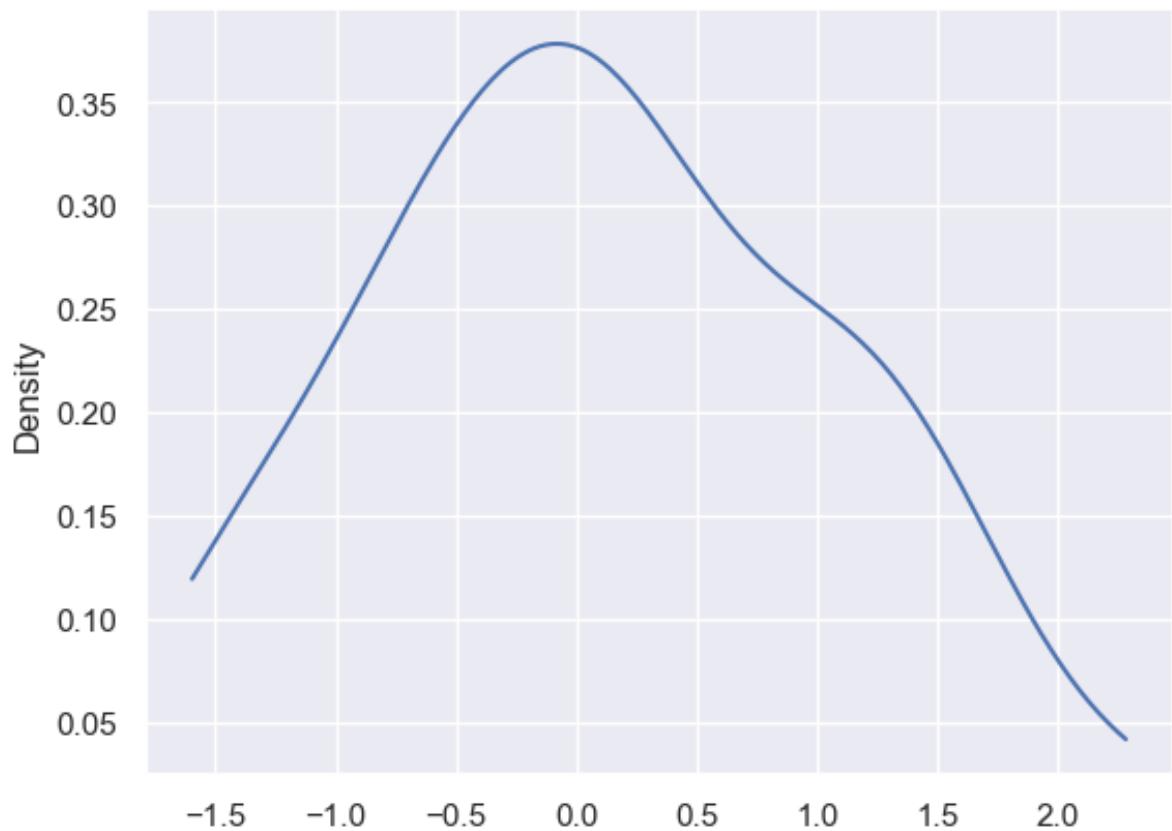
Построим график плотности по вертикальной оси:

```
In [105]: ax = sns.kdeplot(y=x)
```



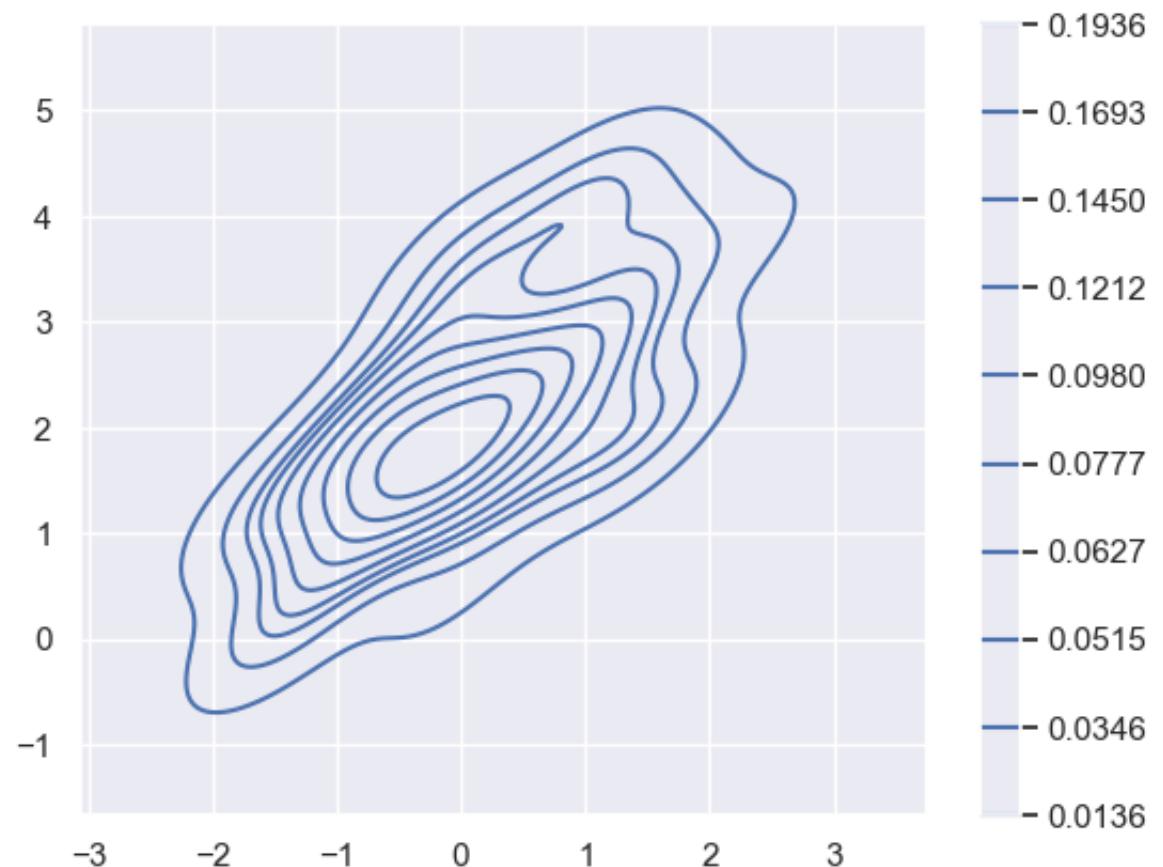
Ограничим кривую плотности в пределах диапазона данных:

```
In [106]: ax = sns.kdeplot(x, cut=0)
```



Добавим цветовую шкалу для контуров:

```
In [107]: ax = sns.kdeplot(x=x, y=y, cbar=True)
```



2. Двумерные распределения

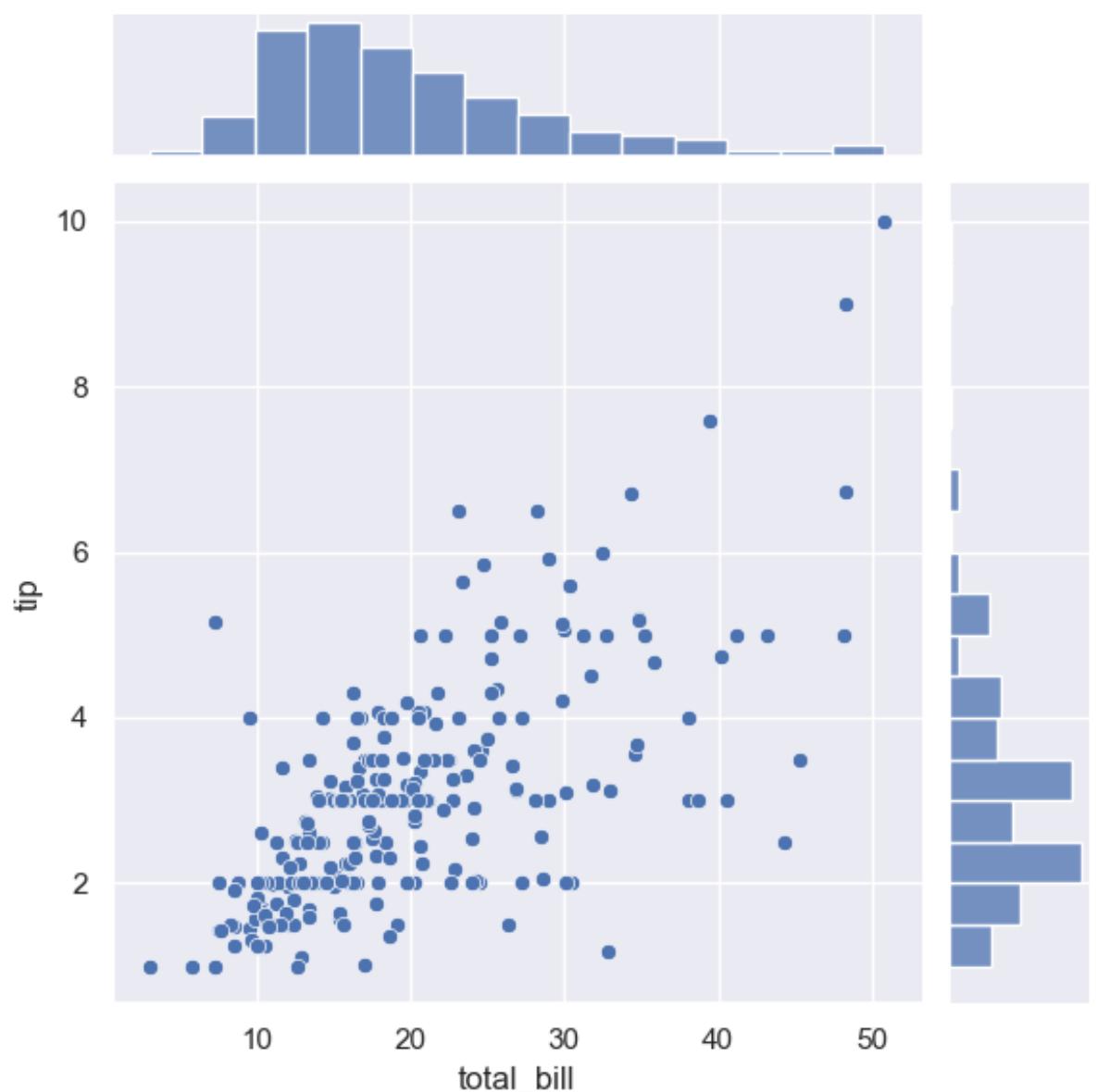
In [108]: `tips.head()`

Out[108]:

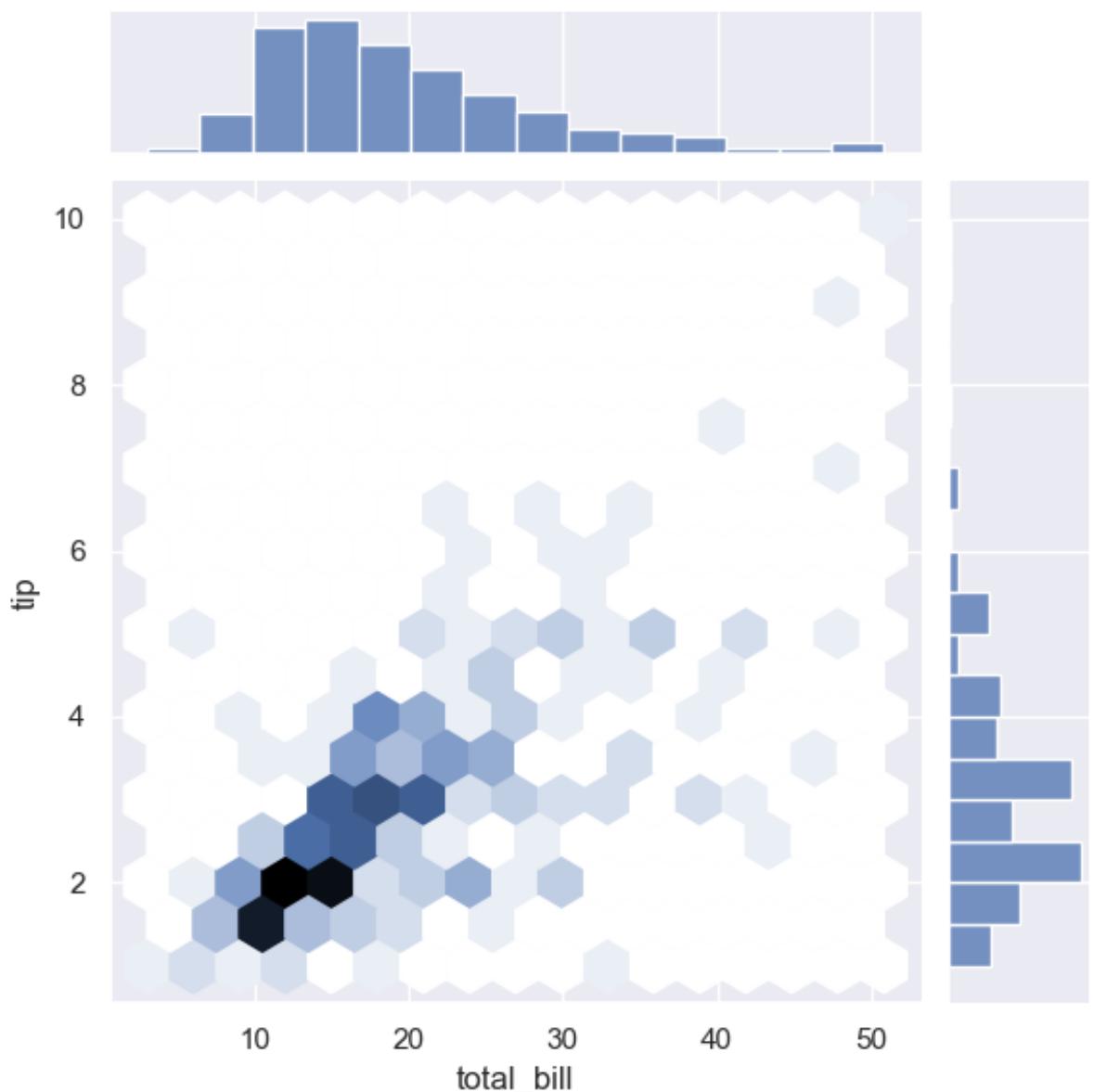
	total_bill	tip	sex	smoker	day	time	size	
0	16.99	1.01	Female		No	Sun	Dinner	2
1	10.34	1.66	Male		No	Sun	Dinner	3
2	21.01	3.50	Male		No	Sun	Dinner	3
3	23.68	3.31	Male		No	Sun	Dinner	2
4	24.59	3.61	Female		No	Sun	Dinner	4

In [109]: `x= tips['total_bill']`
`y= tips['tip']`

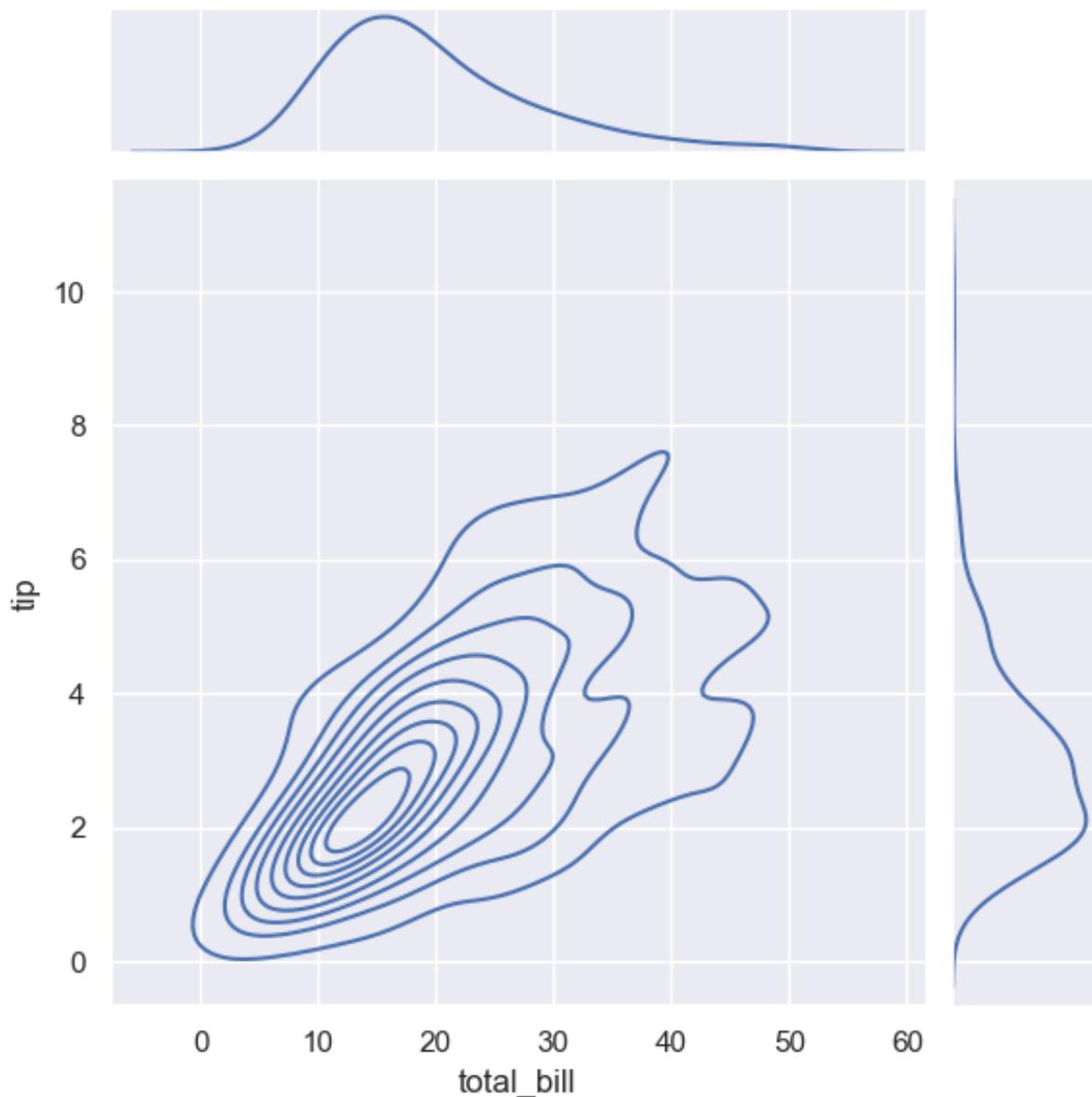
```
In [110]: sns.jointplot(x= x, y= y);
```



```
In [111]: sns.jointplot(x= x, y= y, kind= 'hex');
```

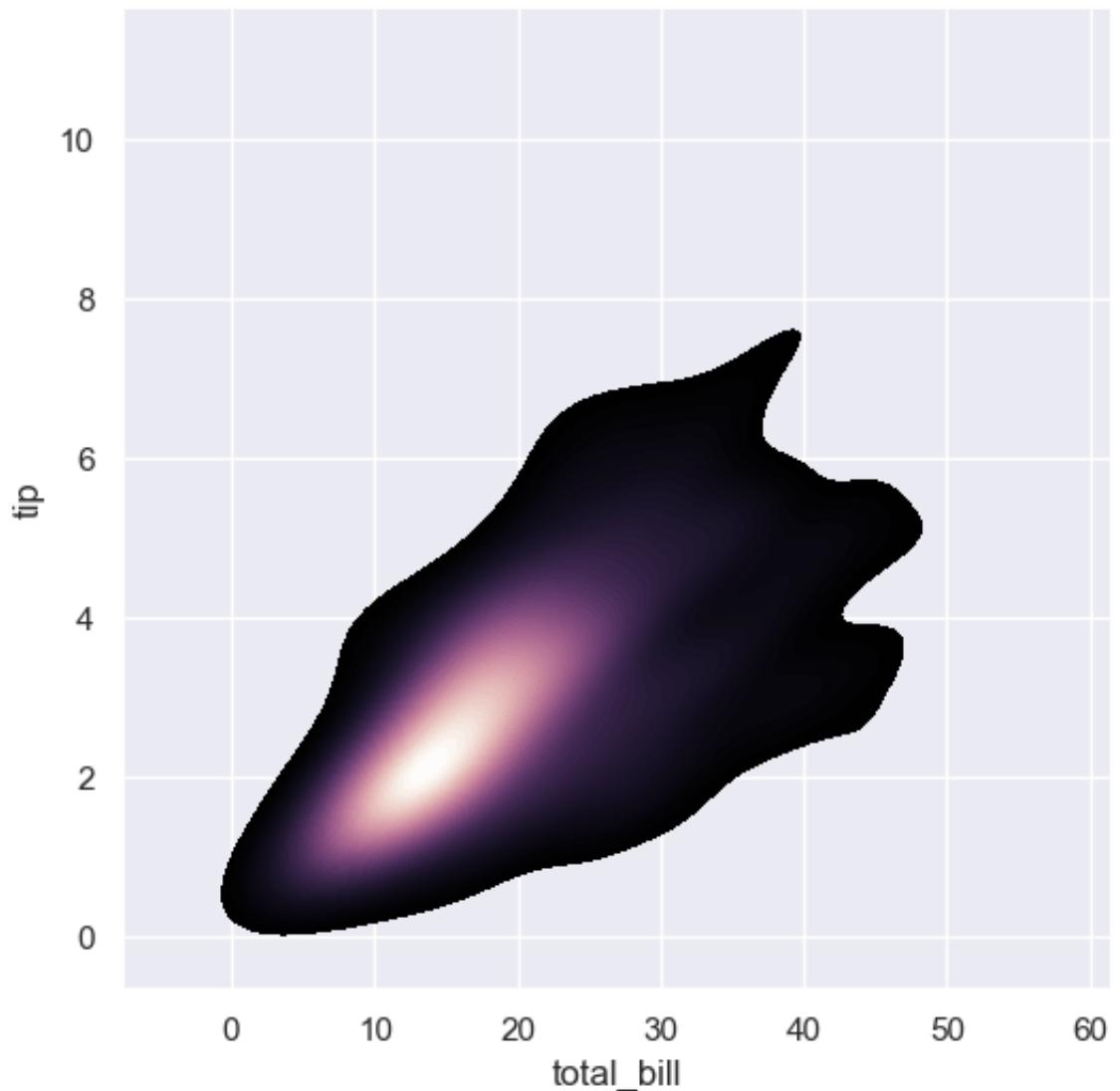


```
In [112]: sns.jointplot(x= x, y= y, kind= 'kde');
```

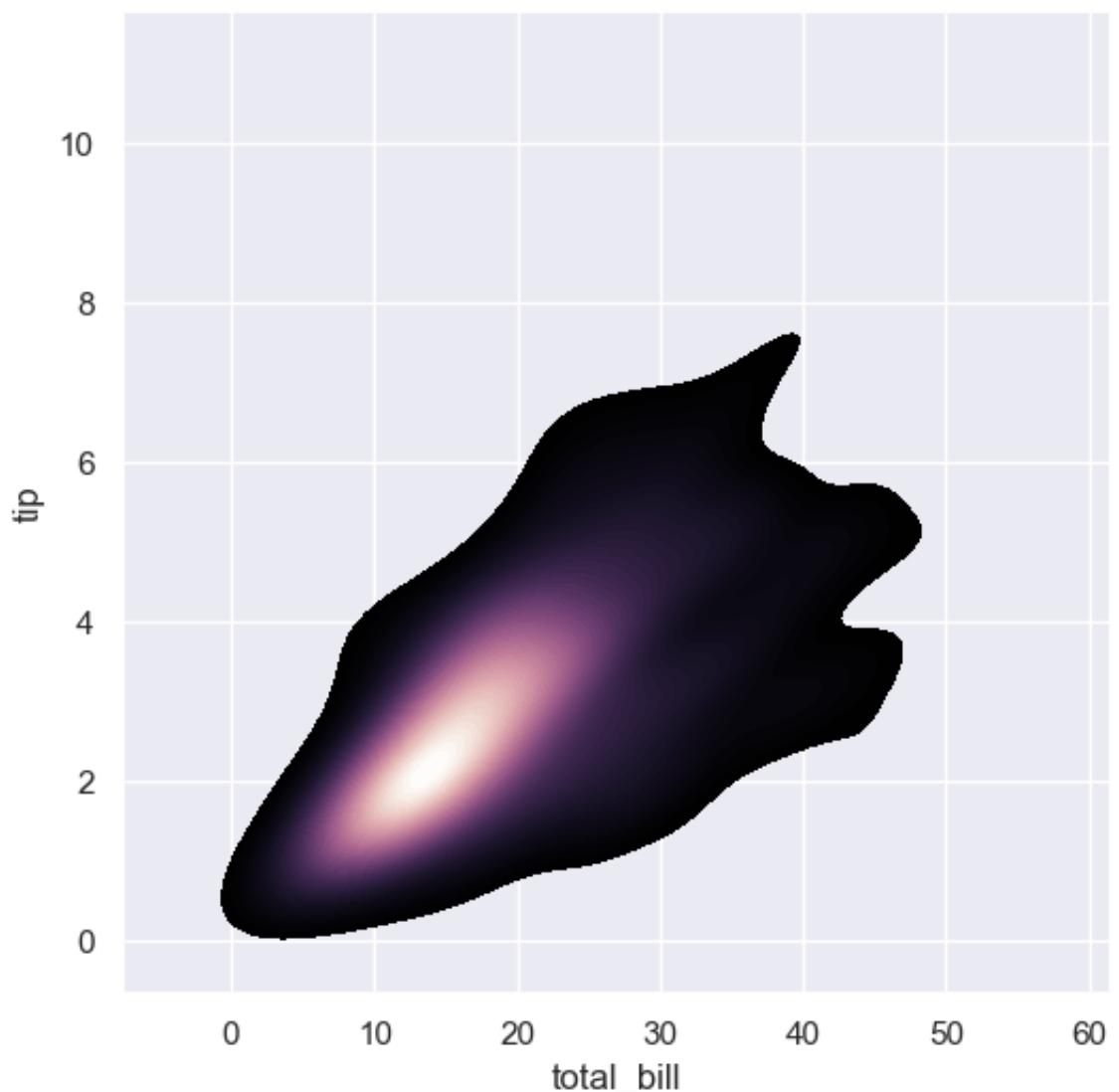


Этот код создаёт цветовую карту с линейно уменьшающейся (или увеличивающейся) яркостью. Это означает, что информация будет сохранена при печати в чёрно-белом формате или при просмотре дальтоником.

