Визуализация данных: Matplotlib и Seaborn

А зачем?

Визуализация данных помогает быстрее понять суть данных, прикинуть зависимости, выявить закономерности и тенденции, обнаруживать аномалии (выбросы). Это фундаментальный этап анализа данных, мы начинаем строить свои предположения исходя из информации, полученной от графиков, гистограмм, скаттерплотов (диаграмм рассеяния). Это не всемогущий инструмент, его использование (особенно некорректное) может приводить к ошибкам и ложным предположениям, но его нельзя не использовать из-за количества иинформации с которого можно начать анализ.

`matplotlib` vs `seaborn`

Для визуализации чаще всего применяются две библиотеки: `matplotlib`, к которой привыкли очень много пользователей, и `seaborn`, основанная на прошлой библиотеке, но привносящая свои плюсы: более лёгкое использование, визуализации чаще красивее, чем в `matplotlib`, даже если ничего не настраивать и легче работать с `Pandas`, которую мы разбирали в прошлой части.

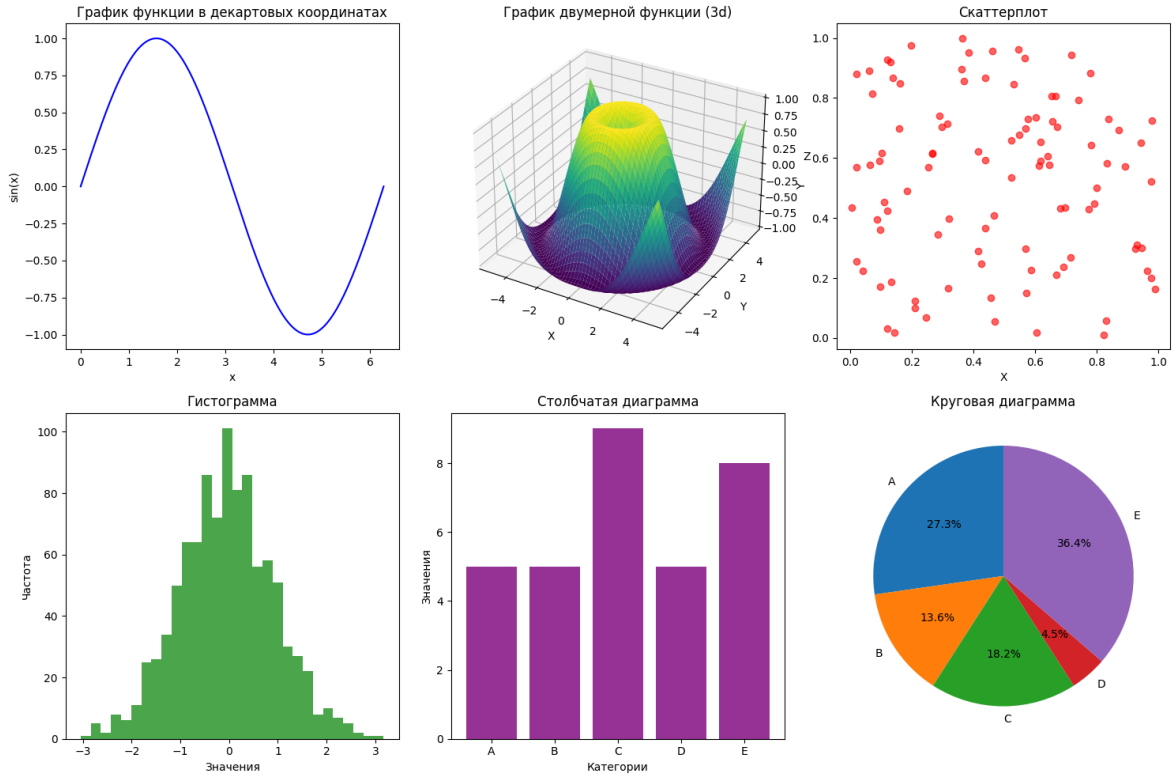
Типы графиков | Экспресс-версия

Можно сказать, что ниже экспресс-версия этой части. Здесь я представил самые распространённые графики/диаграммы в анализе данных, но не все. Есть способы рисовать графики в 4-ёх мерном пространстве, это даже не так сложно, но мы опустим. Аффинные, полярные, цилиндрические и другие системы координат также рассмотрены не будут, во-первых потому что встречаются они не так часто, а во-вторых, вы всегда сможете найти это сами и разобраться будет не сложнее, чем в материале, представленном ниже.

ВНИМАНИЕ: код ниже не запустится, если у вас отстутствует библиотека `matplotlib` (подразумевается, что `numpy` у вас уже есть).

In [3]:
```
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(0)
fig = plt.figure(figsize=(15, 10))

ax1 = fig.add_subplot(2, 3, 1)
x = np.linspace(0, 2*np.pi, 400)
y = np.sin(x)
ax1.plot(x, y, color='blue')
```

```
ax1.set_title("График функции в декартовых координатах")
ax1.set_xlabel("x")
ax1.set_ylabel("sin(x)")

ax2 = fig.add_subplot(2, 3, 2, projection='3d')
X = np.linspace(-5, 5, 50)
Y = np.linspace(-5, 5, 50)
X, Y = np.meshgrid(X, Y)
Z = np.sin(np.sqrt(X**2 + Y**2))
ax2.plot_surface(X, Y, Z, cmap='viridis')
ax2.set_title("График двумерной функции (3d)")
ax2.set_xlabel("X")
ax2.set_ylabel("Y")
ax2.set_zlabel("Z")

ax3 = fig.add_subplot(2, 3, 3)
x_scatter = np.random.rand(100)
y_scatter = np.random.rand(100)
ax3.scatter(x_scatter, y_scatter, color='red', alpha=0.6)
ax3.set_title("Скаттерплот")
ax3.set_xlabel("X")
ax3.set_ylabel("Y")

ax4 = fig.add_subplot(2, 3, 4)
data = np.random.randn(1000)
ax4.hist(data, bins=30, color='green', alpha=0.7)
ax4.set_title("Гистограмма")
ax4.set_xlabel("Значения")
ax4.set_ylabel("Частота")

ax5 = fig.add_subplot(2, 3, 5)
categories = ['A', 'B', 'C', 'D', 'E']
values = np.random.randint(1, 10, size=len(categories))
ax5.bar(categories, values, color='purple', alpha=0.8)
ax5.set_title("Столбчатая диаграмма")
ax5.set_xlabel("Категории")
ax5.set_ylabel("Значения")

ax6 = fig.add_subplot(2, 3, 6)
pie_values = np.random.randint(1, 10, size=len(categories))
ax6.pie(pie_values, labels=categories, autopct='%1.1f%%', startangle=90)
ax6.set_title("Круговая диаграмма")

plt.tight_layout()
plt.show()
```



Установка | Полная версия (отсюда и далее)

`pip install matplotlib seaborn` (мы устанавливаем две библиотеки сразу). Кстати, работу с терминалом из блокнота мы уже демонстрировали, но, если кто не догадался, то пакеты также можно устанавливать прямиком из нотбука. Хорошим тоном является комментирование таких строк, этому правилу последуем и мы. При необходимости выполнить команду строку нужно расскоментировать и запустить (ничем не отличается от запуска обычной python-ячейки; на Windows иногда не работает, в таком случае используйте более "традиционный" подход через терминал).

In [ ]:
```
# %pip install matplotlib seaborn
```

`Matplotlib`

`Figure` и `Axis`

Matplotlib держится на двух китах: ~~Stack Overflow и ChatGPT~~ `Figure` и `Axes`. Первое --- верхнеуровневый контейнер, грубо говоря холст, на котором рисуются все графики, легенды, надписи и прочее. Внутри одного `Figure` может быть несколько `Axes`. Последнее --- область внутри `Figure`, неосредственно на которой строится сам график, там располагаются оси координат, линии, точки, подписи, в общем всё то, что относится напрямую к визуализации. `Axes` содержит `Axis` (оси) и `Artist` --- элементы (линии, точки и т.д.). `Axis` всегда не более трёх. Для упомянутых графиков многомерных ($\mathbb{R}^{\geqslant 4}$) пространств используются

ухищрения как изменение цвета, формы или размера для отображения изменения по четвёртой оси. Изменения в `Axes` не влияет на весь `Figure`.

Основные команды

Импорт

In [1]:
```
# я помню, что выше я уже импортировал эти библиотеки
import numpy as np
import matplotlib.pyplot as plt
```

Построение графиков

Вообще, если планируется использовать графики в Jupyter, то нужно использовать следующую магическую команду: `%matplotlib inline` (лучше всего в самом начале нотбука), чтобы графики отражались напрямую в VS Code. У меня всё работает и без этой команды, но об этом нужно помнить и можно сказать, что это "правило хорошего тона", которое нужно соблюдать.

In [5]:
```
%matplotlib inline
```

Другое правило хорошего тона перед каждым новым графиком создавать новый объект `Figure`. Если вам нужен только один простой график за весь документ, можно не создавать. Чисто в теории, если у вас только один график на ячейку, то тоже можно не создавать, но некоторые так не считают и рекомендуют создавать новый объект всё равно. Кого слушать --- дело ваше. Инициализация переменной `fig` также необязательна (указать игнорирование возвращаемого значения можно с помощью `_` вместо имени переменной), но если вам нужна гибкость в настройках размеров, разрешения, фона, расположения и количество осей и другое, то тогда вам `fig` нужен.

Последнее правило хорошего тона: для отображения вызывать `plt.show()`. График отобразится и без вызова, но если вы перейдёте из блокнота в обычный скрипт, то там уже так не сработает и для того, чтобы привыкнуть мы будем использовать `plt.show()`. В своих проектах вам делать это не обязательно, но если вы работаете не один, а в команде и если в команде ваш голос не решающий, то договоритесь о каком-то едином стиле; собственно, речь о единообразии идёт с первой части (которая 1, а не 0).

In [17]:
```
# _ = plt.figure() # возвращаемая переменная игнорируется
fig = plt.figure()
pass
```

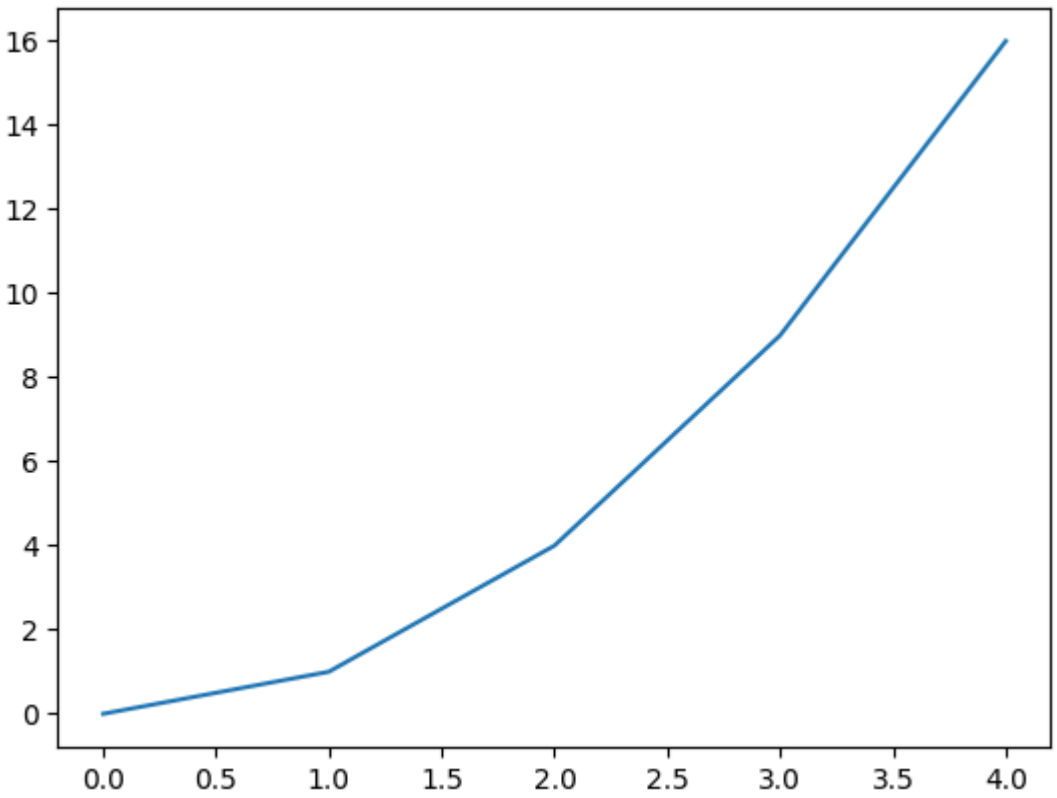
```
<Figure size 640x480 with 0 Axes>
```