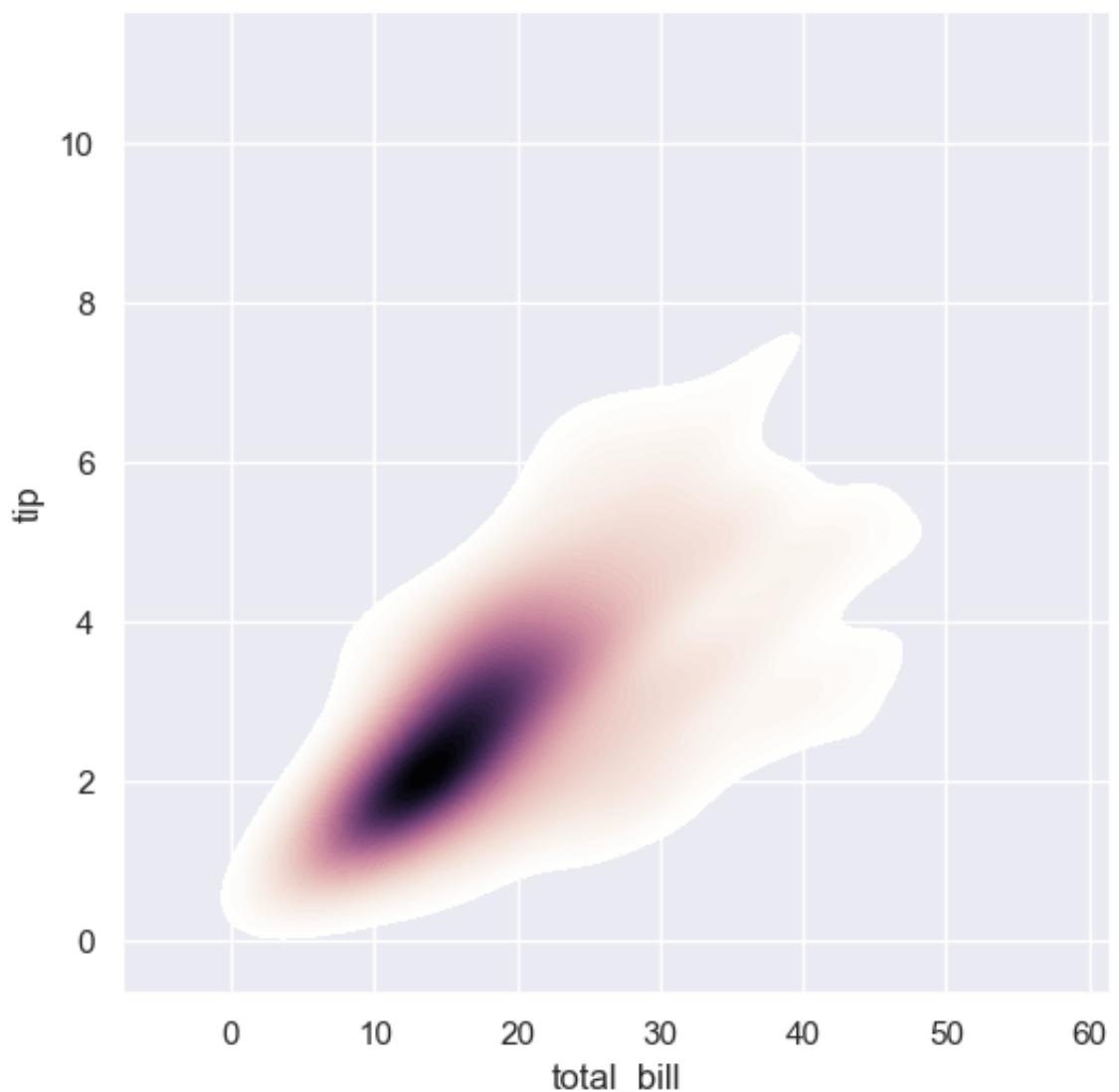
```
In [113]: f, ax= plt.subplots(figsize= (6,6))
cmap= sns.cubehelix_palette(light=1, dark= 0, reverse= True, as_cmap=True)
sns.kdeplot(x=x, y=y, cmap= cmap, n_levels= 60, fill= True);
```



```
In [114]: f, ax= plt.subplots(figsize= (6,6))
cmap= sns.cubehelix_palette(start= 0, rot= 0.4, gamma= 1.0, hue= 0.8,
                             light=0, dark= 1, reverse= False, as_cmap= True)
sns.kdeplot(x=x, y=y, cmap= cmap, n_levels= 60, fill= True);
```



```
In [115]: f, ax= plt.subplots(figsize= (6,6))
cmap= sns.cubehelix_palette(start= 0, rot= 0.4, gamma= 1.0, hue= 0.8,
                             light=1, dark= 0, reverse= False, as_cmap= True)
sns.kdeplot(x=x, y=y, cmap= cmap, n_levels= 60, fill= True);
```



Параметры:

n_colors : int Количество цветов в палитре.

start : float, $0 \leq start \leq 3$ Оттенок в начале спирали.

rot : float Обороты по цветовому кругу в диапазоне палитры.

gamma : float $0 \leq gamma$ Гамма-фактор для выделения более тёмных ($gamma < 1$) или более светлых ($gamma > 1$) цветов.

hue : float, $0 \leq hue \leq 1$ Насыщенность цветов.

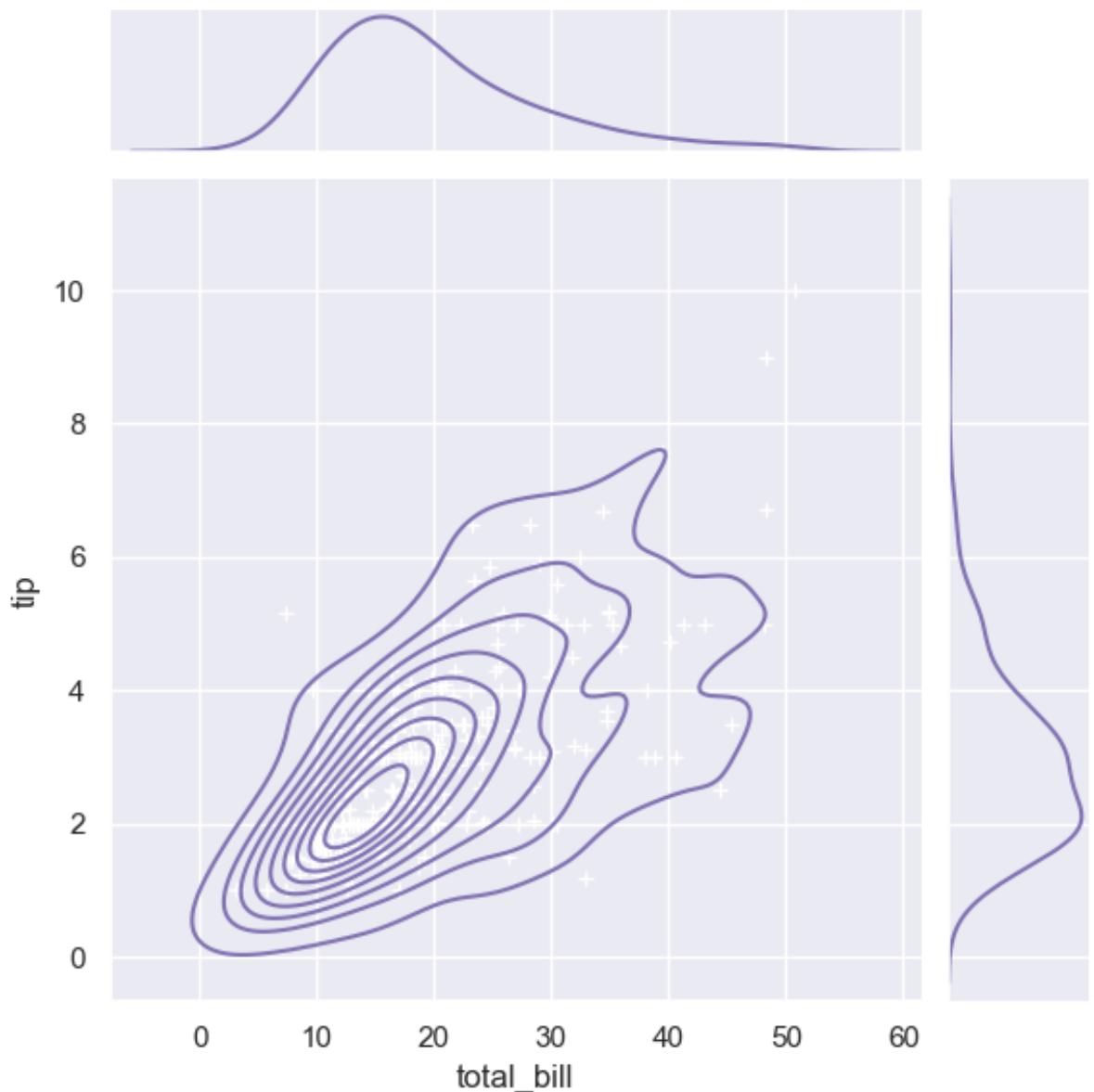
dark : float $0 \leq dark \leq 1$ Интенсивность самого тёмного цвета в палитре.

light : float $0 \leq light \leq 1$ Интенсивность самого светлого цвета в палитре.

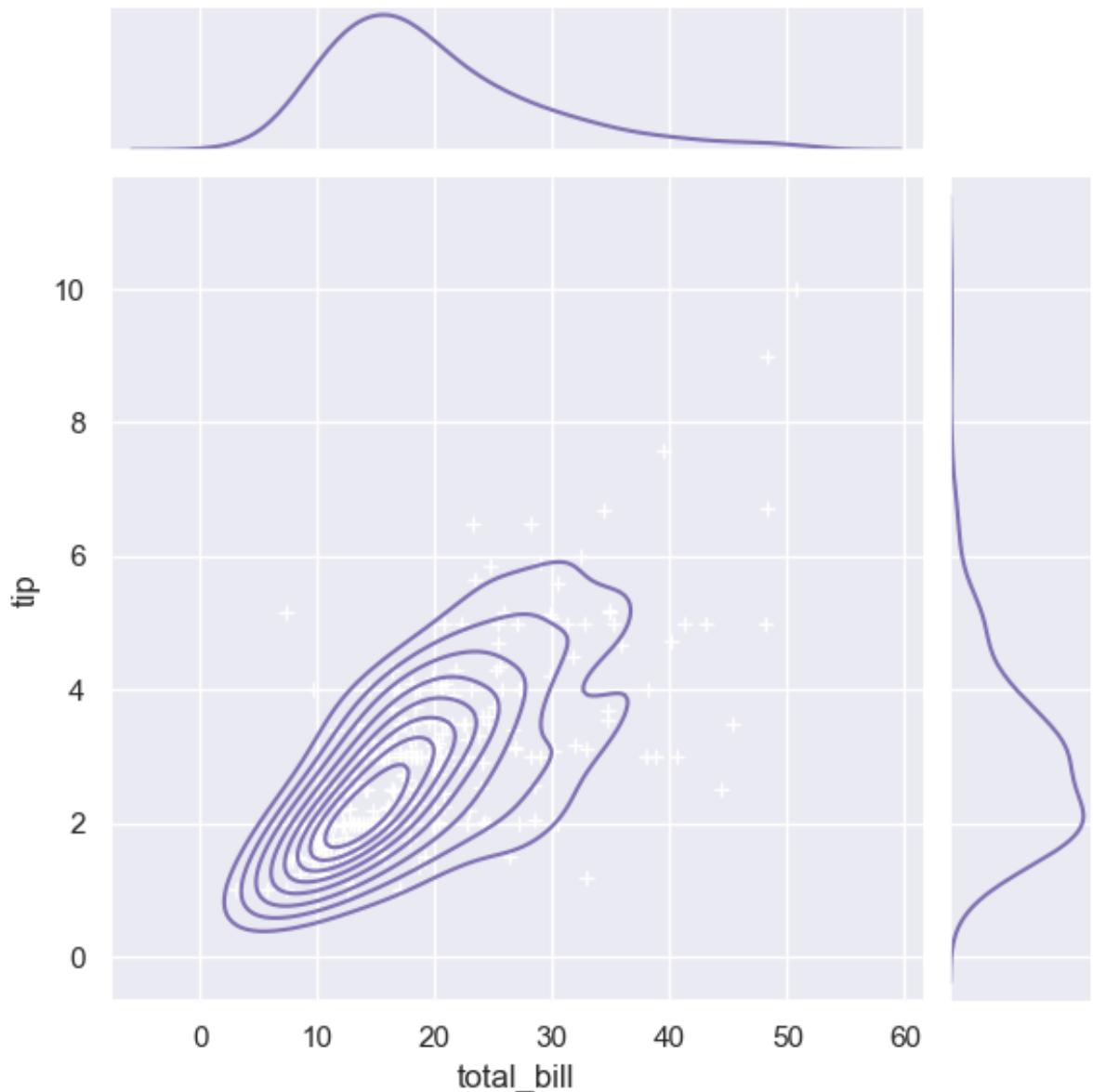
reverse : bool Если True, палитра будет меняться от тёмного к светлому.

as_cmap : bool Если True, возвращает цветовую карту Matplotlib вместо списка цветов.

```
In [116]: g= sns.jointplot(x= x, y= y, kind= 'kde', color='m')
g.plot_joint(plt.scatter, c= 'w', s= 30, linewidth= 1, marker= '+')
#g.ax_joint.collections[0].set_alpha(0)
```



```
In [117]: g= sns.jointplot(x= x, y= y, kind= 'kde', color='m')
g.plot_joint(plt.scatter, c= 'w', s= 30, linewidth= 1, marker= '+')
g.ax_joint.collections[0].set_alpha(0)
```

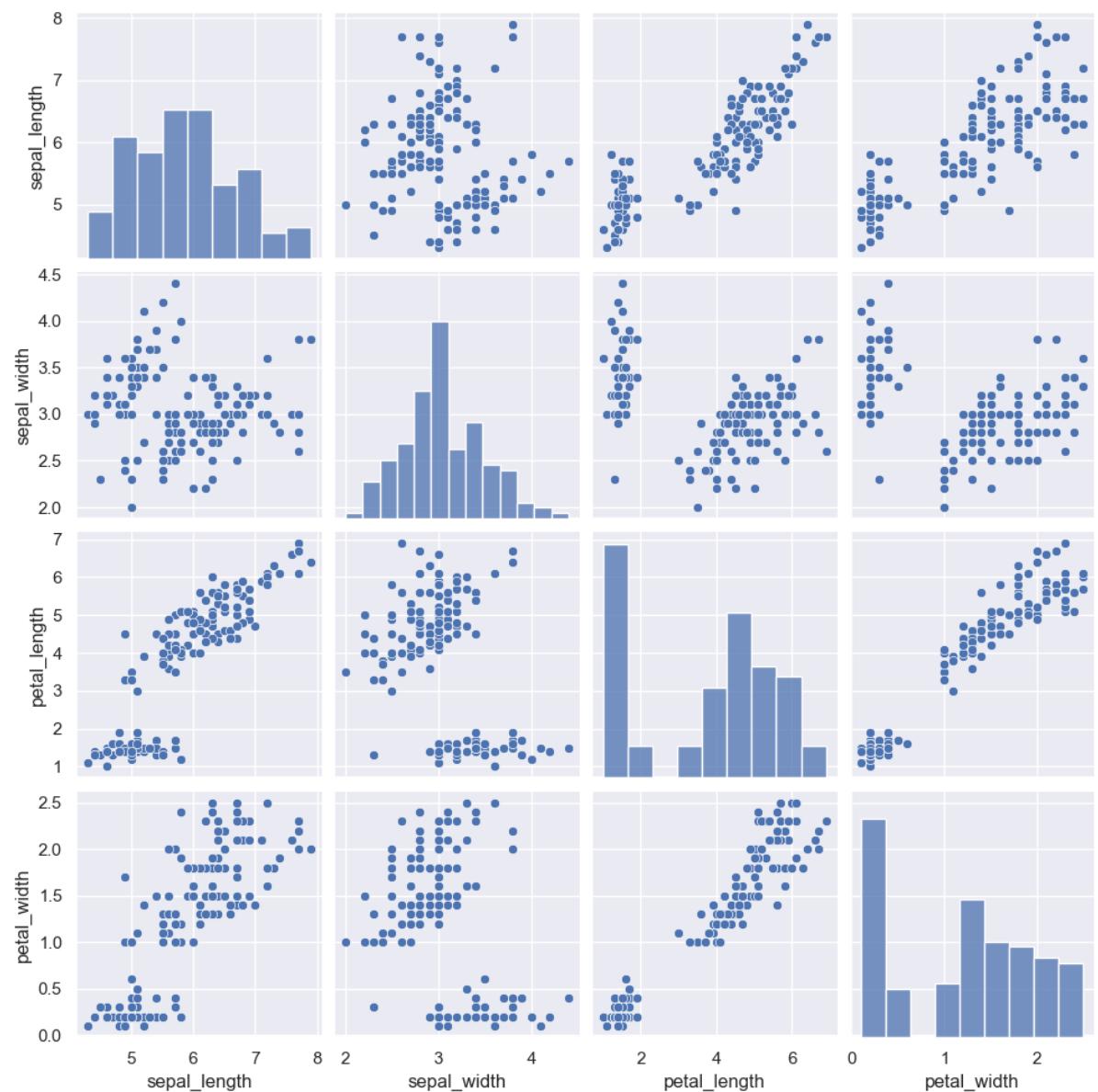


```
In [118]: iris.head()
```

Out[118]:

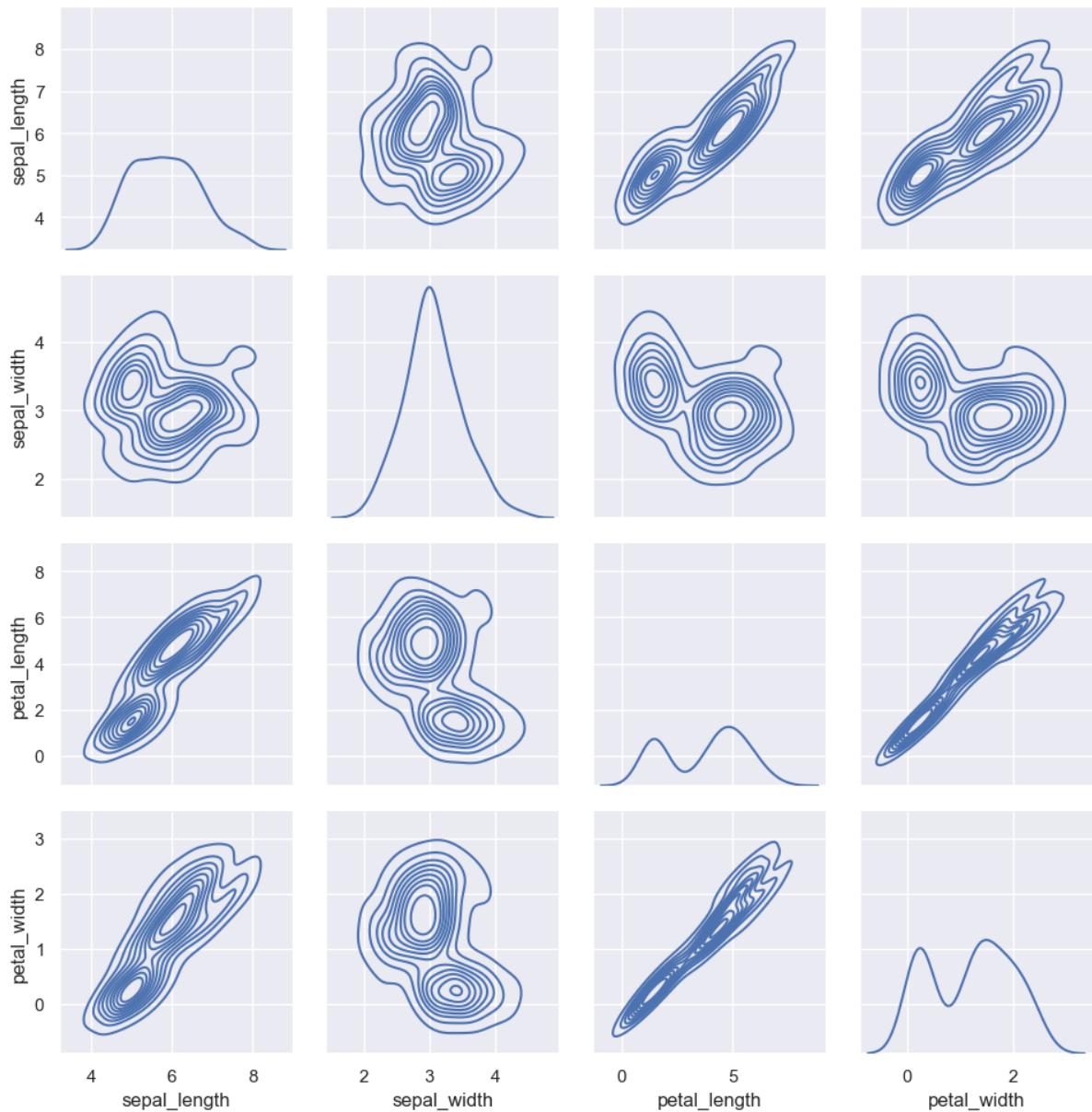
	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

```
In [119]: sns.pairplot(iris);
```