Lineplot

С этим термином есть небольшие проблемы, поскольку в английском это слово обозначает привычные нам графики, то в переводе на русский мы получаем "Линейный график", что несколько отличается по значению от первоначального

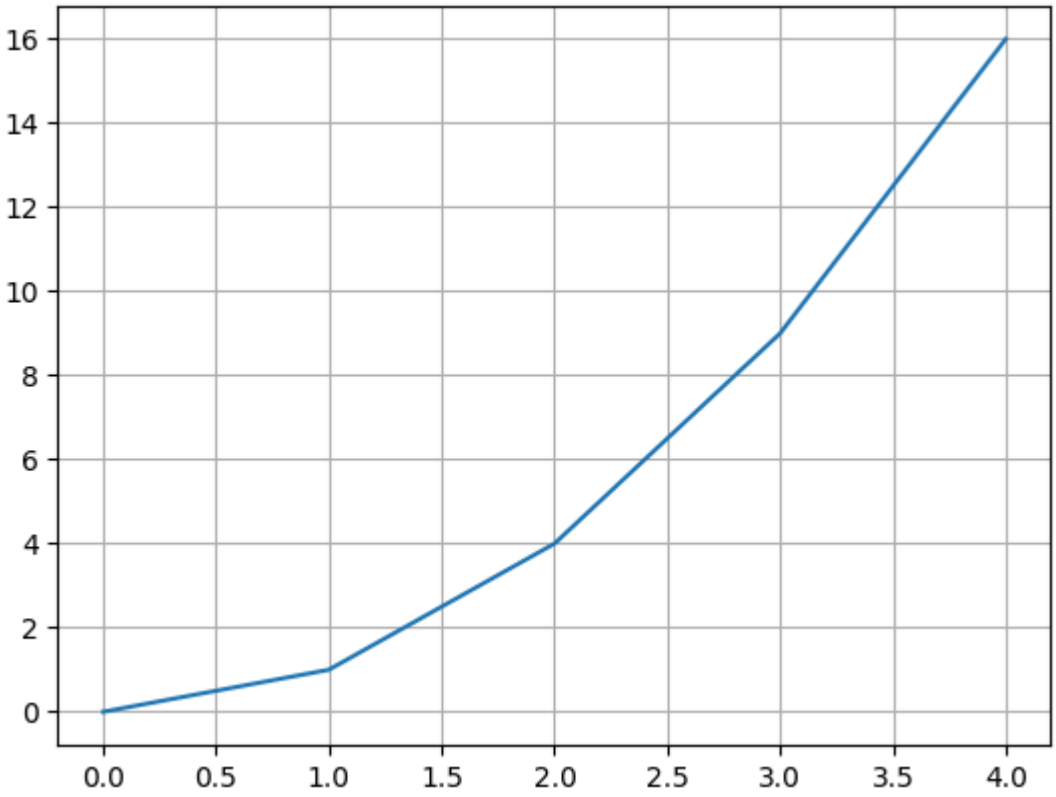
(e.g.: график $y = x^2$ --- это `Lineplot`, но это *не* линейный график), поэтому под `Lineplot` мы будем иметь в виду обычный график, хоть это весьма неконкретное понятие.

In [84]:
```
x = [0, 1, 2, 3, 4]
y = [0, 1, 4, 9, 16]
plt.plot(x, y)
plt.show()
```



Для большего удобства чтения графика можно добавить сетку.

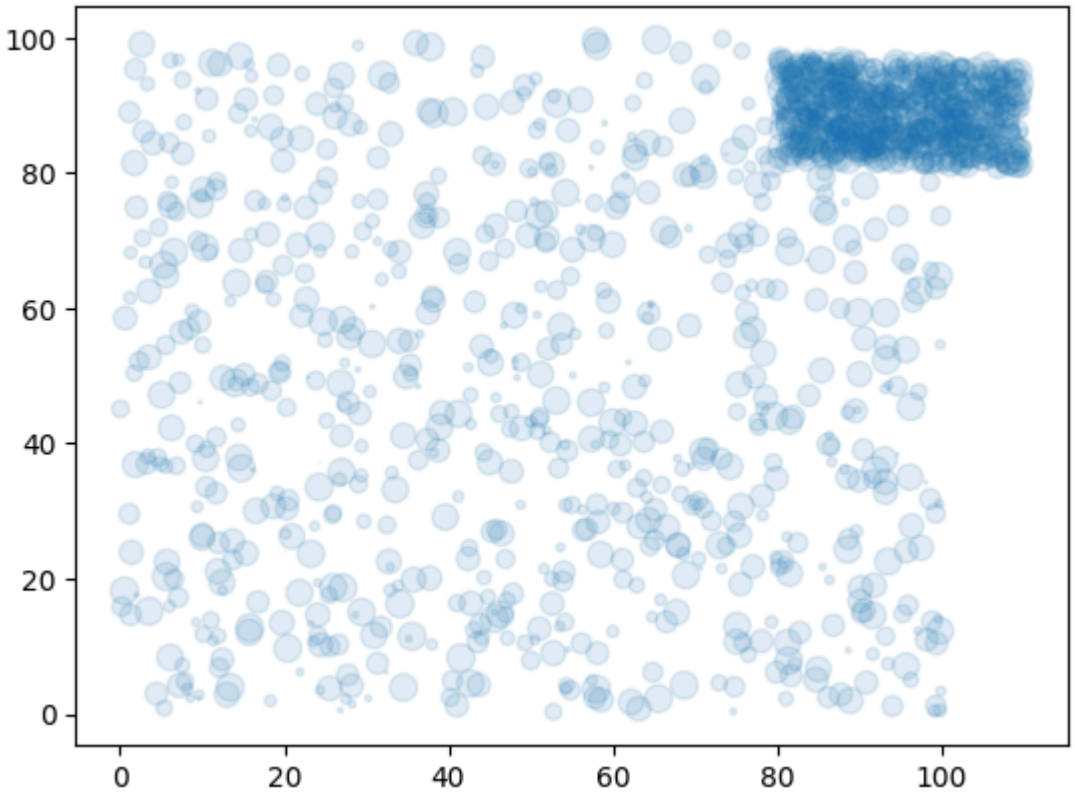
In [85]:
```
fig = plt.figure()
plt.plot(x, y)
plt.grid() # може после plot
plt.show()
```