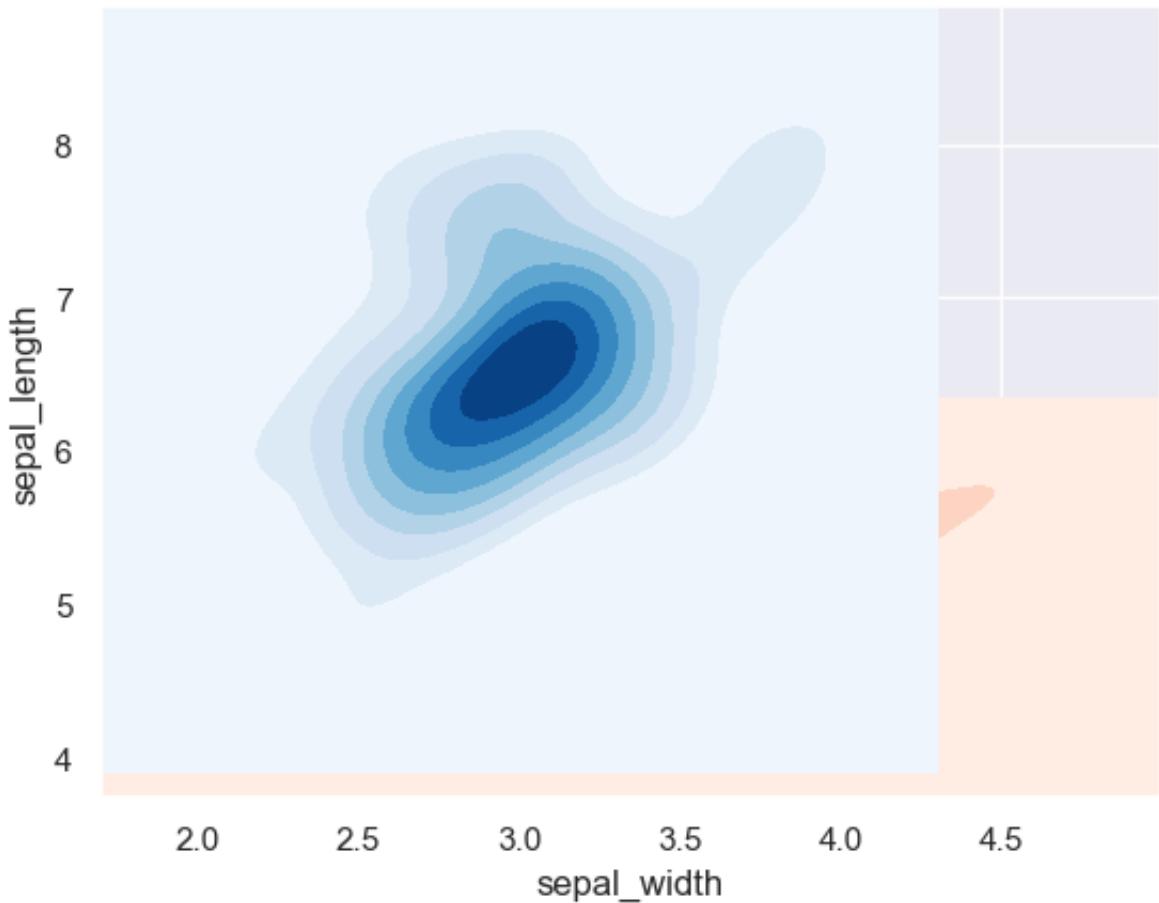


Функция `pairplot` отображает одномерный график данных по диагонали как `histplot`, а двумерный график данных по недиагонали как `scatterplot`.

```
In [120]: g= sns.PairGrid(iris)
g.map_diag(sns.kdeplot)
g.map_offdiag(sns.kdeplot, n_levels= 10);
```



```
In [121]: setosa = iris.loc[iris.species == "setosa"]
virginica = iris.loc[iris.species == "virginica"]
ax = sns.kdeplot(x=setosa.sepal_width, y=setosa.sepal_length,
                  cmap="Reds", fill=True, thresh=False)
ax = sns.kdeplot(x=virginica.sepal_width, y=virginica.sepal_length,
                  cmap="Blues", fill=True, thresh=False)
```



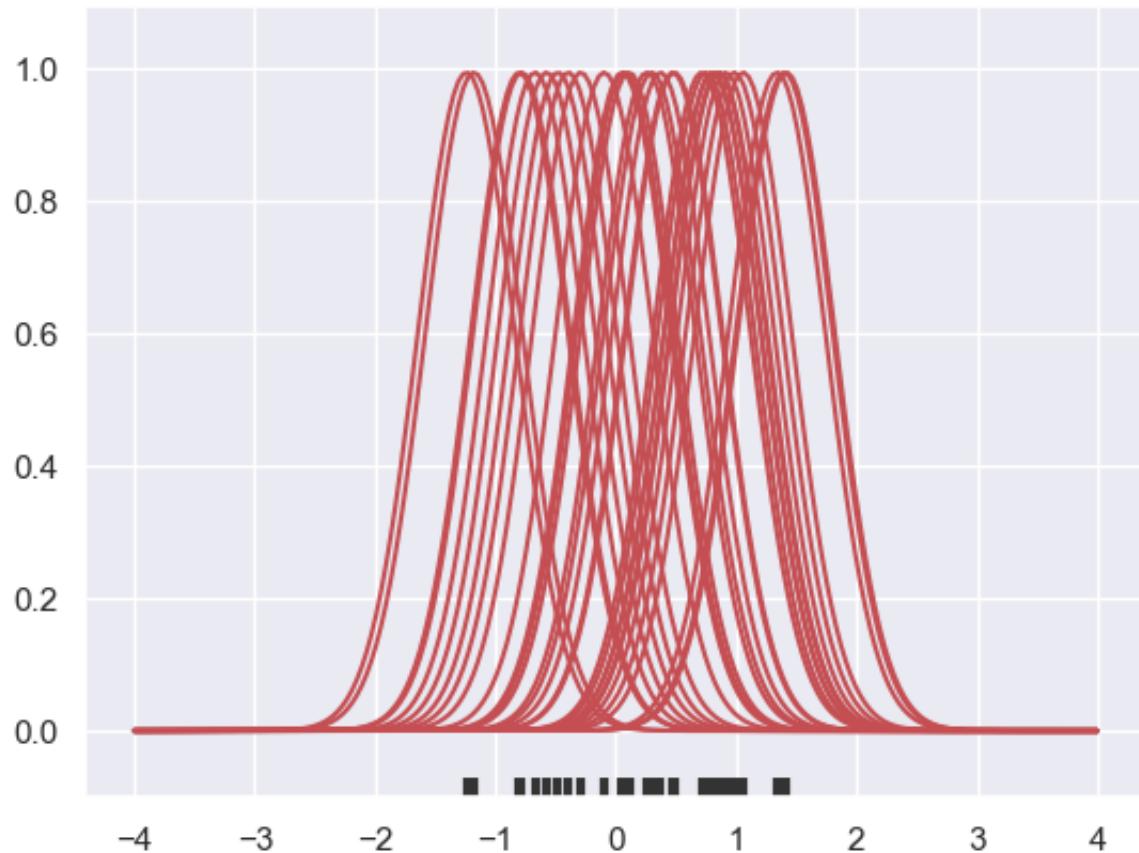
Построение KDE требует больше вычислительных затрат, чем построение гистограммы. Сначала каждое наблюдение заменяется нормальной (гауссовой) кривой с центром в этом значении:

```
In [122]: from scipy import stats
x = np.random.normal(0, 1, size=30)
bandwidth = 1.06 * x.std() * x.size ** (-1 / 5.)
support = np.linspace(-4, 4, 200)

kernels = []
for x_i in x:

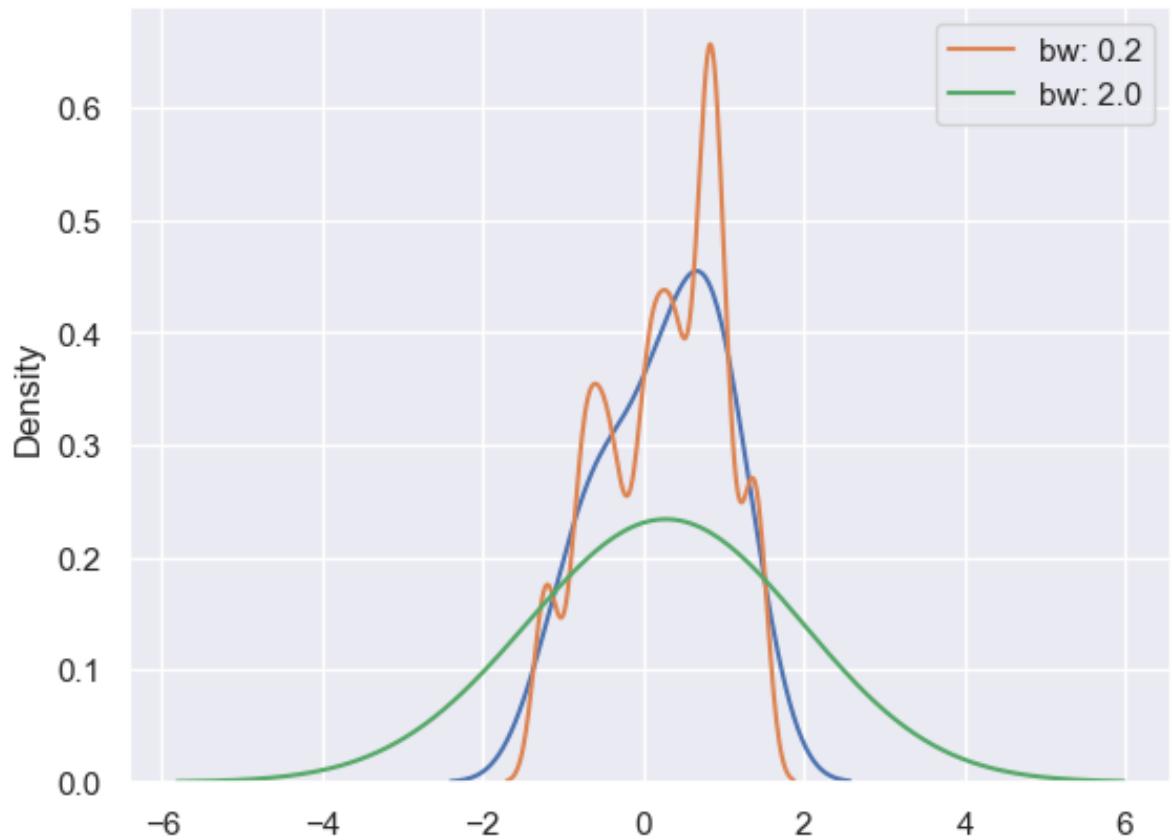
    kernel = stats.norm(x_i, bandwidth).pdf(support)
    kernels.append(kernel)
    plt.plot(support, kernel, color="r")

sns.rugplot(x, color=".2", linewidth=3);
```



Параметр пропускной способности (bw) KDE управляет точностью соответствия оценки данным, подобно размеру ячейки в гистограмме. Он соответствует ширине ядер, которые мы построили выше. По умолчанию алгоритм пытается подобрать подходящее значение, используя общее эталонное правило, но может быть полезно попробовать большие или меньшие значения:

```
In [123]: sns.kdeplot(x)
sns.kdeplot(x, bw_method=.2, label="bw: 0.2")
sns.kdeplot(x, bw_method=2., label="bw: 2.0")
plt.legend();
```



Линейная регрессия

- regplot()
- lmplot()

Функции `regplot()` и `lmplot()` тесно связаны, но первая — это функция уровня области рисования (`axes`), а вторая — функция уровня рисунка (`figure`), объединяющая `regplot()` и `FacetGrid`.

Графики регрессии в Seaborn в первую очередь предназначены для визуального представления набора данных, помогающего выявлять закономерности в ходе разведочного анализа данных.

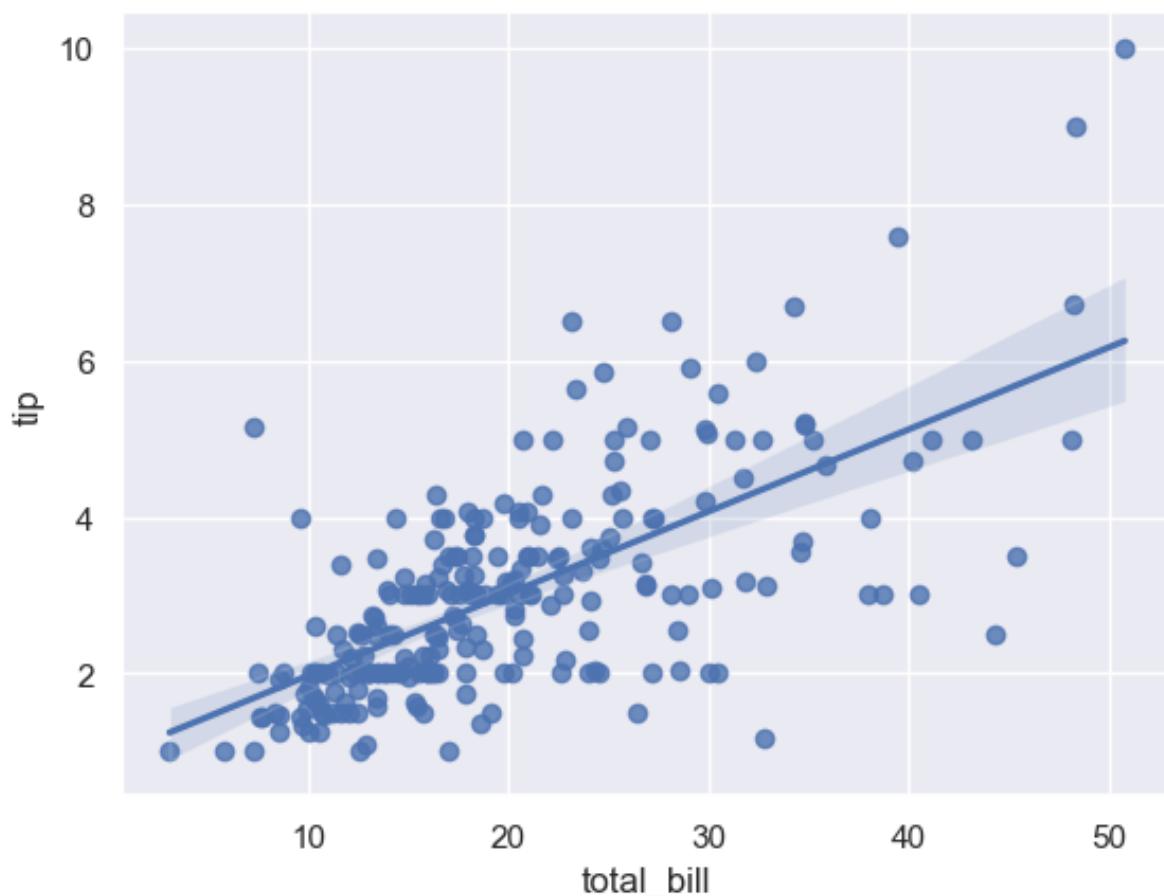
```
In [124]: sns.set(color_codes=True)
```

```
In [125]: tips.head()
```

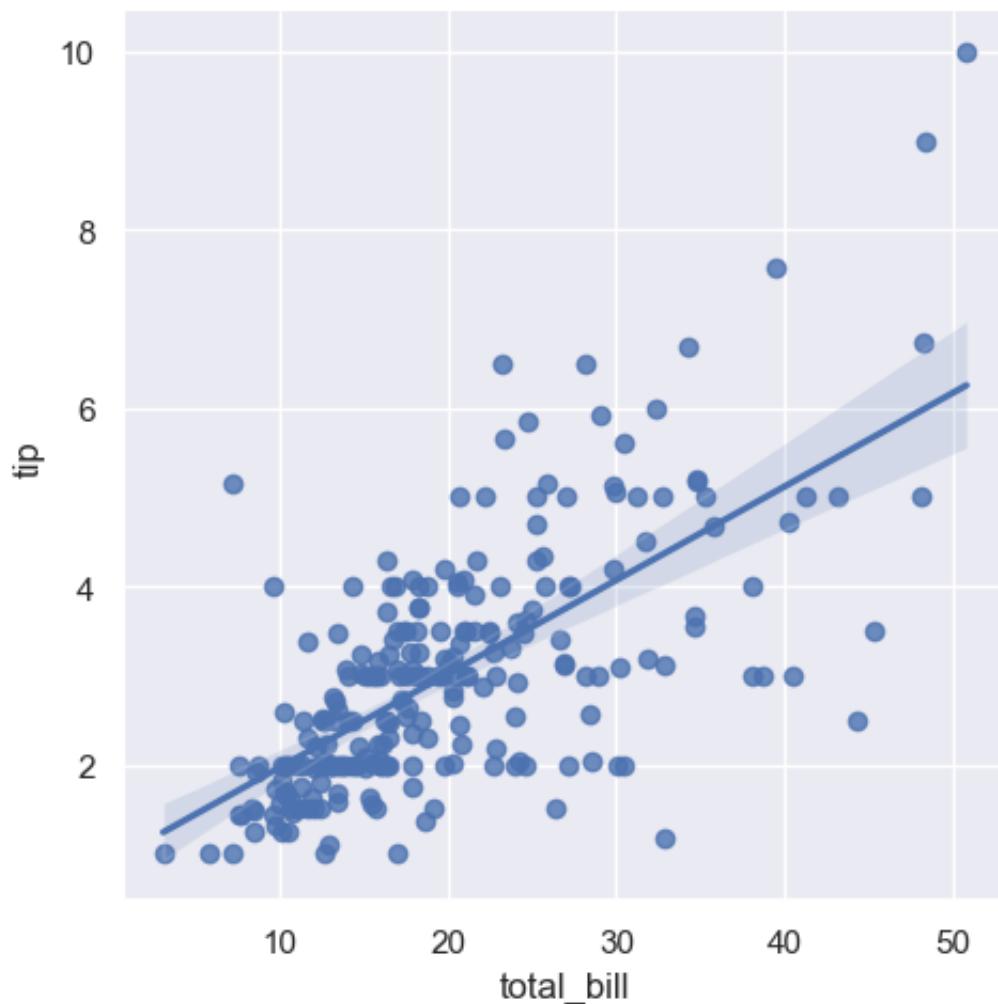
Out[125]:

	total_bill	tip	sex	smoker	day	time	size	
0	16.99	1.01	Female		No	Sun	Dinner	2
1	10.34	1.66	Male		No	Sun	Dinner	3
2	21.01	3.50	Male		No	Sun	Dinner	3
3	23.68	3.31	Male		No	Sun	Dinner	2
4	24.59	3.61	Female		No	Sun	Dinner	4

```
In [126]: sns.regplot(x="total_bill", y="tip", data=tips);
```

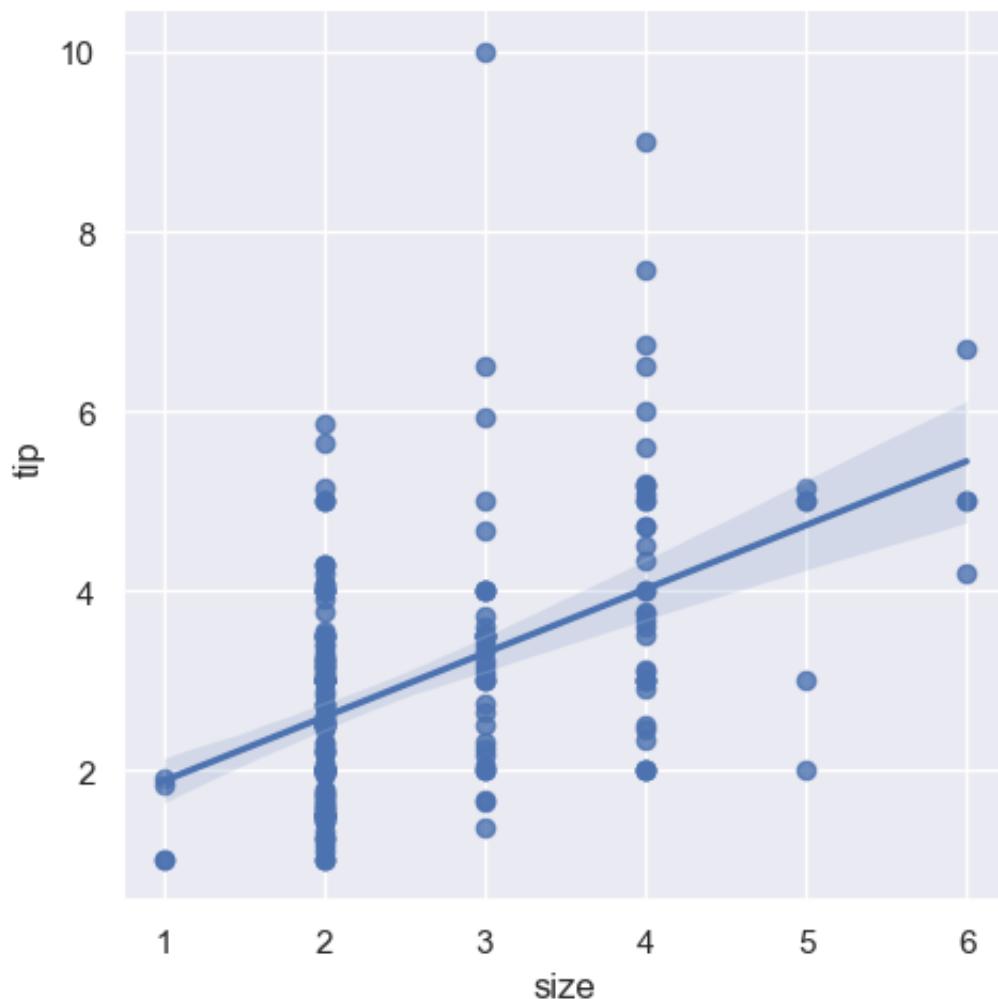


```
In [127]: sns.lmplot(x="total_bill", y="tip", data=tips);
```



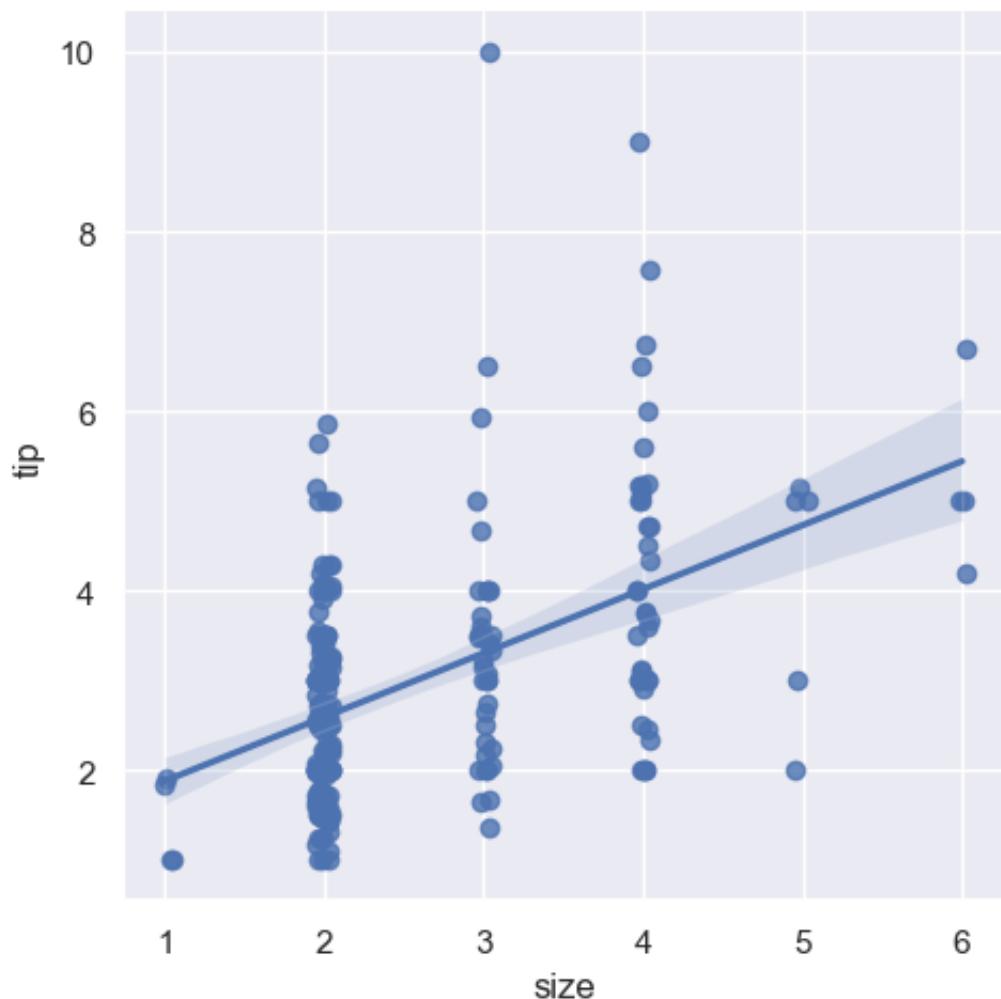
Можно построить линейную регрессию, когда одна из переменных принимает дискретные значения, однако простая диаграмма рассеяния, полученная с помощью такого набора данных, часто не является оптимальной:

```
In [128]: sns.lmplot(x="size", y="tip", data=tips);
```



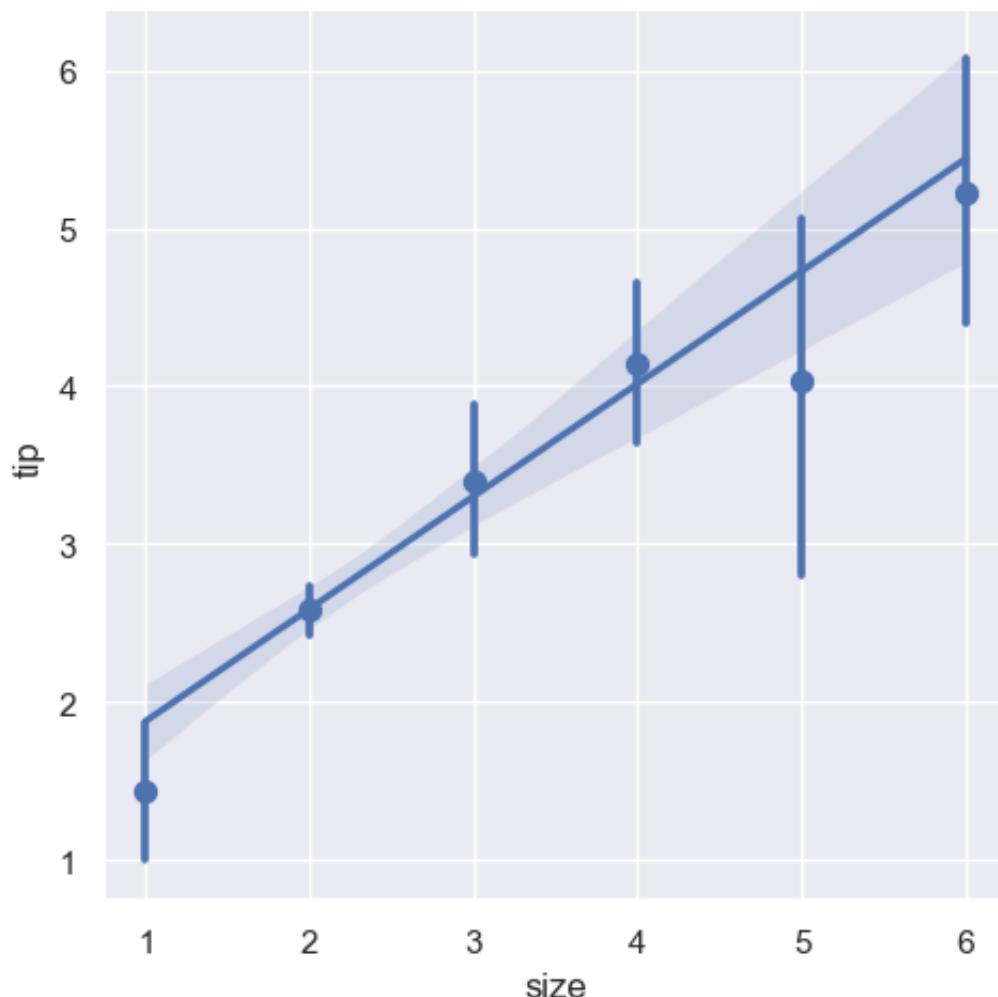
Один из вариантов — добавить случайный шум («джиттер») к дискретным значениям, чтобы сделать их распределение более чётким. Обратите внимание, что джиттер применяется только к данным диаграммы рассеяния и не влияет на аппроксимацию линии регрессии:

```
In [129]: sns.lmplot(x="size", y="tip", data=tips, x_jitter=.05);
```



Второй вариант — свернуть наблюдения в каждой дискретной ячейке, чтобы построить оценку центральной тенденции вместе с доверительным интервалом:

```
In [130]: sns.lmplot(x="size", y="tip", data=tips, x_estimator=np.mean);
```

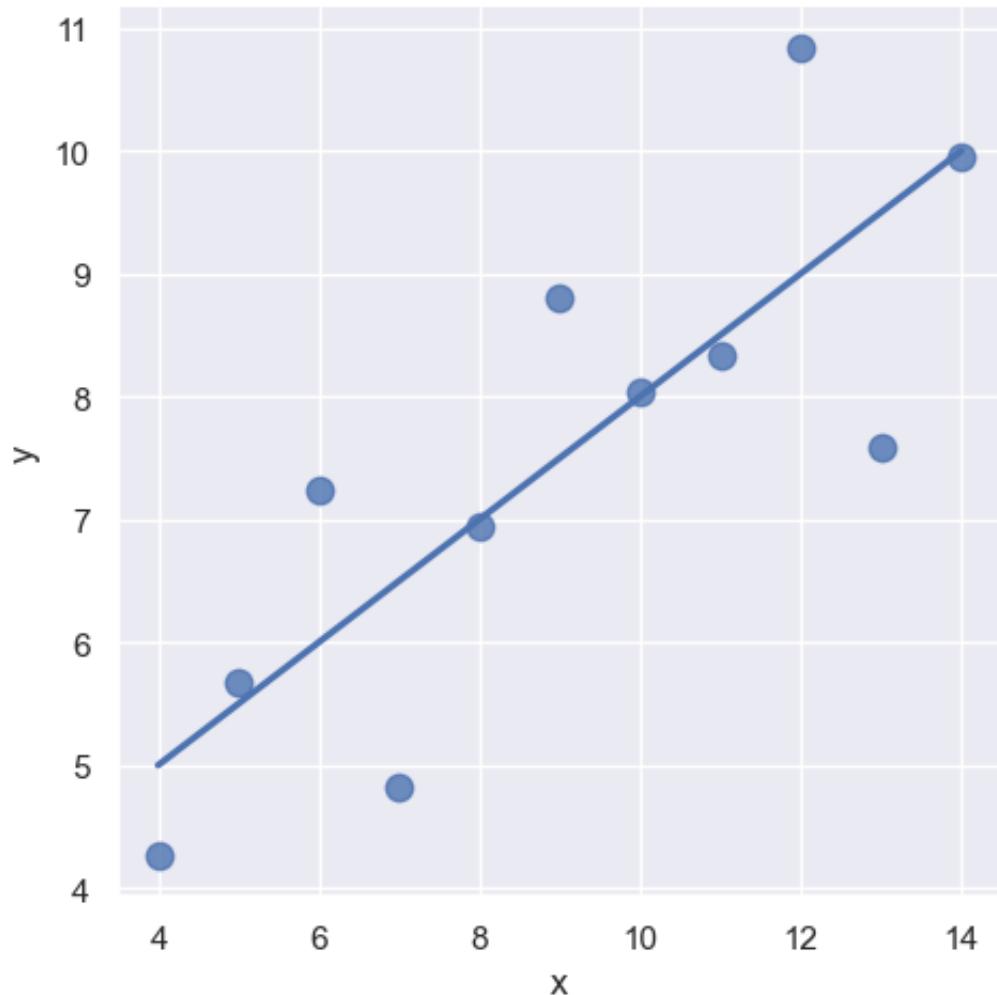


Подгонка различных типов моделей

Использованная выше простая модель линейной регрессии очень проста в подгонке, однако она не подходит для некоторых типов наборов данных.

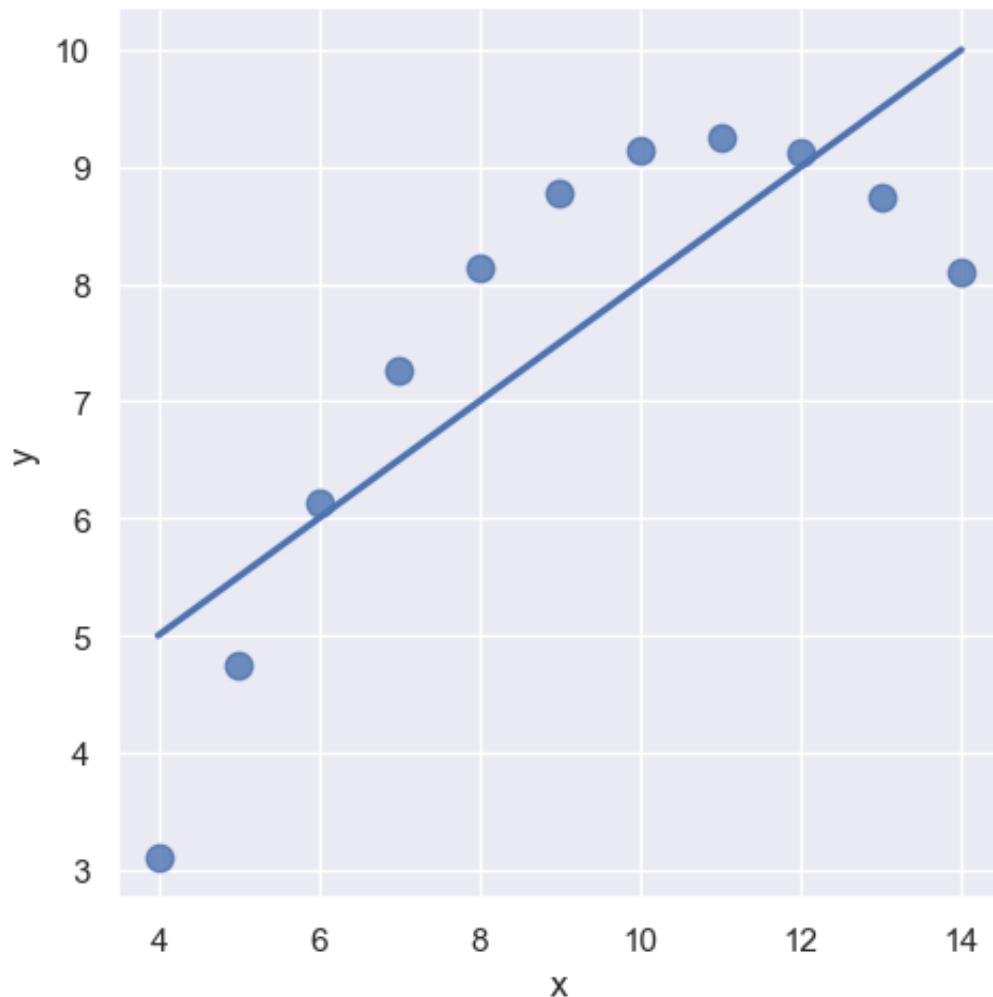
```
In [131]: anscombe = sns.load_dataset("anscombe")
```

```
In [132]: sns.lmplot(x="x", y="y", data=anscombe.query("dataset == 'I'"),
                    ci=None, scatter_kws={"s": 80});
```



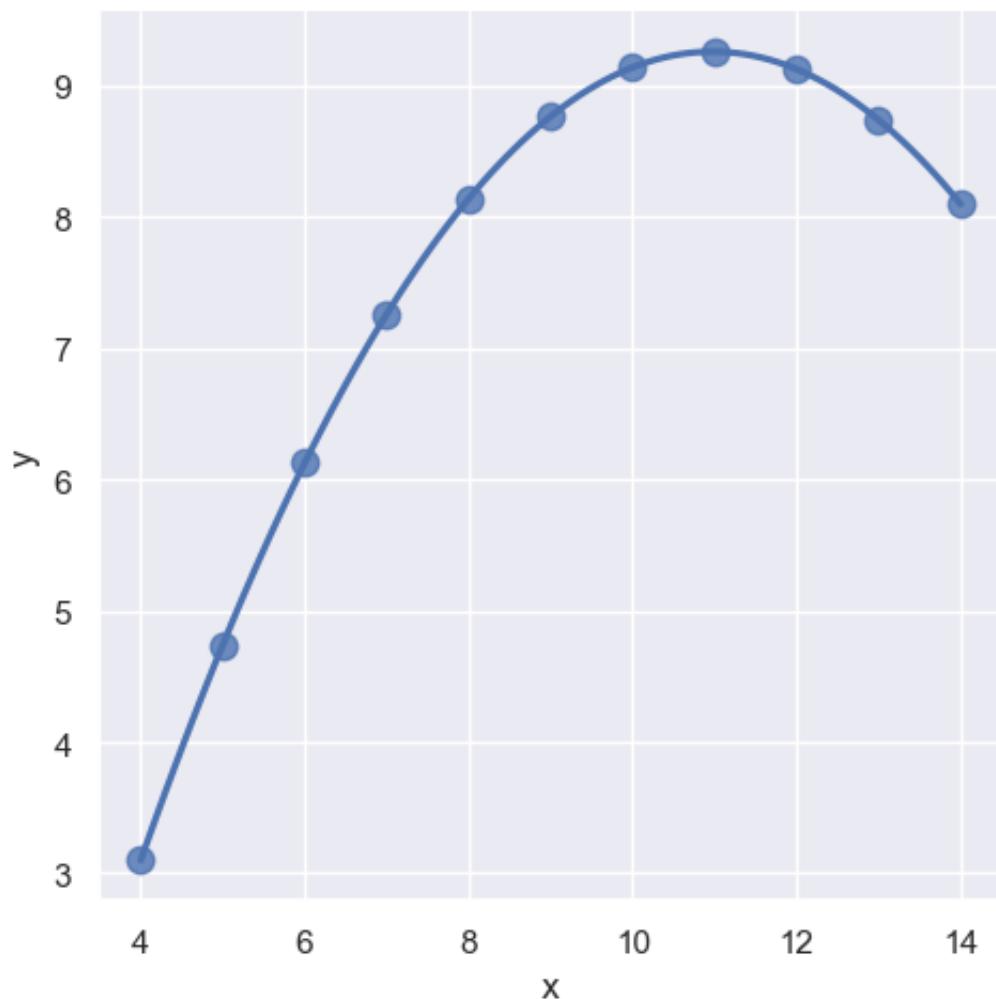
Линейная зависимость во втором наборе данных та же, но график ясно показывает, что это не очень хорошая модель:

```
In [133]: sns.lmplot(x="x", y="y", data=anscombe.query("dataset == 'II'"),
                    ci=None, scatter_kws={"s": 80});
```



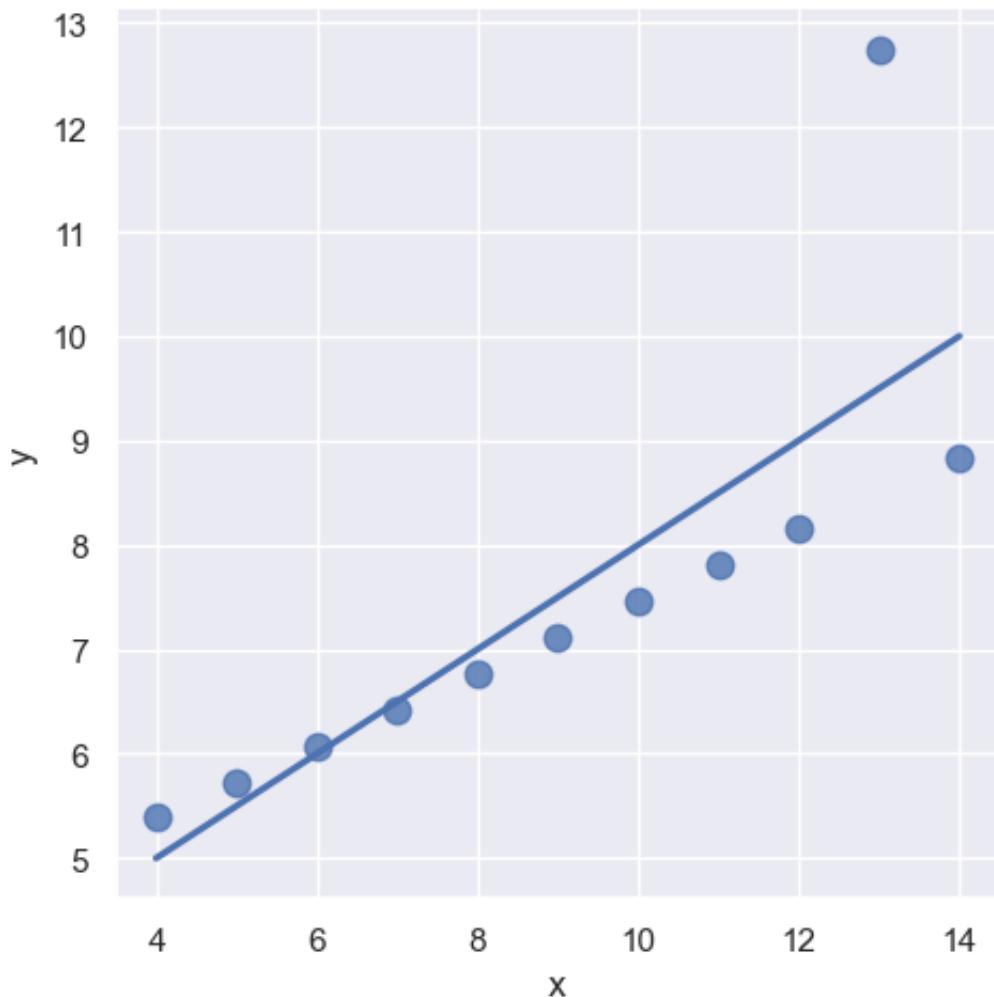
При наличии такого рода отношений более высокого порядка `lmplot()` и `regplot()` могут подогнать модель полиномиальной регрессии для исследования простых видов нелинейных тенденций в наборе данных:

```
In [134]: sns.lmplot(x="x", y="y", data=anscombe.query("dataset == 'II'"),
order=2, ci=None, scatter_kws={"s": 80});
```



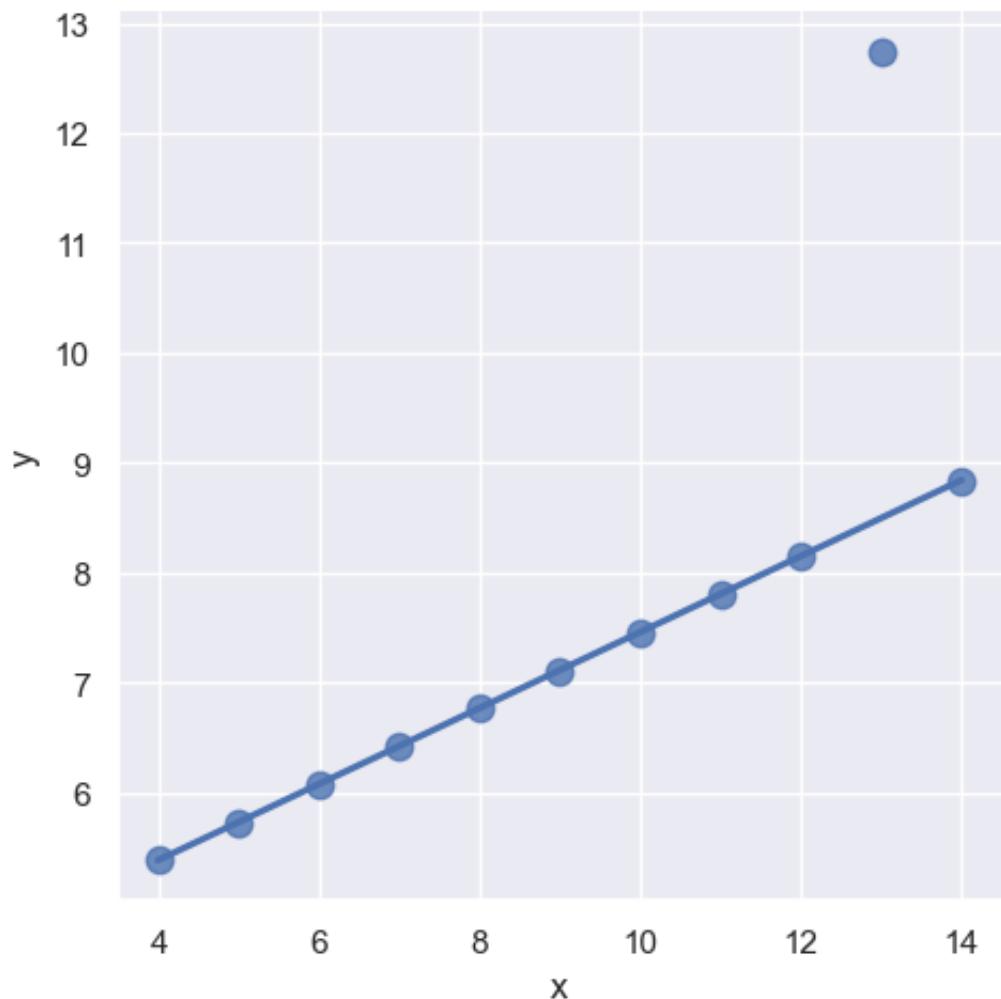
Другая проблема возникает из-за точек «выбросов», которые отклоняются по какой-то причине, не связанной с основной изучаемой зависимостью:

```
In [135]: sns.lmplot(x="x", y="y", data=anscombe.query("dataset == 'III'"),
                    ci=None, scatter_kws={"s": 80});
```