Scatterplot / Скаттерплот / Диаграмма рассеивания

Пример ниже искусственен донельзя, но вместе с тем и нагляден. Скаттерплот нужен для того, чтобы прикидывать зависимость одного параметра от другого и смотреть как зависит третий параметр от них обоих. В качестве "третьего параметра" у нас выступает размер, у вас, скорее всего, это будет искомое значение, будь то число или класс. Необязательно использовать размер, с таким же успехом (или даже большим) можно раскрасить их в разные цвета: тепловая диаграмма (меньше -- холодные цвета, больше -- тёплые цвета) или конечный набор цветов (в случае классификации/кластеризации, например). Для того, чтобы легче было оценивать большие скопления каждую точку делают полупрозрачной (ну или восмьюпрозрачной, как в нашем случае) и скопление точек в правом верхнем углу теперь привлекает ещё больше внимания.

In [82]:
```
fig = plt.figure()

count = 750
x = np.random.rand(count) * 30 + 80
y = np.sqrt(15**2 - x) + 70 +  np.random.rand(count) * 15

noise_x = np.random.rand(count) * 100
noise_y = np.random.rand(count) * 100

x = np.concatenate((x, noise_x))
y = np.concatenate((y, noise_y))

sizes = 100 * np.random.rand(count*2)  # размеры точек

plt.scatter(x, y, s=sizes, alpha=0.125)
plt.show()
```

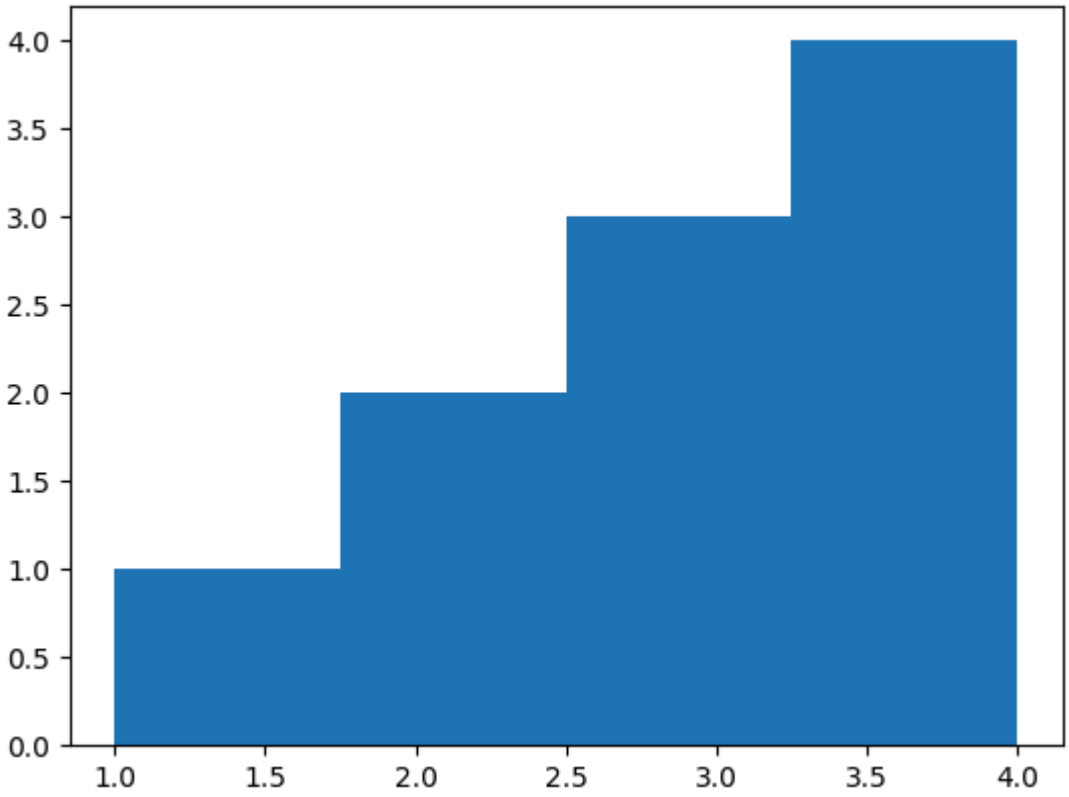


Я, кстати, знаю, что нет слова: "Восмьюпрозрачный", спасибо.