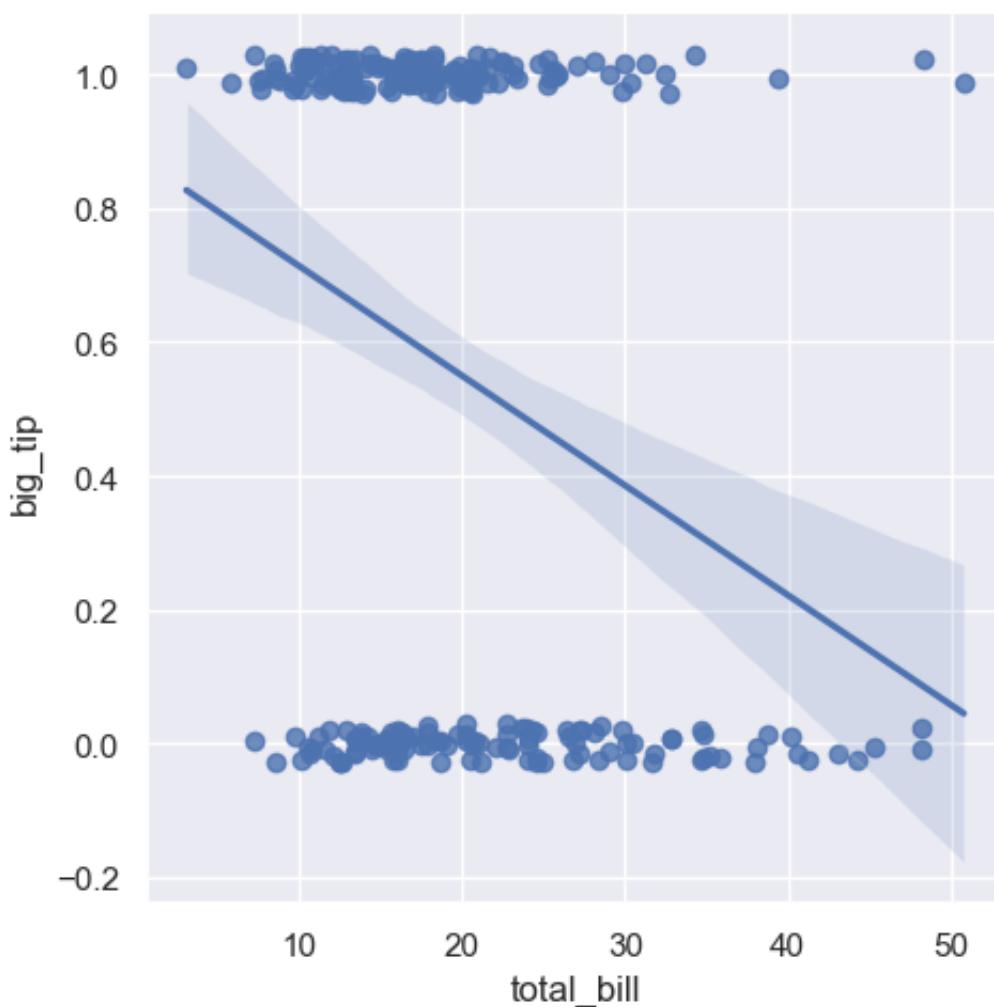
При наличии выбросов может быть полезно подобрать надежную регрессию, которая использует другую функцию потерь для снижения веса относительно больших остатков:

```
In [136]: # install statsmodels!
sns.lmplot(x="x", y="y", data=anscombe.query("dataset == 'III'"),
            robust=True, ci=None, scatter_kws={"s": 80});
```



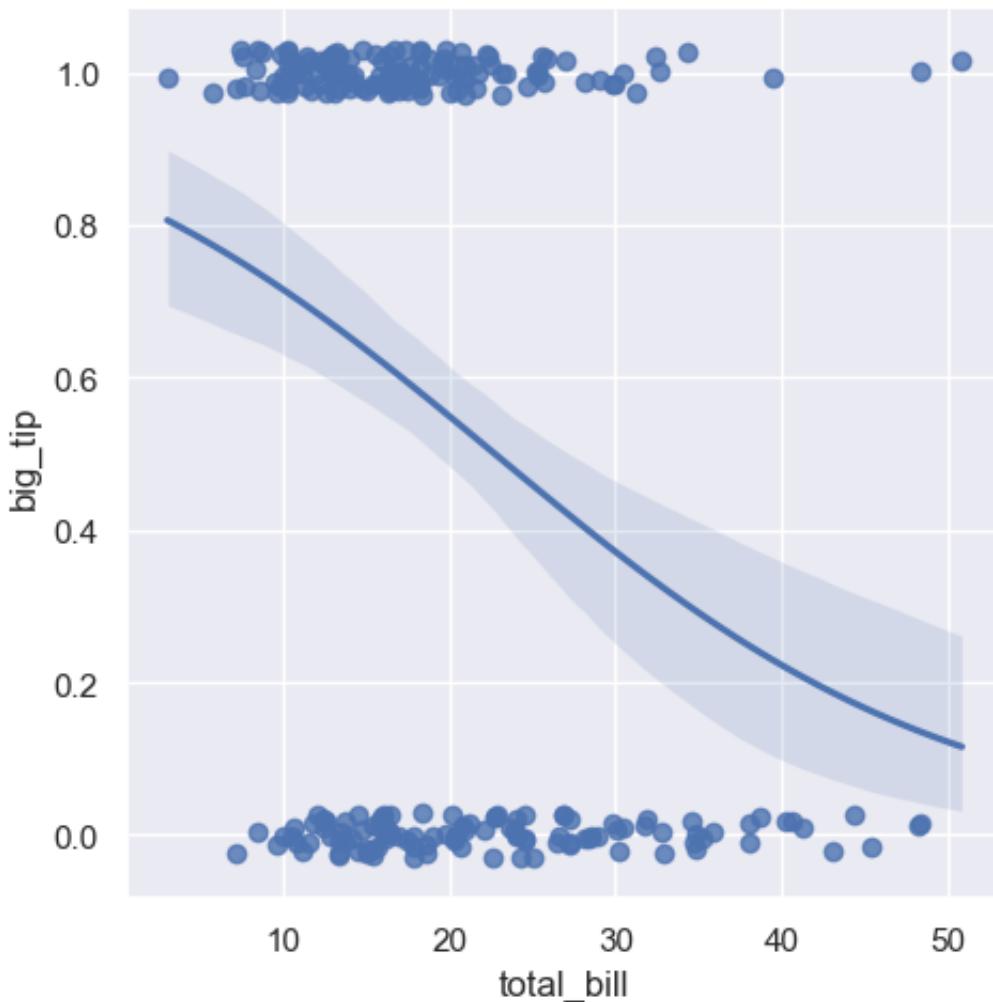
Когда переменная y является бинарной, простая линейная регрессия также «работает», но дает неправдоподобные прогнозы:

```
In [137]: tips["big_tip"] = (tips.tip / tips.total_bill) > .15
sns.lmplot(x="total_bill", y="big_tip", data=tips,
            y_jitter=.03);
```



Решением в этом случае является построение логистической регрессии таким образом, чтобы линия регрессии показывала предполагаемую вероятность $y = 1$ для заданного значения x :

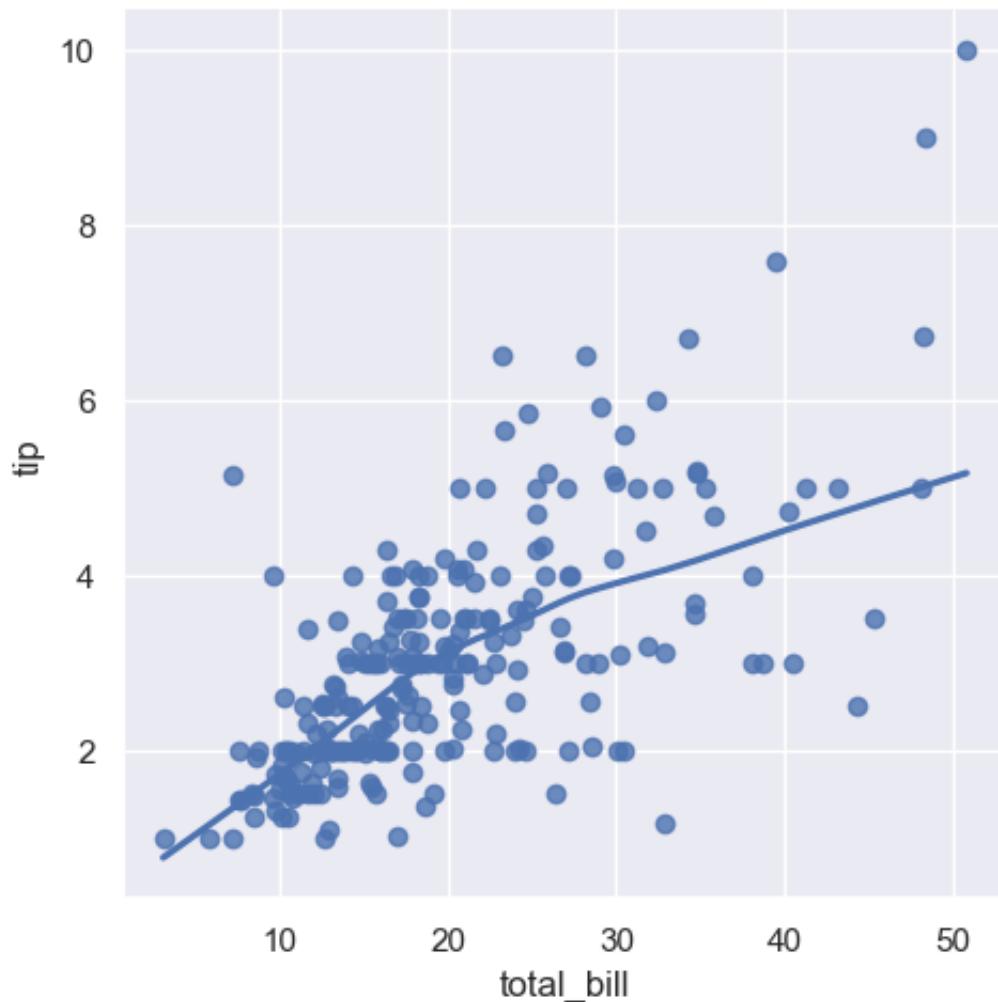
```
In [138]: sns.lmplot(x="total_bill", y="big_tip", data=tips,  
logistic=True, y_jitter=.03);
```



Обратите внимание, что оценка логистической регрессии требует значительно больше вычислительных ресурсов (это справедливо и для надежной регрессии), чем простая регрессия, и поскольку доверительный интервал вокруг линии регрессии вычисляется с использованием процедуры бутстрата, вы можете отключить ее для более быстрой итерации (используя ci=None).

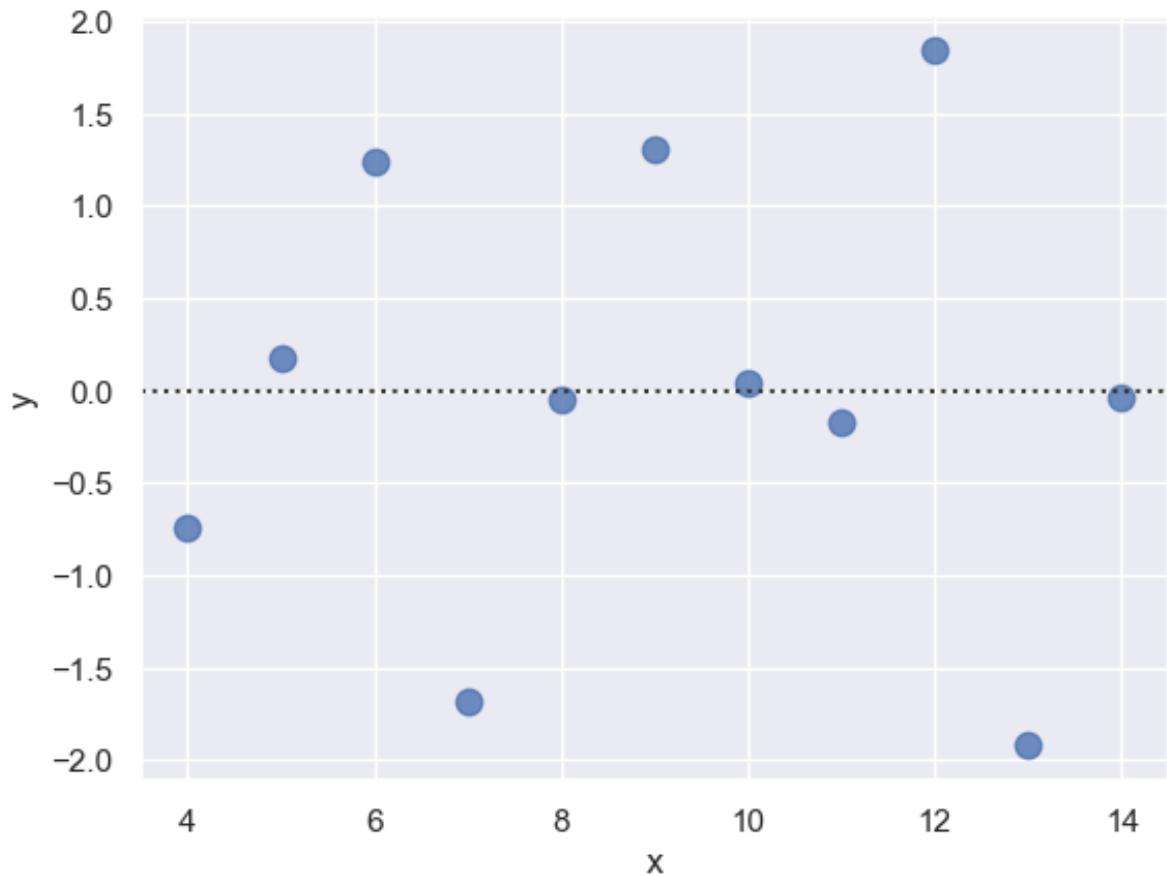
Совершенно иной подход заключается в подгонке непараметрической регрессии с использованием сглаживателя наименьшего порядка. Этот подход предполагает наименьшее количество допущений, хотя он требует значительных вычислительных ресурсов, поэтому в настоящее время доверительные интервалы вообще не рассчитываются:

```
In [139]: sns.lmplot(x="total_bill", y="tip", data=tips,  
                    lowess=True);
```



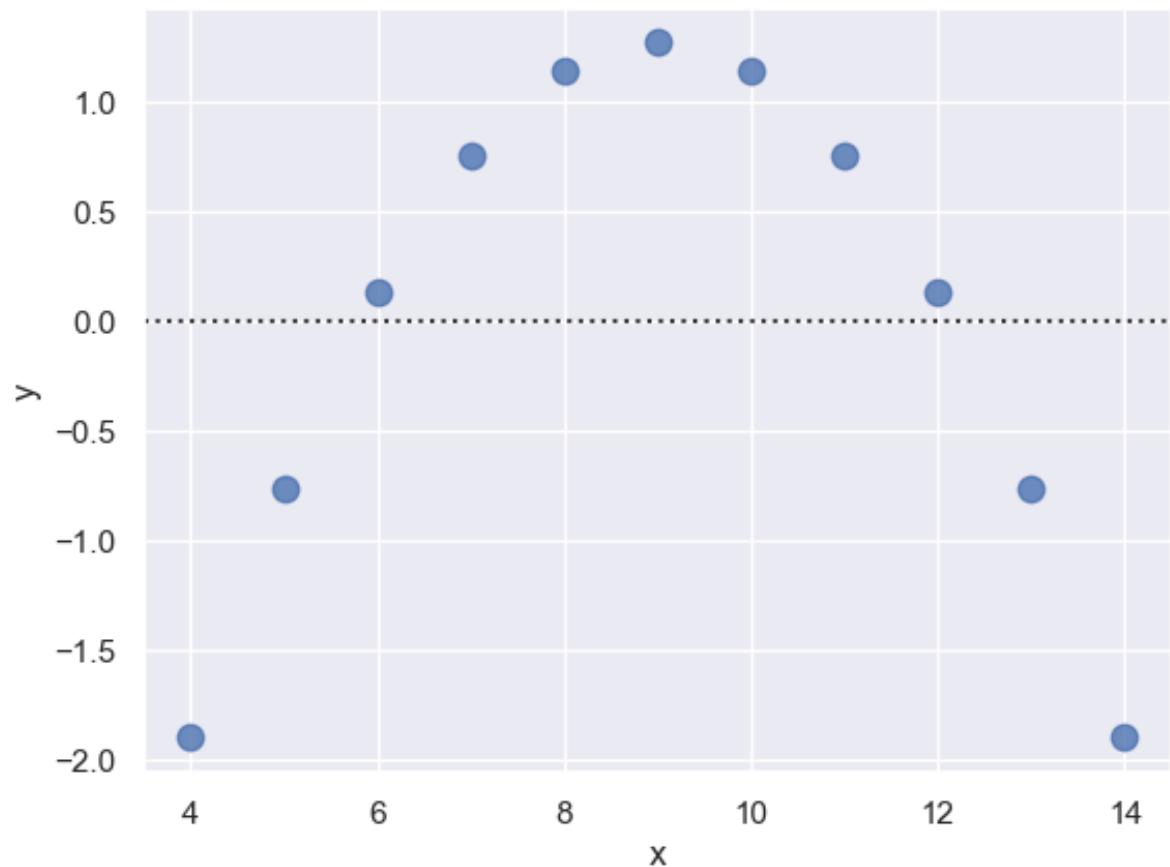
Функция `residplot()` может быть полезным инструментом для проверки соответствия модели простой регрессии набору данных. Она подбирает и удаляет простую линейную регрессию, а затем строит график остаточных значений для каждого наблюдения. В идеале эти значения должны быть случайным образом разбросаны вокруг точки $y = 0$:

```
In [140]: sns.residplot(x="x", y="y", data=anscombe.query("dataset == 'I'"),
                      scatter_kws={"s": 80});
```



Если в остатках присутствует структура, это говорит о том, что простая линейная регрессия не подходит:

```
In [141]: sns.residplot(x="x", y="y", data=anscombe.query("dataset == 'II'"),
                      scatter_kws={"s": 80});
```

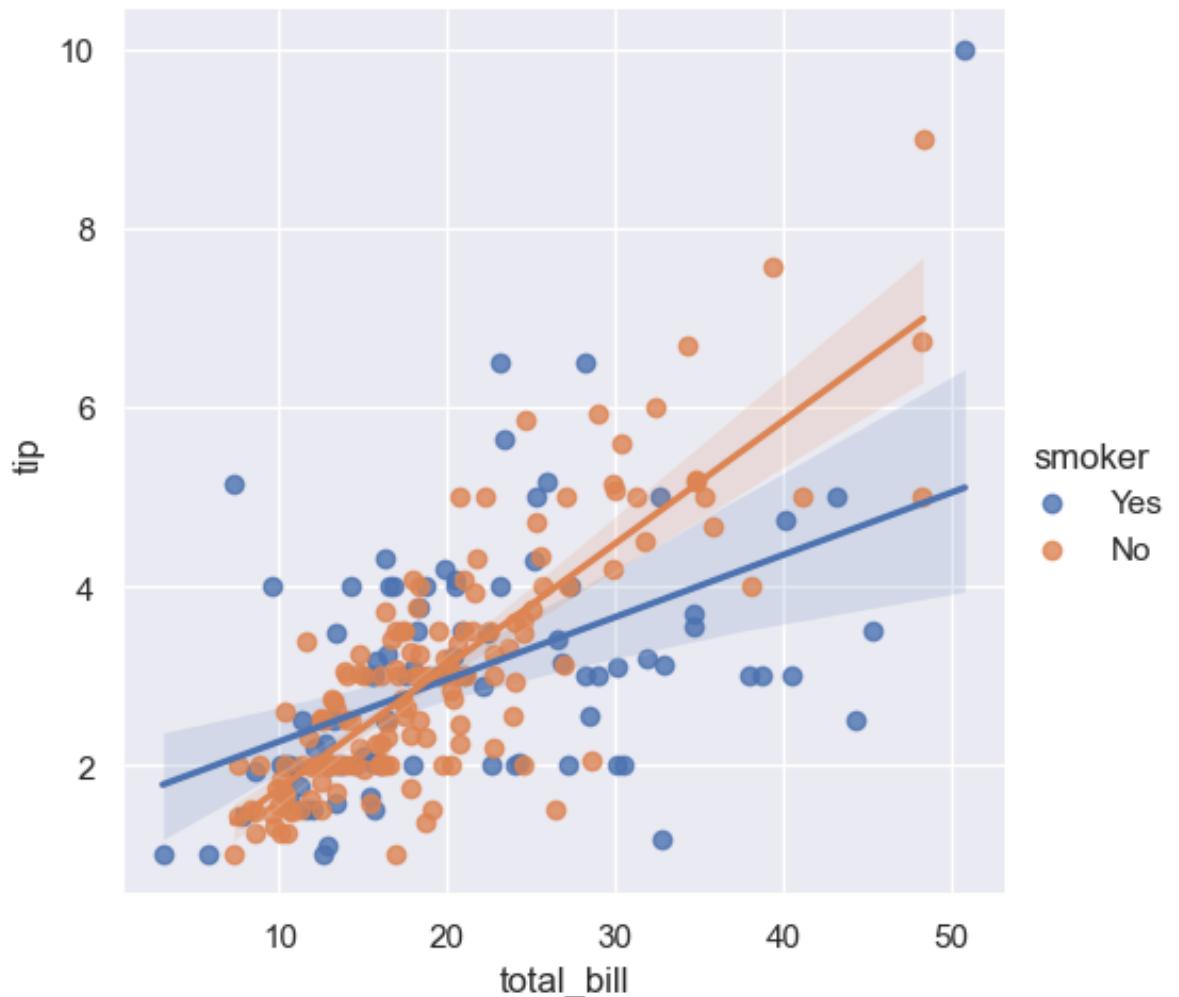


regplot() vs lmplot()

В то время как `regplot()` всегда отображает одну взаимосвязь, `lmplot()` объединяет `regplot()` с `FacetGrid`, предоставляя простой интерфейс для отображения линейной регрессии на «фасетных» графиках, позволяющих исследовать взаимодействие с тремя дополнительными категориальными переменными.