Гистограммы

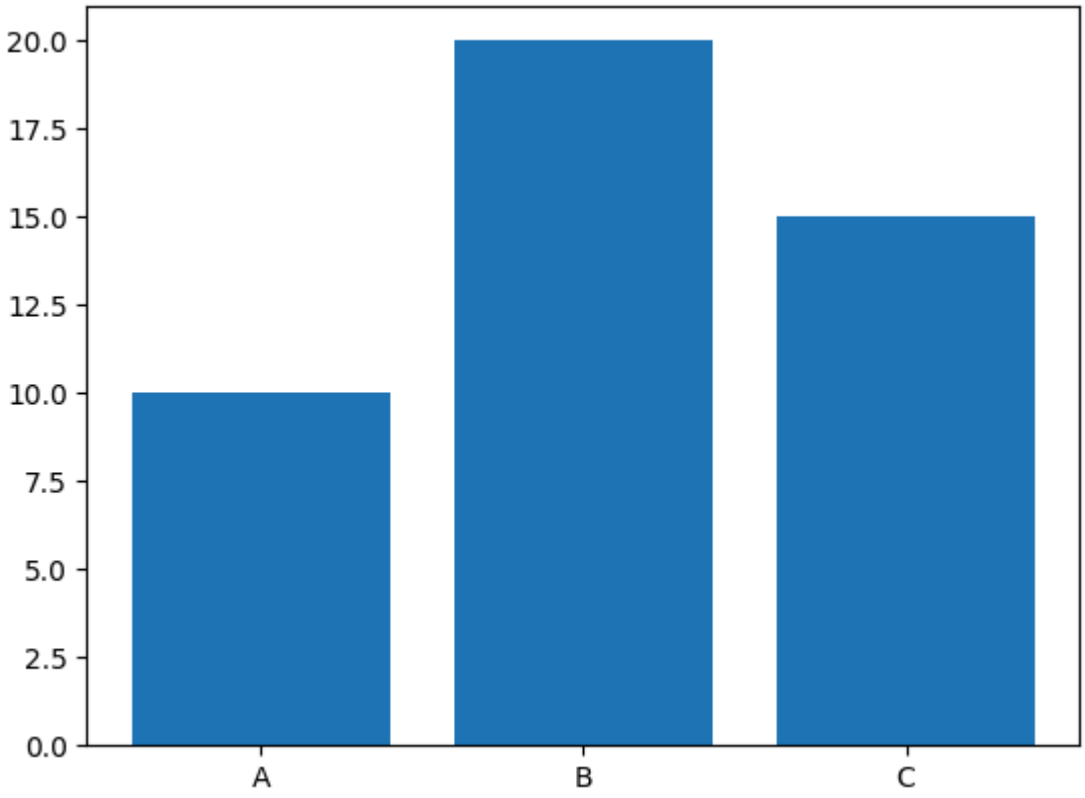
`bins` -- указывает количество столбцов в гистограмме (что такое гистограмма вы должны были проходить в курсе теорвера, поэтому объяснять не буду).

In [40]:
```
fig = plt.figure()
data = [1, 2, 2, 3, 3, 3, 4, 4, 4, 4]
plt.hist(data, bins=4)
plt.show()
```



Столбчатые диаграммы

In [41]:
```
fig = plt.figure()
categories = ['A', 'B', 'C']
values = [10, 20, 15]
plt.bar(categories, values)
plt.show()
```

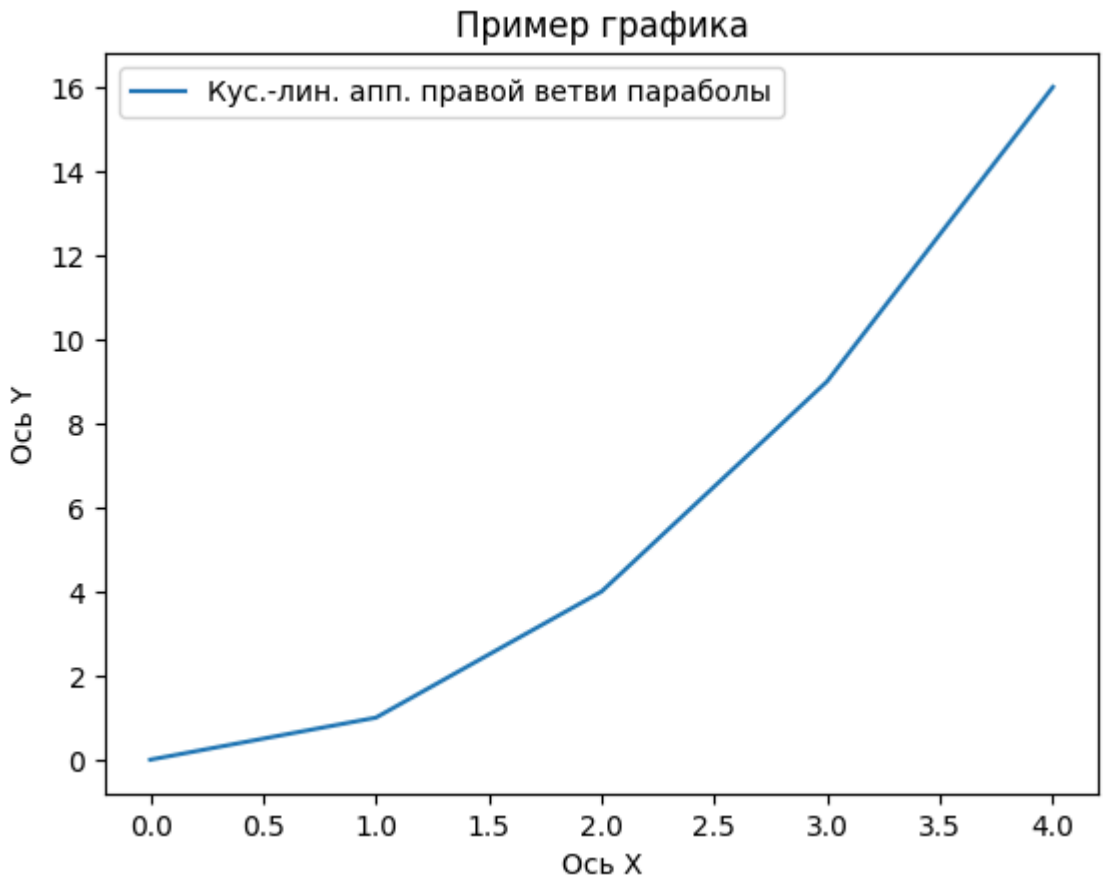


Форматирование графиков

Подписи и легенда

Всё довольно очевидно (прелести самодокументирующегося кода, ага).