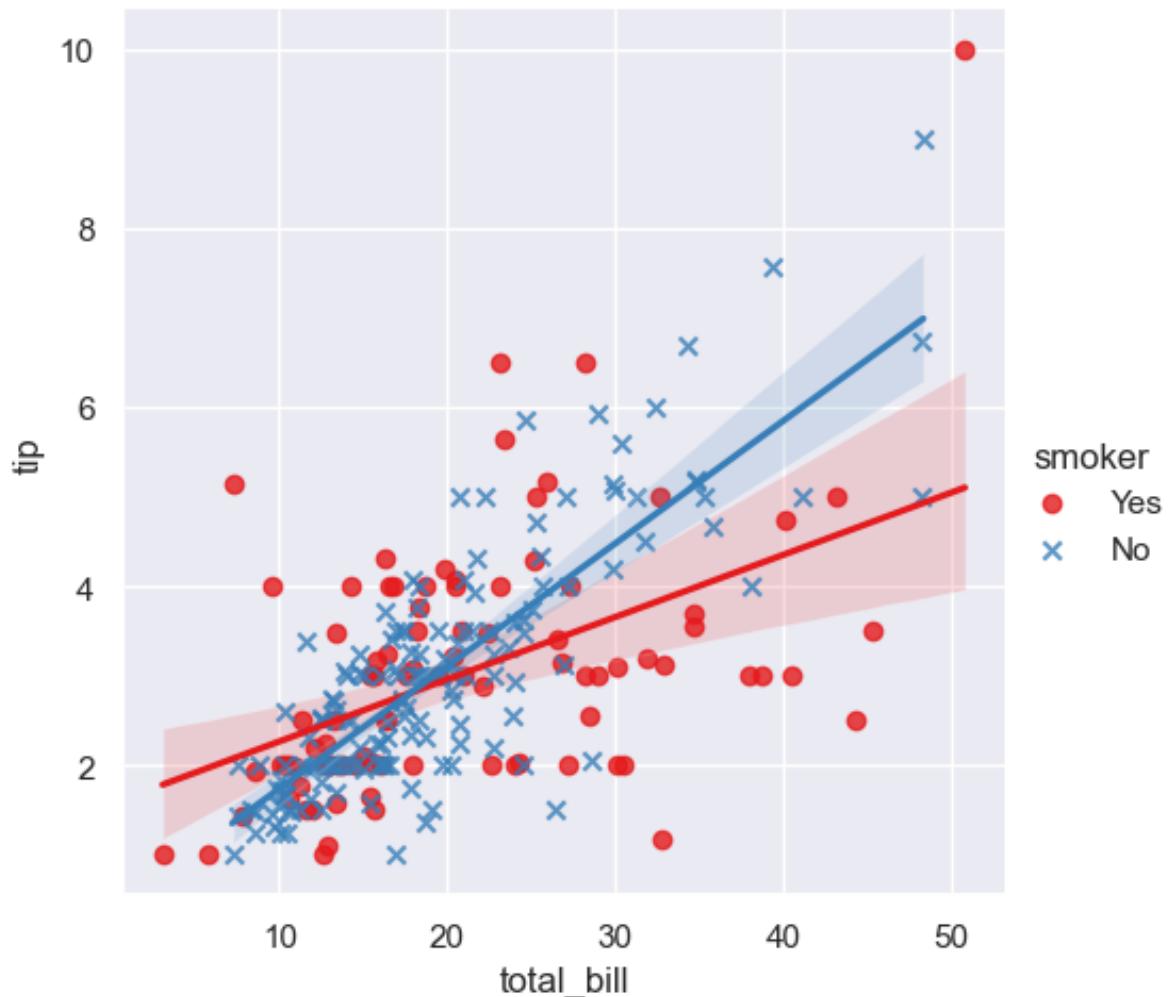
Лучший способ выделить взаимосвязь — отобразить оба уровня на одной оси и использовать цвет для их различия:

```
In [142]: sns.lmplot(x="total_bill", y="tip", hue="smoker", data=tips);
```



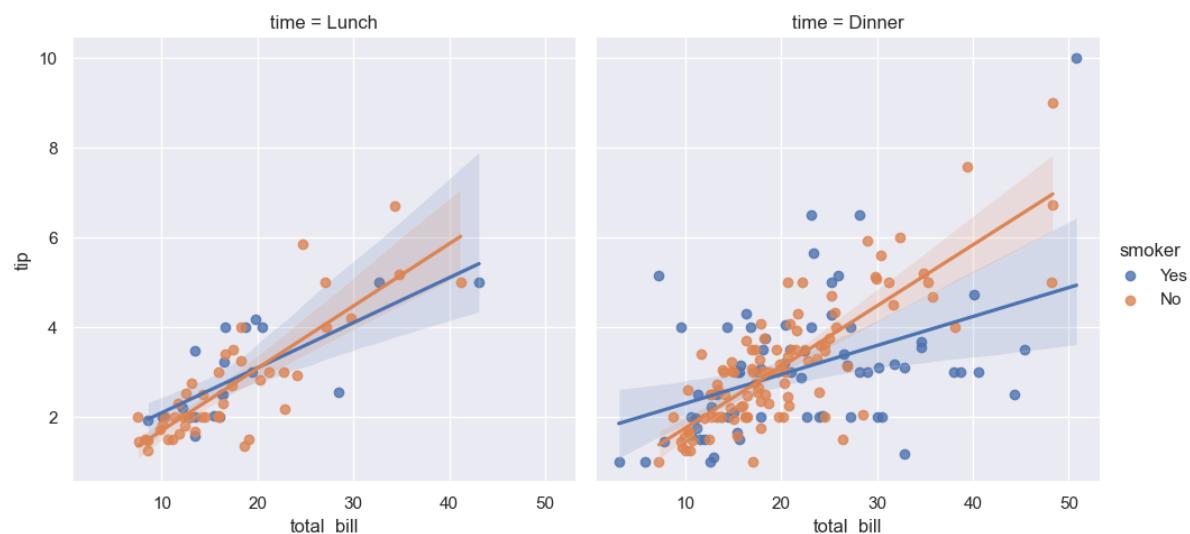
Помимо цвета, можно использовать различные маркеры диаграмм рассеяния для улучшения воспроизведения чёрно-белых изображений. Вы также полностью контролируете используемые цвета:

```
In [143]: sns.lmplot(x="total_bill", y="tip", hue="smoker", data=tips,  
                    markers=["o", "x"], palette="Set1");
```



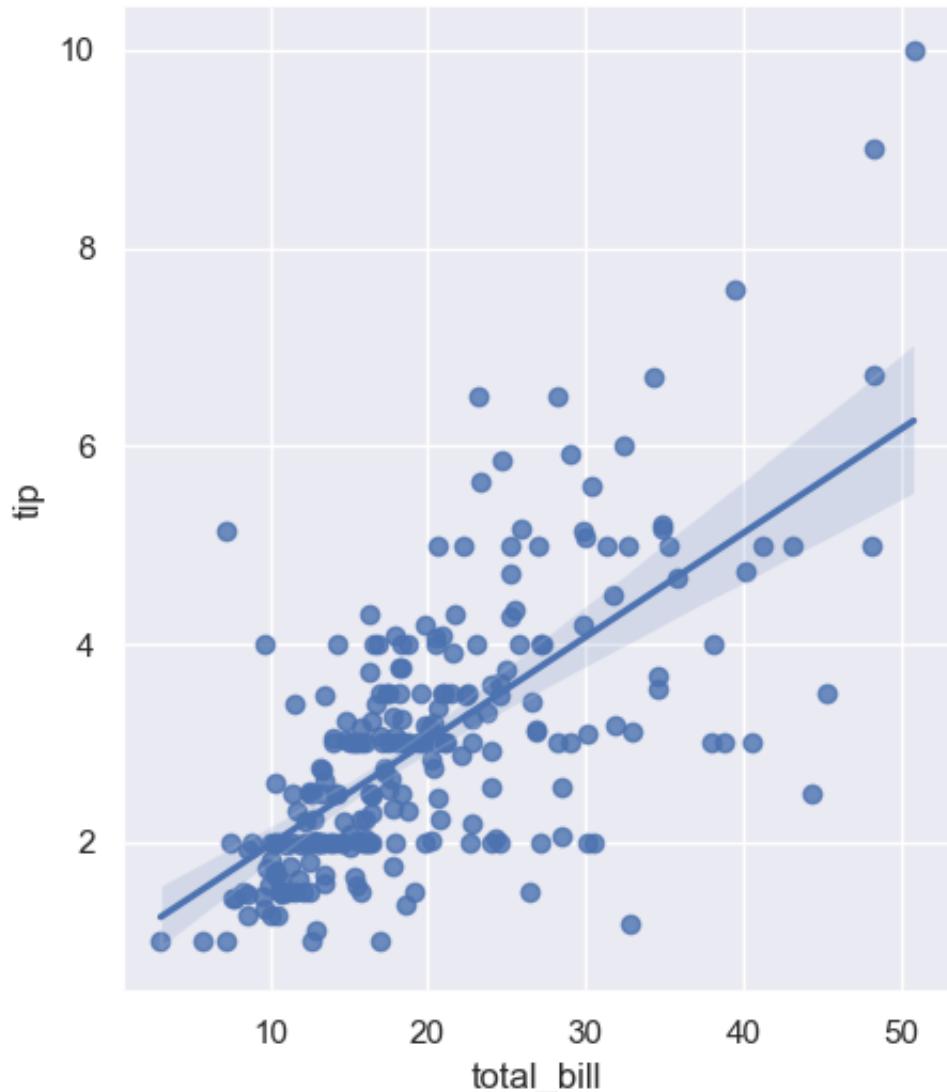
Чтобы добавить еще одну переменную, вы можете нарисовать несколько «граней», каждый уровень которых будет отображаться в строках или столбцах сетки:

```
In [144]: sns.lmplot(x="total_bill", y="tip", hue="smoker", col="time", data=
```



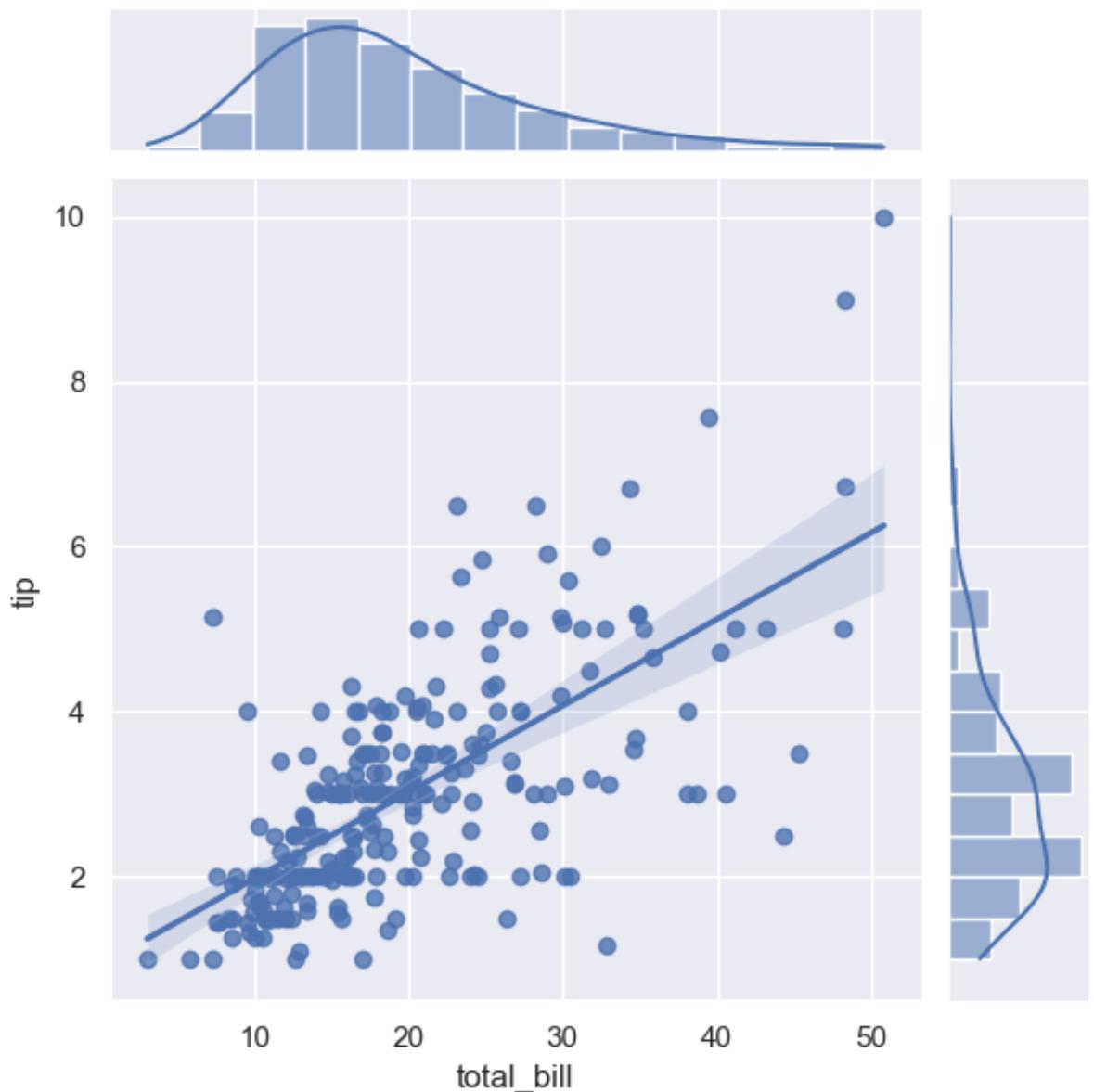
Управление размером и формой графика

```
In [145]: f, ax = plt.subplots(figsize=(5, 6))
sns.regplot(x="total_bill", y="tip", data=tips, ax=ax);
```

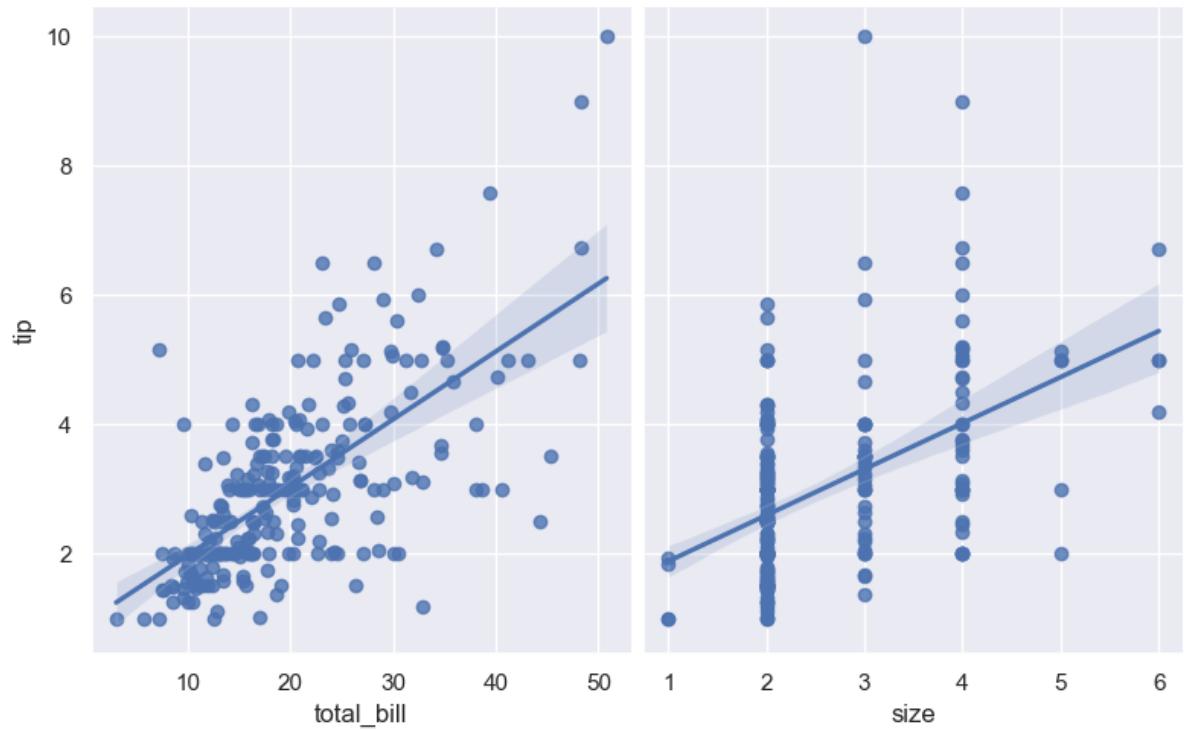


Построение регрессии в других контекстах

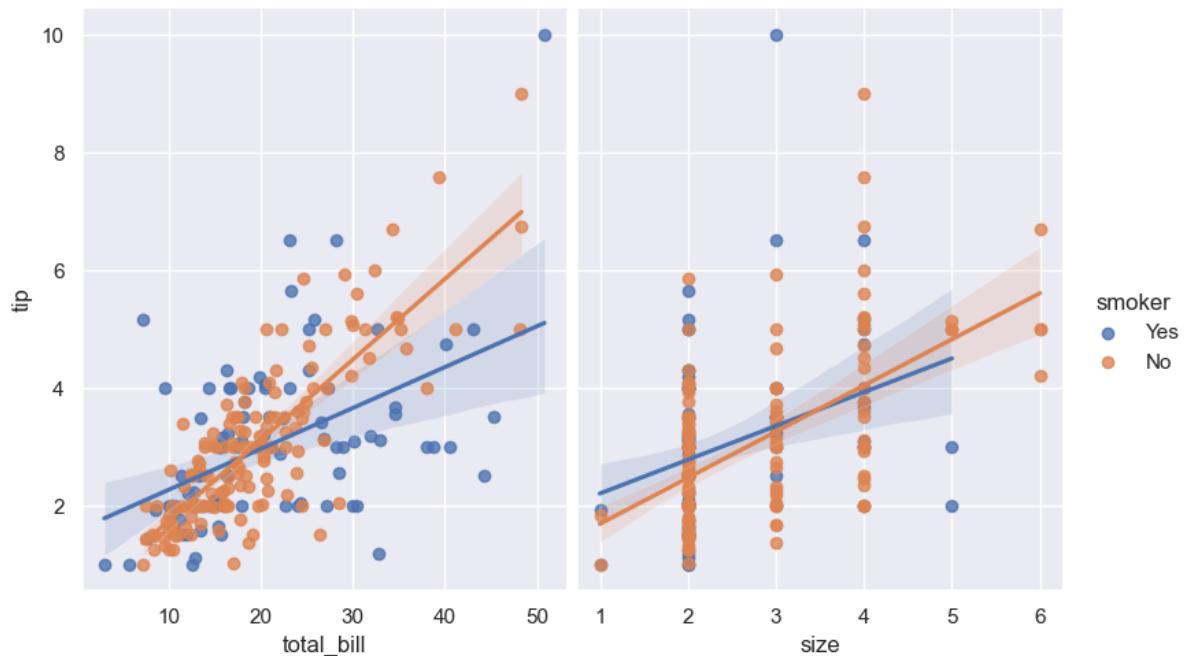
```
In [146]: sns.jointplot(x="total_bill", y="tip", data=tips, kind="reg");
```



```
In [147]: sns.pairplot(tips, x_vars=["total_bill", "size"], y_vars=["tip"], height=5, aspect=.8, kind="reg");
```



```
In [148]: sns.pairplot(tips, x_vars=["total_bill", "size"], y_vars=["tip"], hue="smoker", height=5, aspect=.8, kind="reg");
```

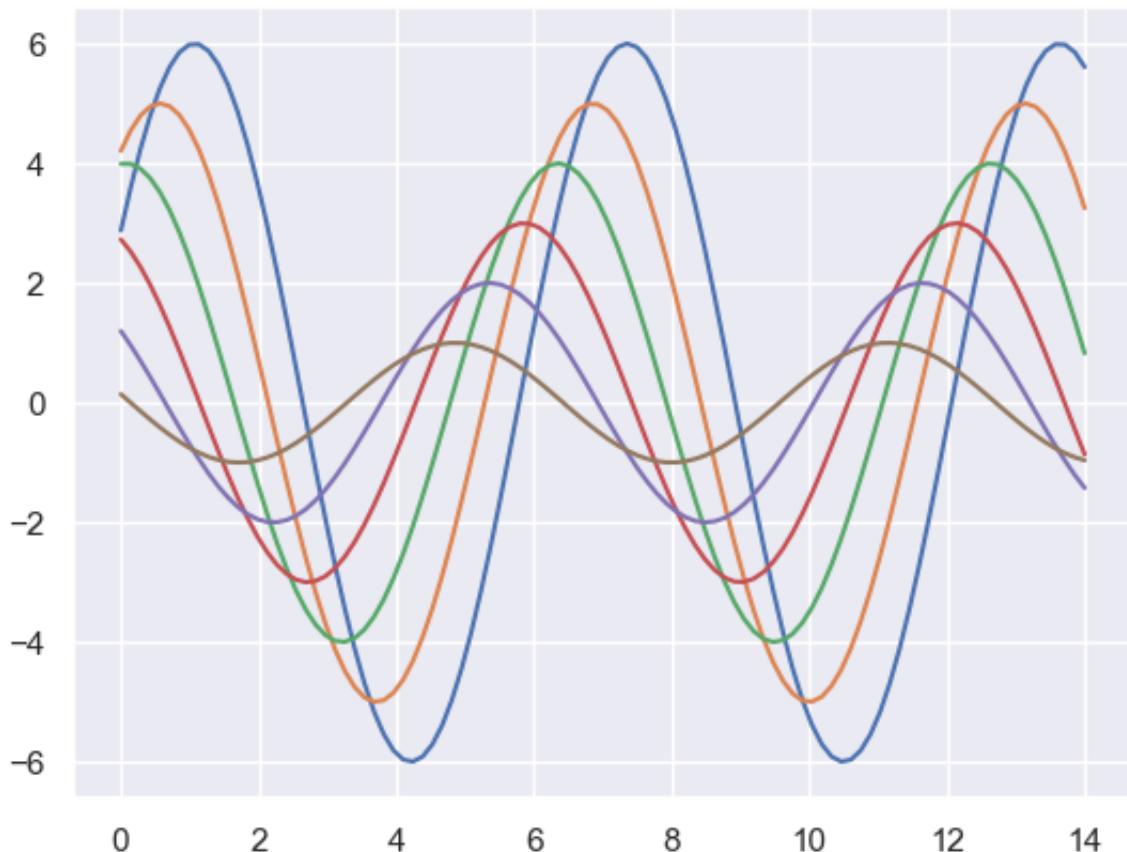


Управление эстетикой рисунка:

- стилизация рисунка
- стилизация осей
- цветовые палитры
- и т. д.

```
In [149]: def sinplot(flip=1):
    x = np.linspace(0, 14, 100)
    for i in range(1, 7):
        plt.plot(x, np.sin(x + i * .5) * (7 - i) * flip)
```

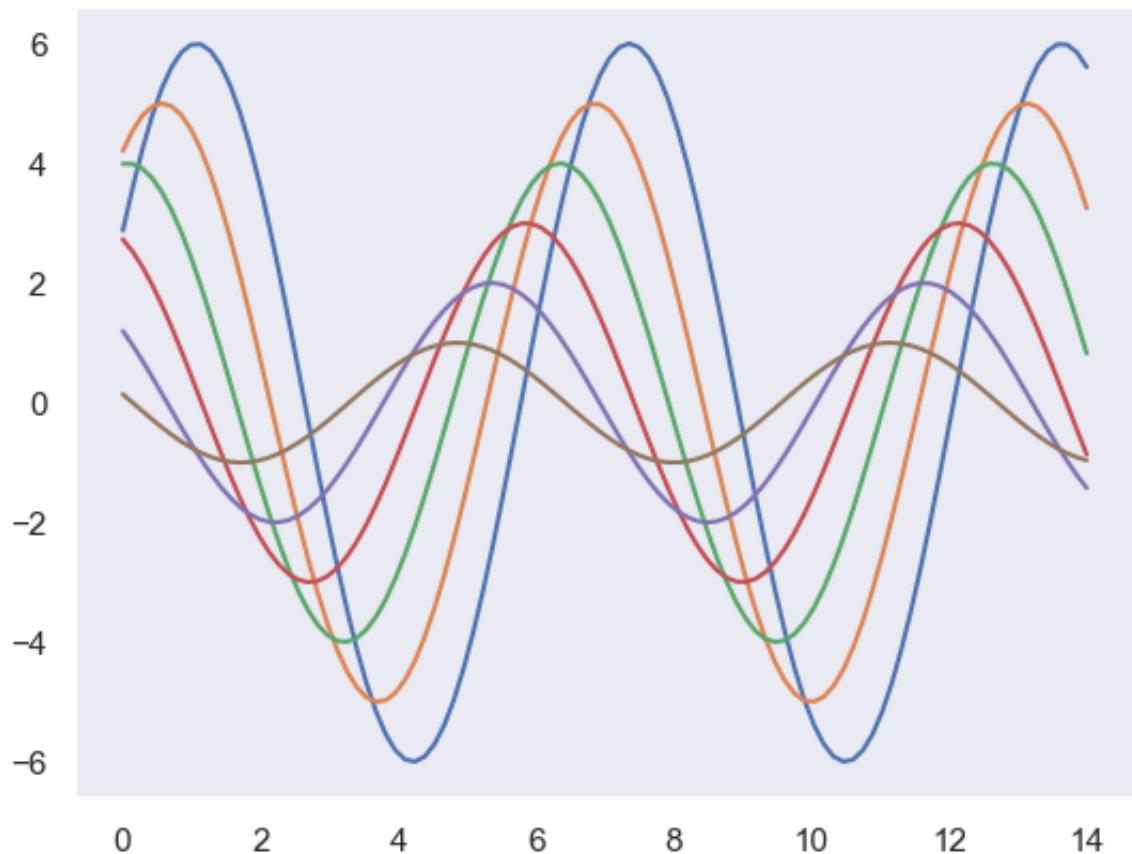
```
In [150]: sinplot()
```



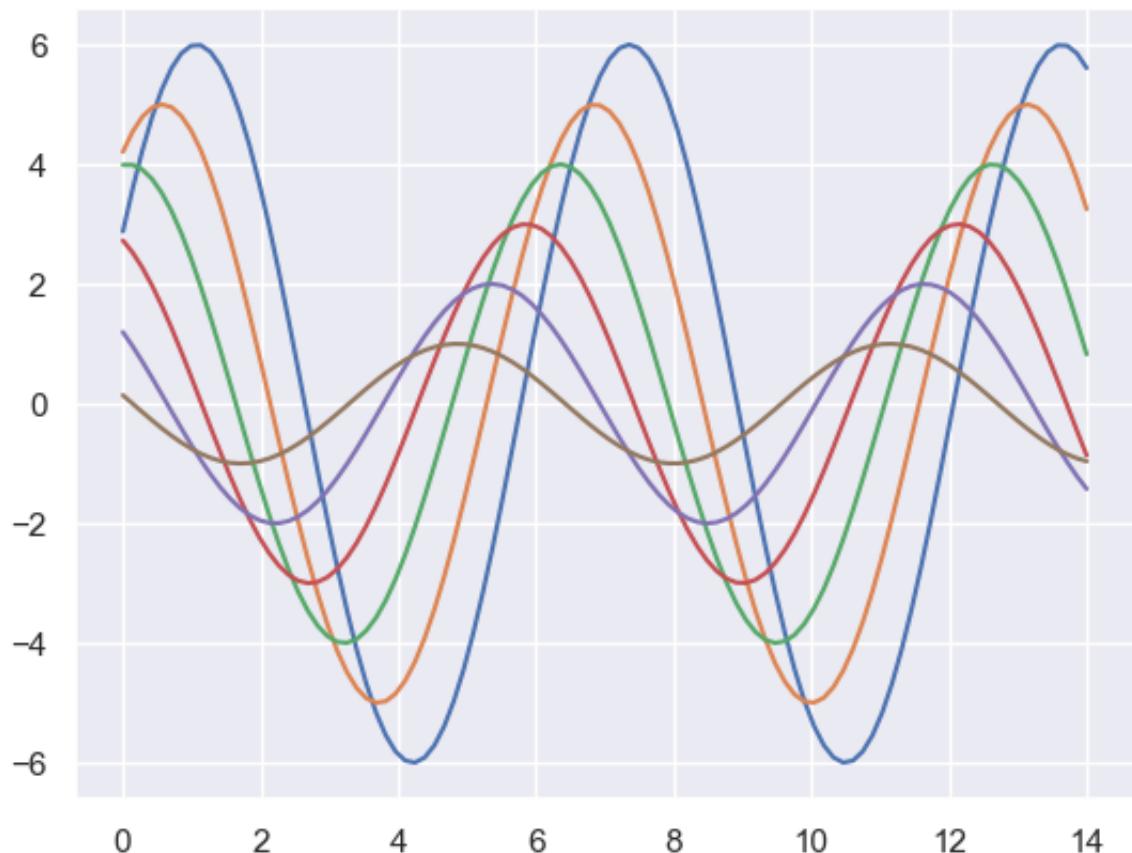
Стили рисунков Seaborn

- dark
- darkgrid
- white
- whitegrid
- ticks

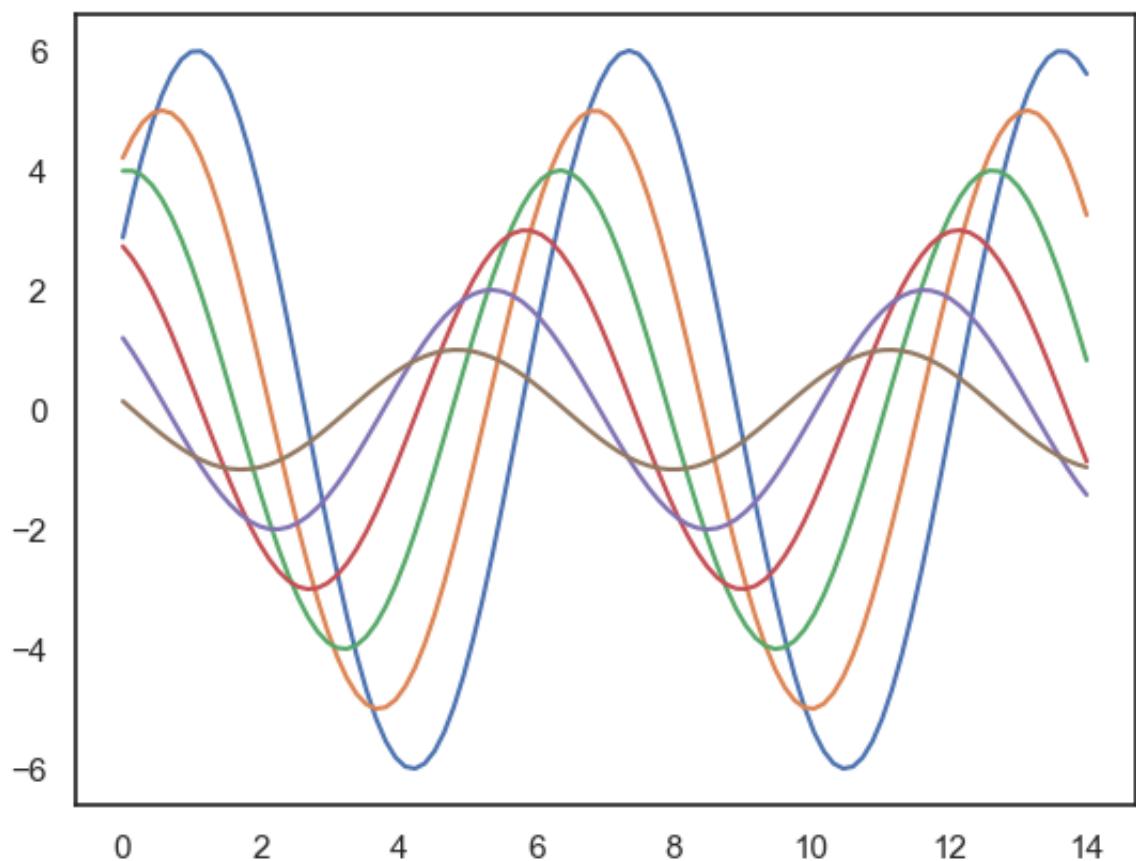
```
In [151]: sns.set_style('dark')
sinplot()
```



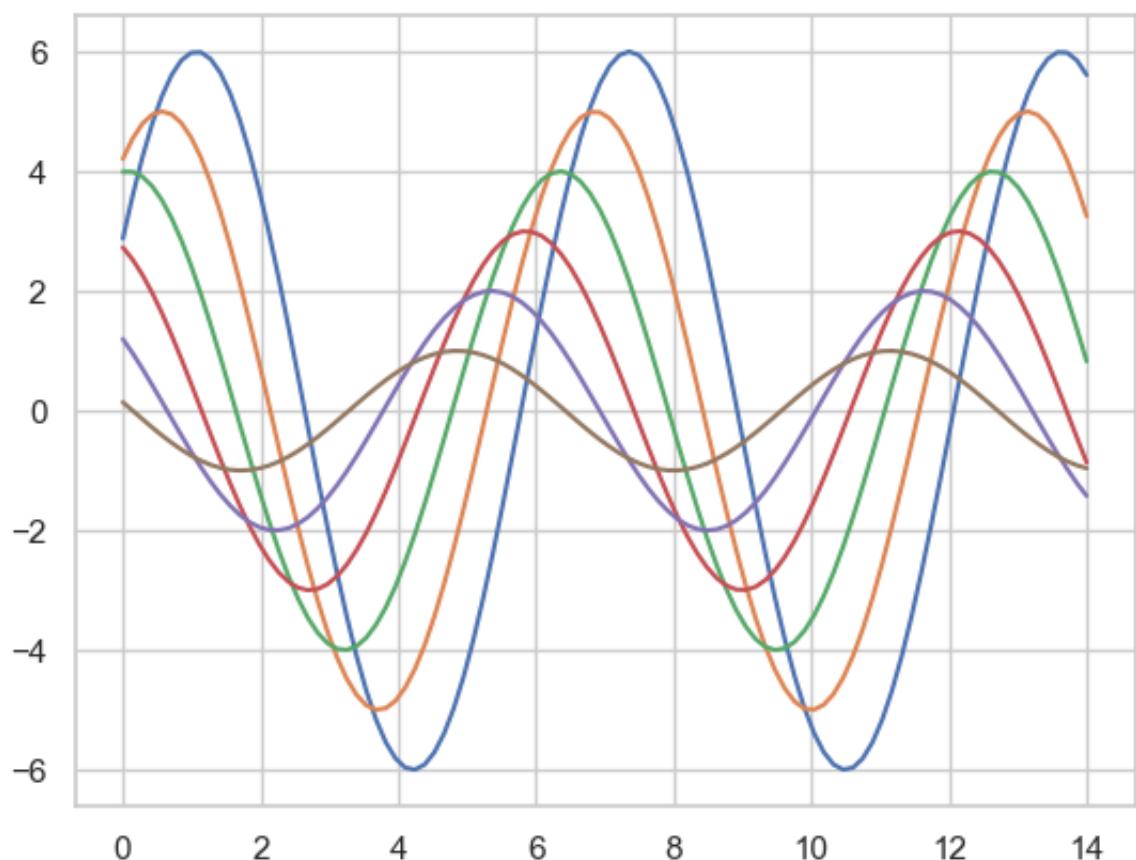
```
In [152]: sns.set_style("darkgrid")
sinplot()
```



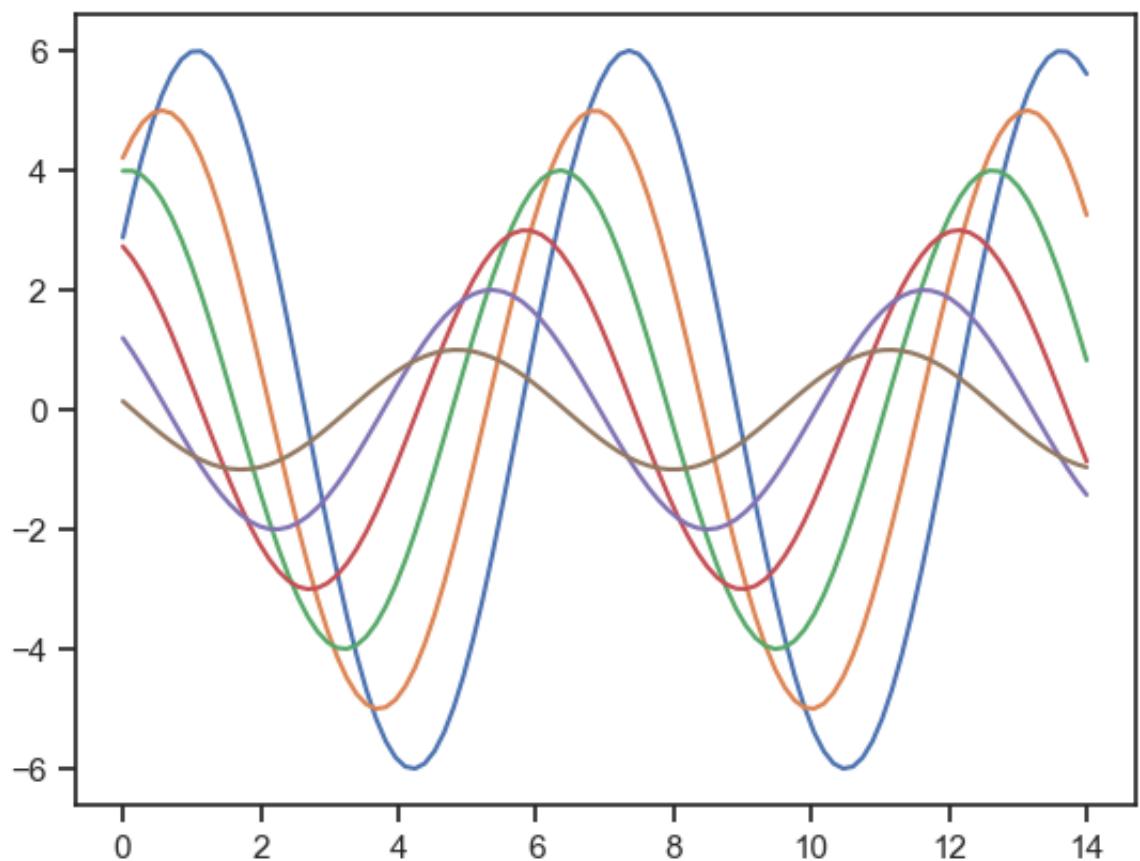
```
In [153]: sns.set_style('white')
sinplot()
```



```
In [154]: sns.set_style("whitegrid")
sinplot()
```

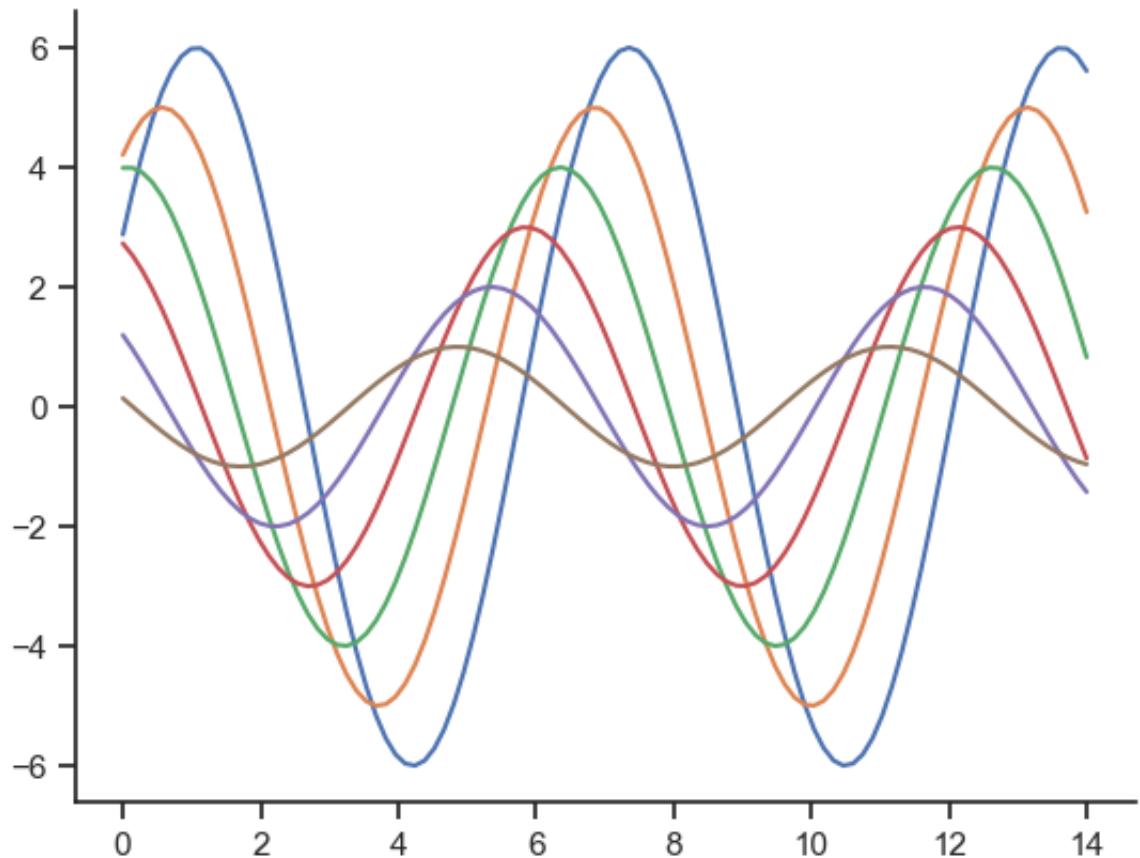


```
In [155]: sns.set_style("ticks")
sinplot()
```



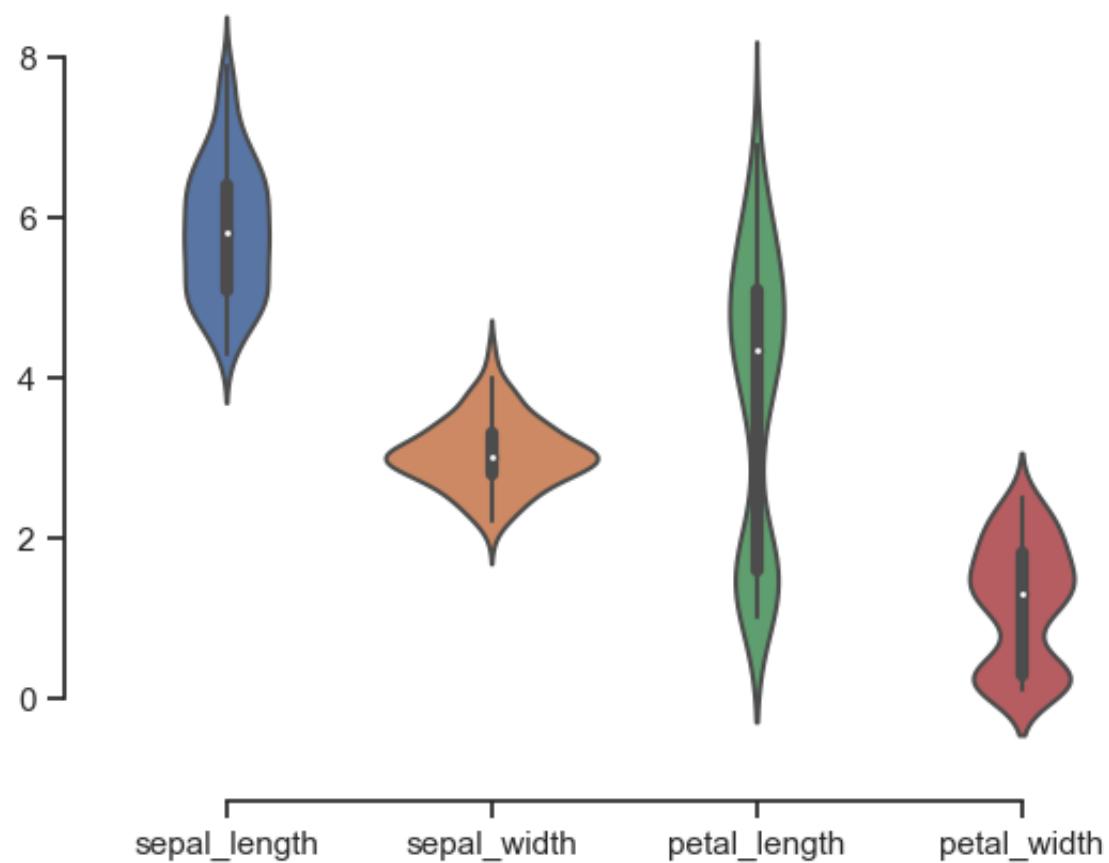
Удаление границ областей рисования

```
In [156]: sinplot()  
sns.despine()
```