In [42]:
```
fig = plt.figure()

x = [0, 1, 2, 3, 4]
y = [0, 1, 4, 9, 16]

plt.xlabel("Ось X")
plt.ylabel("Ось Y")
plt.title("Пример графика")

# label -- для легенды
plt.plot(x, y, label='Кус.-лин. апп. правой ветви параболы')
plt.legend() # легенда добавляется после plot, но до show
plt.show()
```



Красота

> А оно всегда будет таким скучным синим цветом? Можно ли это как-то изменить?

Можно и иногда даже нужно. Я приведу таблицу и ссылки на официальную документацию для Lineplot, покажу пример, а все интересующие вас моменты сможете найти сами, оно всё плюс-минус одинаковое.

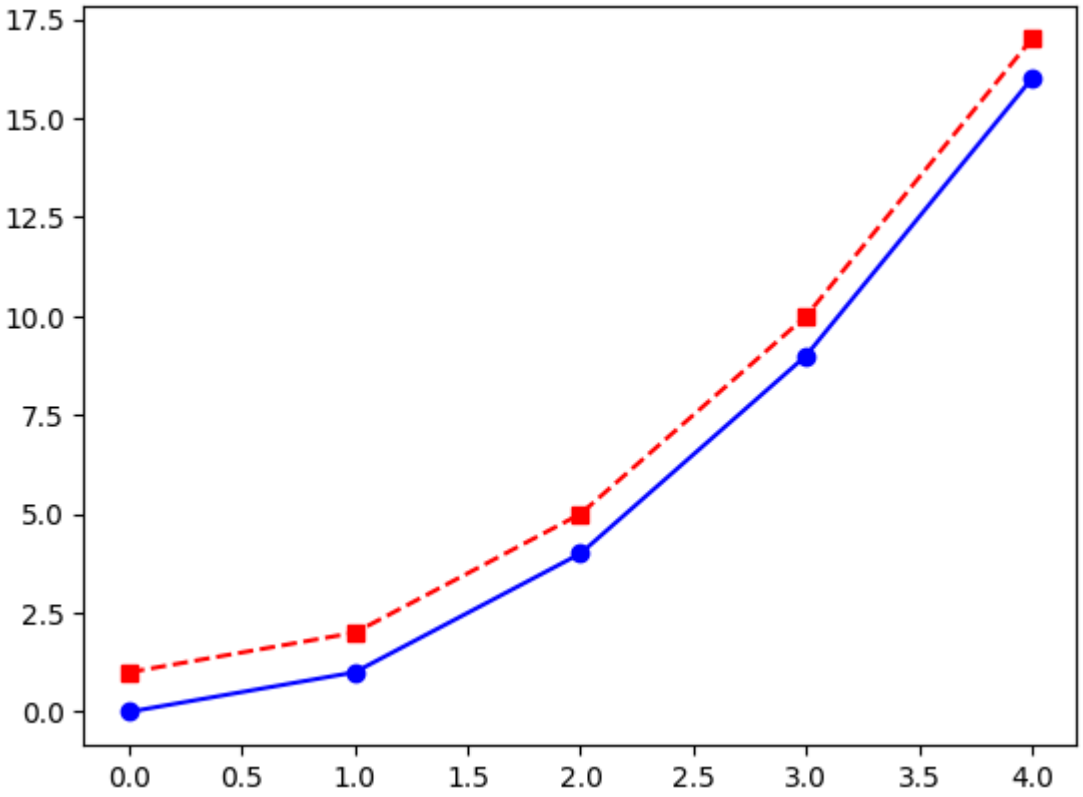
Параметр	Описание	Примеры значений
color	Цвет линии или точек	`'blue'`, `'#FF5733'`, `'r'`
linestyle	Тип линии	`'-'` (сплошная), `'--'` (пунктирная), `'-.'` (штрихпунктирная), `':'` (точечная)
linewidth (lw)	Толщина линии	`1.0`, `2.5`
marker	Тип маркера для точек	`'o'` (круг), `'s'` (квадрат), `'^'` (треугольник), `'D'` (ромб), `'*'` (звезда)
markersize (ms)	Размер маркера	`6`, `8`, `10`
markerfacecolor	Цвет заливки маркера	`'red'`, `'green'`, `'#00FF00'`
markeredgecolor	Цвет контура маркера	`'black'`, `'blue'`
markeredgewidth	Толщина контура маркера	`1.0`, `2.0`

Официальная документация:

- matplotlib.pyplot.plot
- Markers API
- Примеры линий и маркеров (может быть немного душновато)

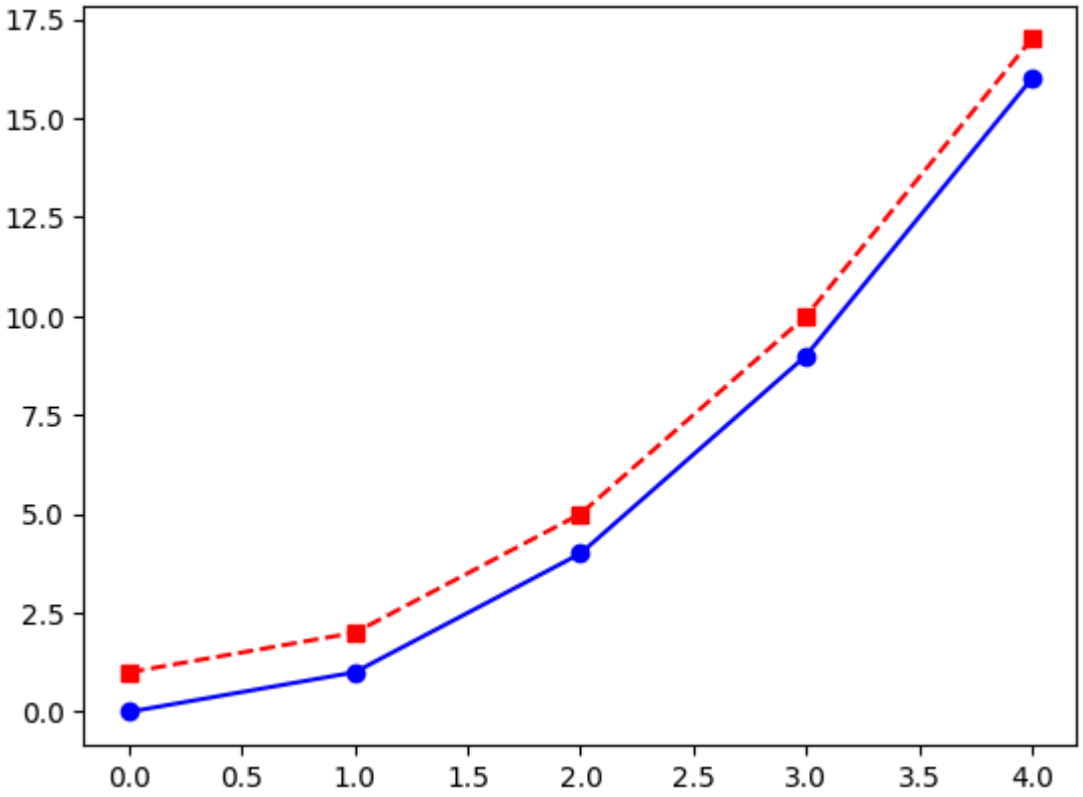
Краткое форматирование

In [43]:
```
fig = plt.figure()

# b = синий
# o = кружок
# - = линия
plt.plot(x, y, 'bo-')

# новый Figure не создаём, т.к. хотим два графика в одном

# r = красный
# s = квадрат
# -- = пунктир
plt.plot(x, np.array(y)+1, 'rs--')
plt.show()
```



Подробное форматирование

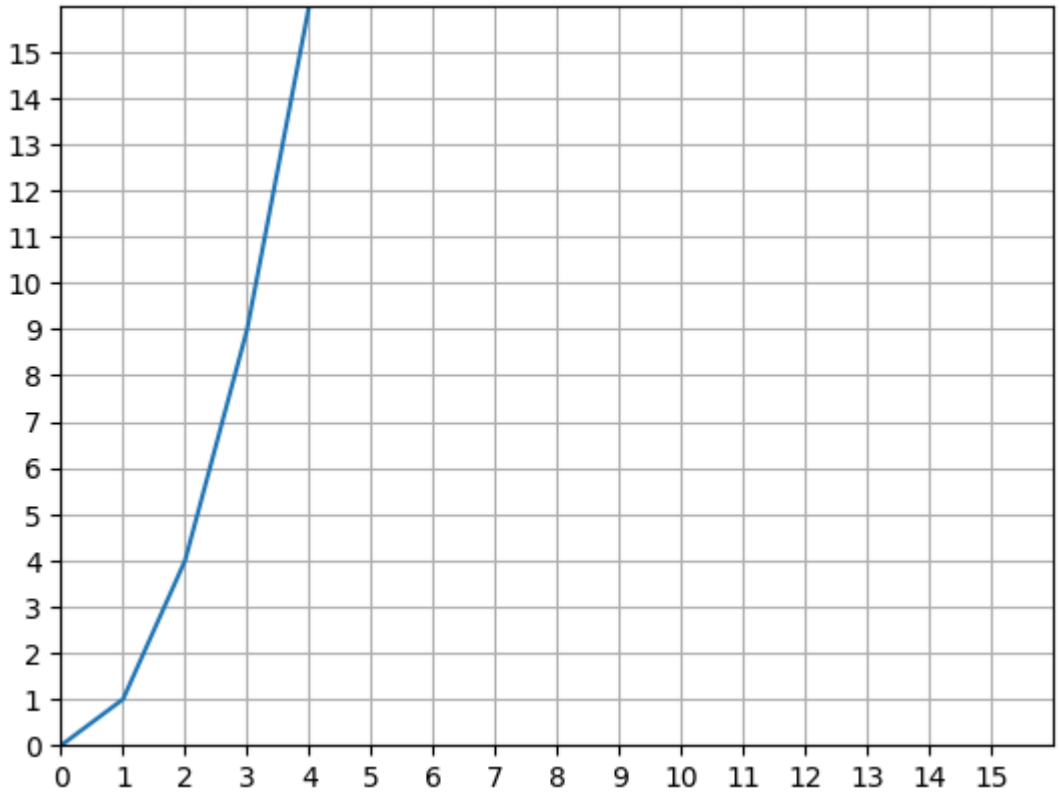
In [54]:
```
fig = plt.figure()

# blue = синий
# o = кружок
# solid = линия
plt.plot(x, y, color='blue', marker='o', linestyle='solid')

# новый Figure не создаём, т.к. хотим два графика в одном

# r = красный
# s = квадрат
# -- = пунктир
plt.plot(x, np.array(y)+1, color='r', marker='s', linestyle='--')
plt.show()
```