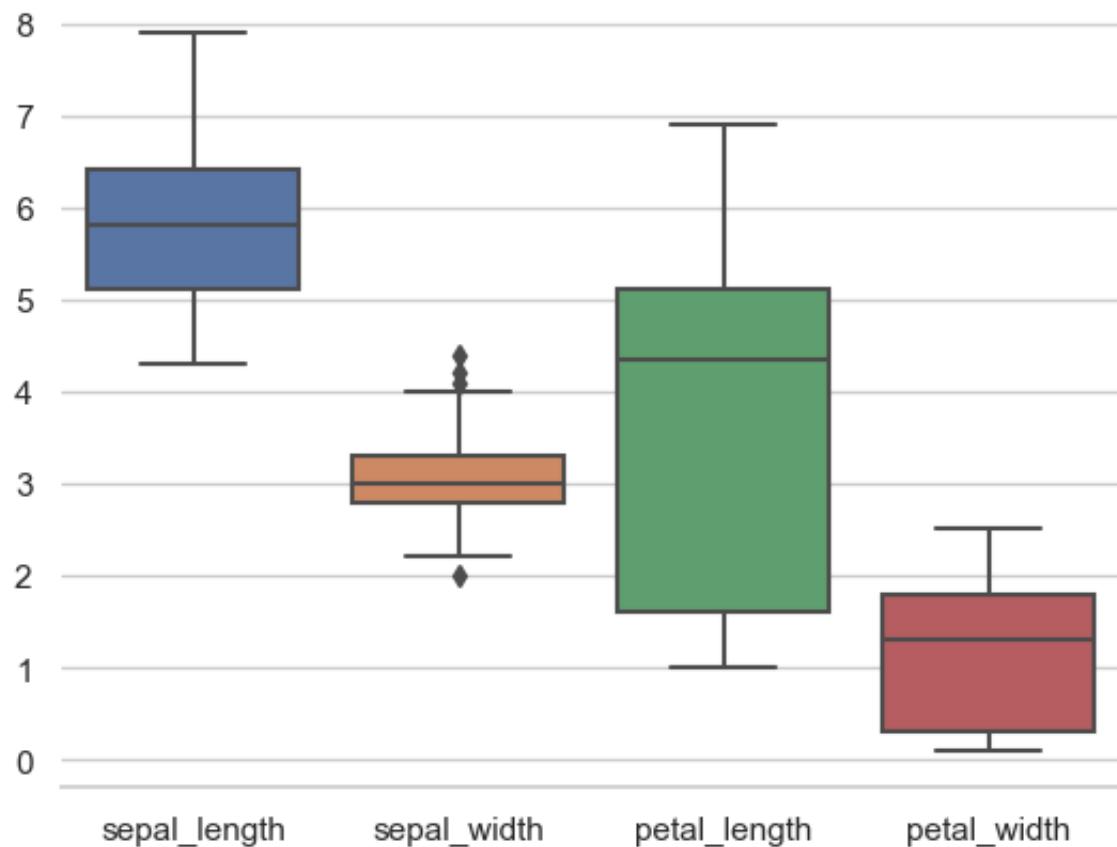
На некоторых графиках полезно смещать рамки рисунка относительно данных, что также можно сделать при вызове функции `despine()`.

```
In [157]: f, ax = plt.subplots()  
sns.violinplot(data=iris)  
sns.despine(offset=10, trim=True)
```



Вы также можете контролировать, какие рамки удаляются, с помощью дополнительных аргументов `despine()`:

```
In [158]: sns.set_style("whitegrid")
sns.boxplot(data=iris, palette="deep")
sns.despine(left=True)
```



Временная установка стиля рисунка

Хотя переключение между стилями не вызывает затруднений, вы также можете использовать функцию `axes_style()` в операторе `with` для временной установки параметров графика. Это также позволяет создавать рисунки с осями разного стиля:

```
In [159]: f = plt.figure(figsize=(6, 6))
gs = f.add_gridspec(2, 2)

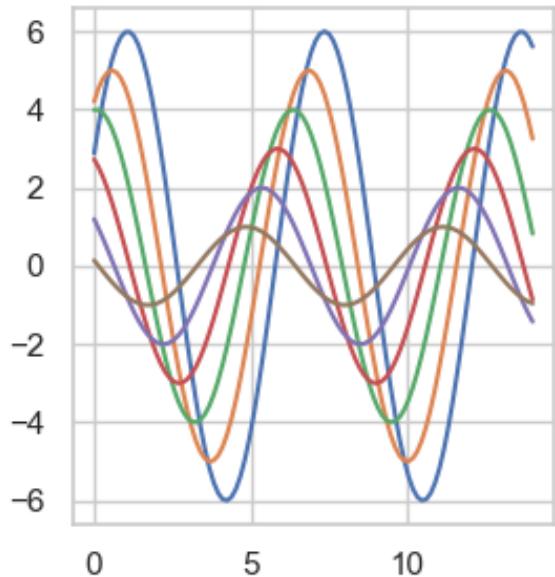
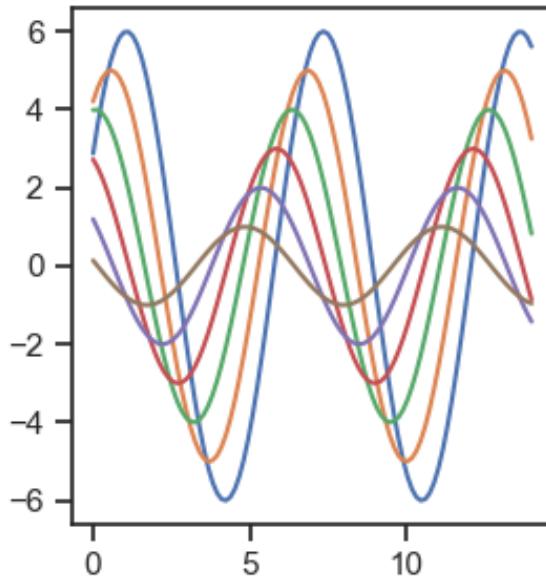
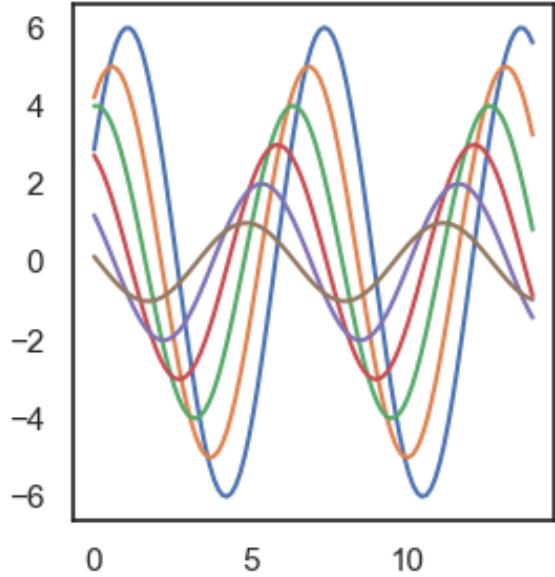
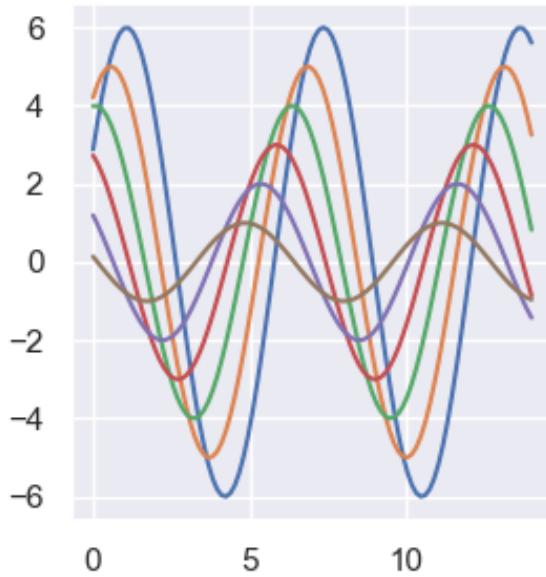
with sns.axes_style("darkgrid"):
    ax = f.add_subplot(gs[0, 0])
    sinplot()

with sns.axes_style("white"):
    ax = f.add_subplot(gs[0, 1])
    sinplot()

with sns.axes_style("ticks"):
    ax = f.add_subplot(gs[1, 0])
    sinplot()

with sns.axes_style("whitegrid"):
    ax = f.add_subplot(gs[1, 1])
    sinplot()

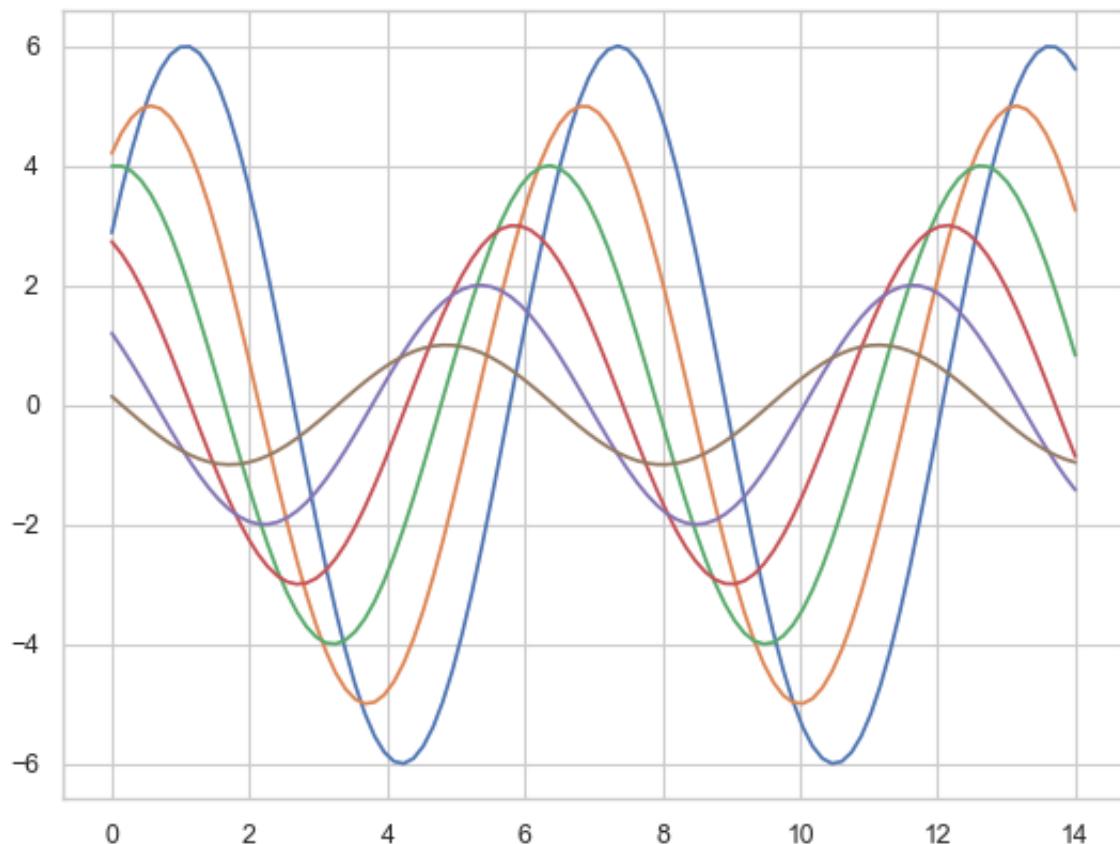
f.tight_layout()
```



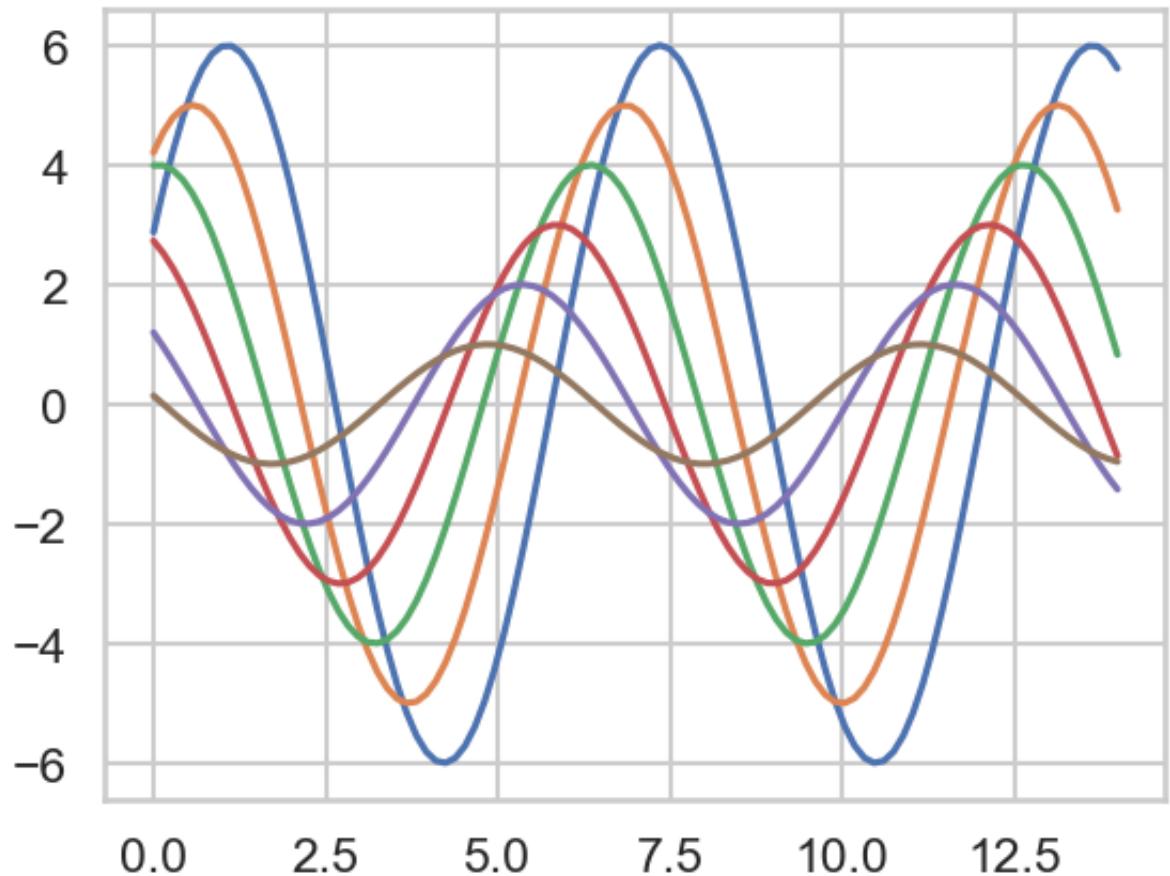
Масштабирование элементов графика

Отдельный набор параметров управляет масштабом элементов графика, что позволяет использовать тот же код для создания графиков, подходящих для использования в условиях, где уместны большие или меньшие размеры графиков.

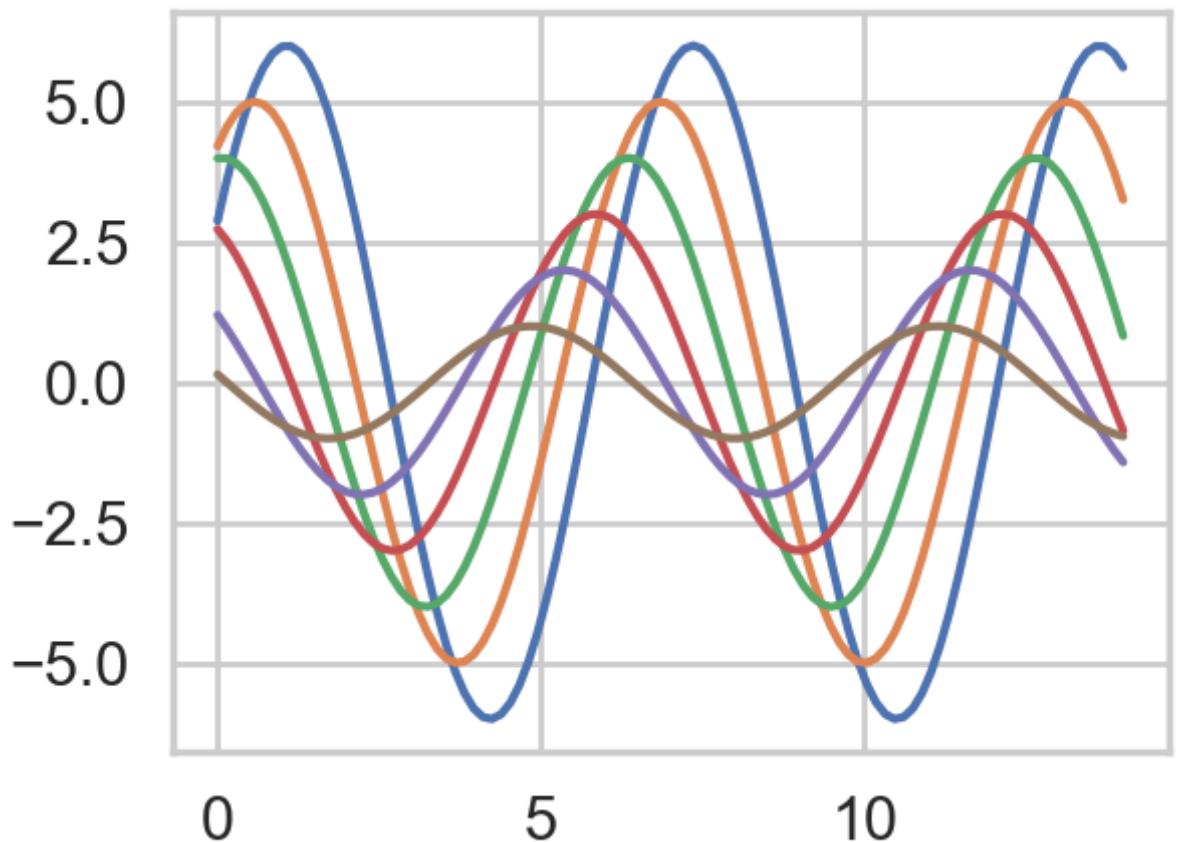
```
In [160]: sns.set_context("paper")
sinplot()
```



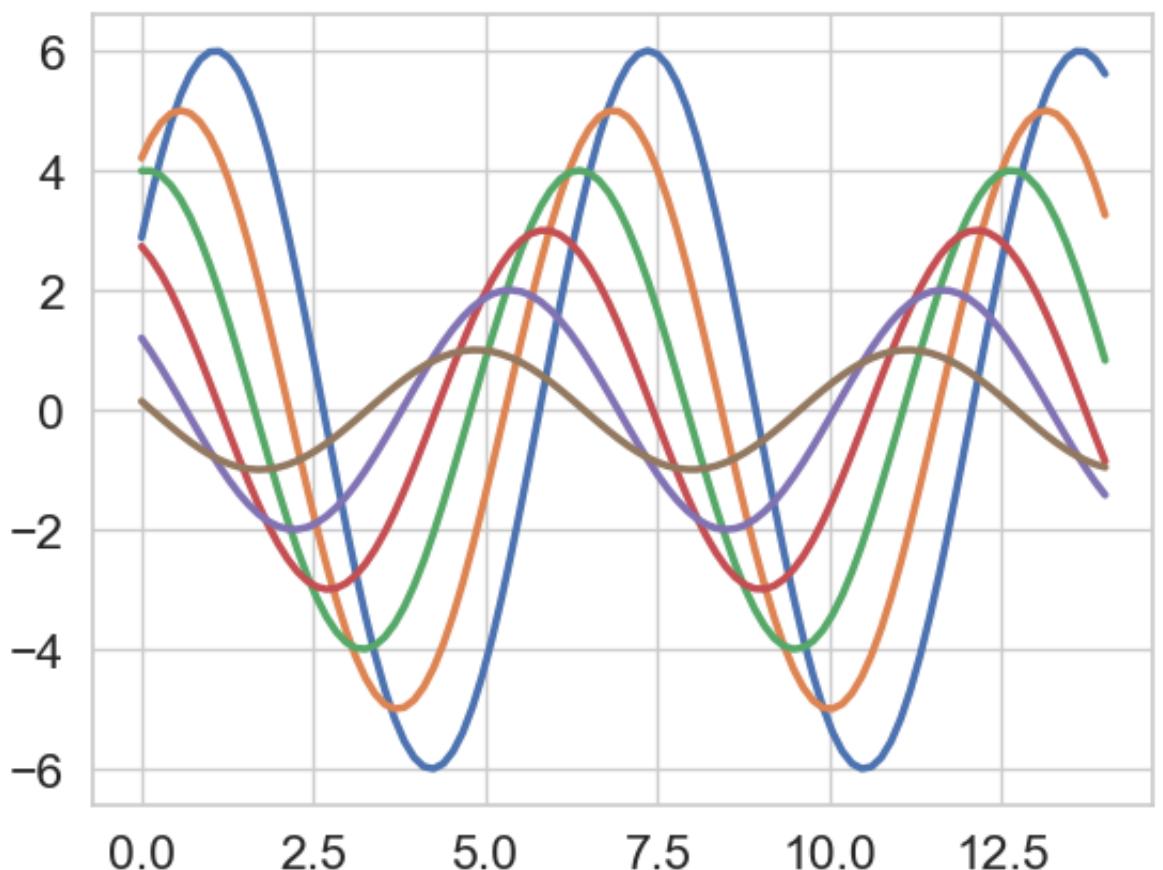
```
In [161]: sns.set_context("talk")
sinplot()
```



```
In [162]: sns.set_context("poster")
sinplot()
```



```
In [163]: sns.set_context("notebook", font_scale=1.5, rc={"lines.linewidth": 2})
sinplot()
```



Задание на ЛР №6

В соответствии с индивидуальным заданием (вариантом), переданным через программу «Мессенджер Яндекс», выполните следующие работы, используя для построения рисунков функции библиотеки seaborn:

1. Считайте из заданного набора данных репозитария UCI значения трех признаков и метки класса.
2. Если среди меток класса имеются пропущенные значения, то удалите записи с пропущенными метками класса. Если в признаках имеются пропущенные значения, то выведите процент записей набора данных с пропущенными значениями и замените пропущенные значения на значения, указанные в индивидуальном задании. При отсутствии пропущенных значений удалите не менее 5% точек набор данных как выбросы при помощи стандартизованной оценки и выведите процент удаленных точек.
3. Постройте диаграмму рассеяния (scatterplot) на плоскости с координатами, соответствующими двум признакам с наибольшей корреляцией, отображая точки различных классов разными цветами и маркерами переменных размеров в зависимости значений третьего признака. Подпишите оси и рисунок, создайте легенду для классов и маркеров.
4. Постройте линейные графики (lineplot) зависимости признака с наибольшей дисперсией от признака с наименьшей дисперсией для разных классов, выделяя линии для разных классов согласно индивидуальному заданию. Подпишите оси и рисунок, создайте легенду для классов.
5. Постройте диаграмму категориального рассеяния (stripplot или swarmplot) зависимости признака с наименьшей дисперсией от классов типа, указанного в индивидуальном задании. Подпишите оси и рисунок, создайте легенду для классов.
6. Постройте диаграмму размаха (boxplot, boxenplot или violinplot) зависимости признака с наибольшей дисперсией от классов типа, указанного в индивидуальном задании. Подпишите оси и рисунок, создайте легенду для классов. Перечислите признаки, в которых выявлены выбросы.
7. Постройте столбчатую диаграмму типа, указанного в индивидуальном задании (barplot, countplot, pointplot), показывающую зависимость признака с наименьшей разностью между третьим и первым квартилями от класса. Подпишите оси и рисунок, создайте легенду для классов.
8. Постройте диаграмму одномерной оценки плотности (kdeplot) для второго признака. Подпишите оси и рисунок.
9. Постройте диаграмму двумерной оценки плотности (jointplot) для первого и третьего признаков. Подпишите оси и рисунок.
10. Постройте график регрессии (regplot или lmplot) зависимости второго признака от первого признака, используя полиномиальную модель второго порядка. Подпишите оси и рисунок.