Настройка осей

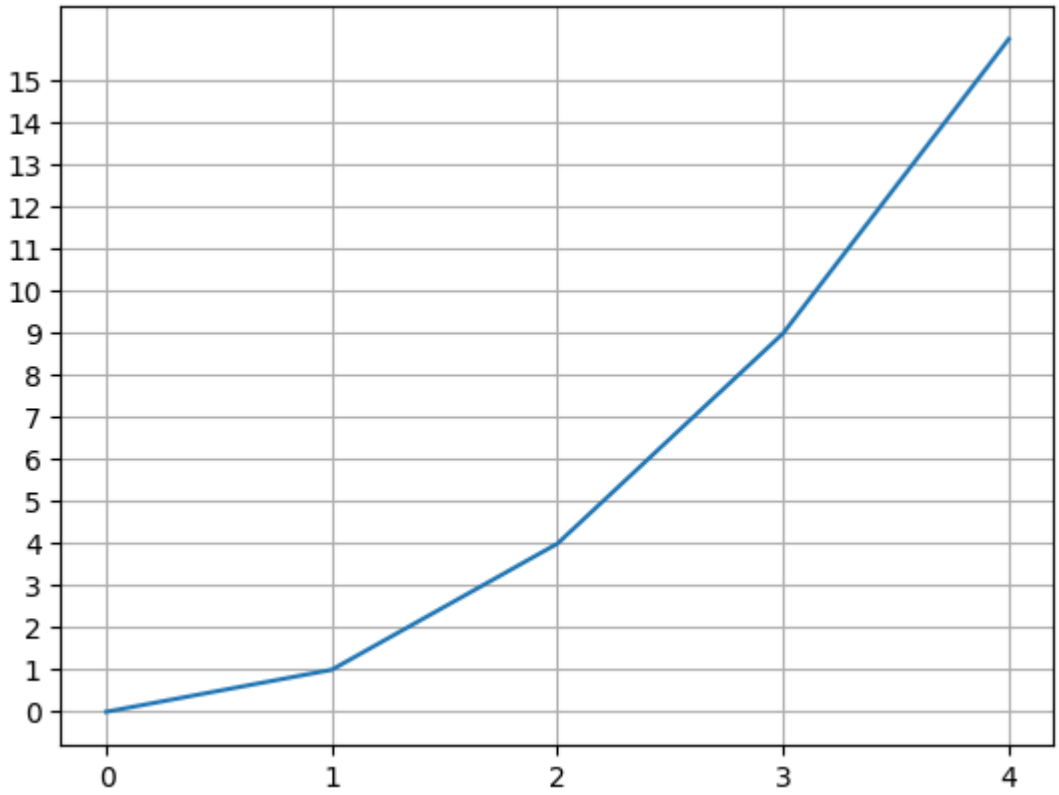
Для задания деления осей используется `plt.xtics` и `plt.ytics`. Для задания минимальных/максимальных значений по осям `xlim` и `ylim`.

In [88]:
```
x = [0, 1, 2, 3, 4]
y = [0, 1, 4, 9, 16]

fig = plt.figure()

plt.xticks(np.arange(0, 16, 1))
plt.yticks(np.arange(0, 16, 1))

plt.xlim(0, 16)
plt.ylim(0, 16)

plt.plot(x, y)
plt.grid()
plt.show()
```



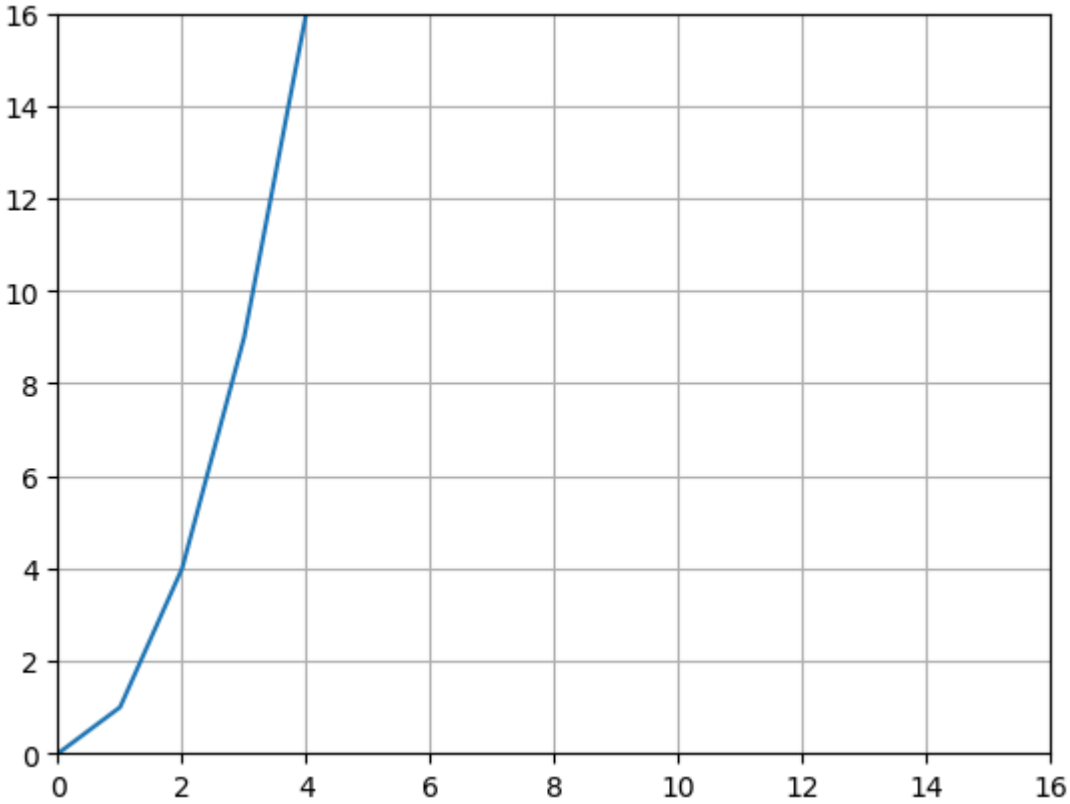
Без указания лимитов.

In [91]:
```
fig = plt.figure()

plt.xticks(np.arange(0, 16, 1))
plt.yticks(np.arange(0, 16, 1))

plt.plot(x, y)
plt.grid()
plt.show()
```



Без указания деления

In [92]:
```
fig = plt.figure()

plt.xlim(0, 16)
plt.ylim(0, 16)

plt.plot(x, y)
plt.grid()
plt.show()
```

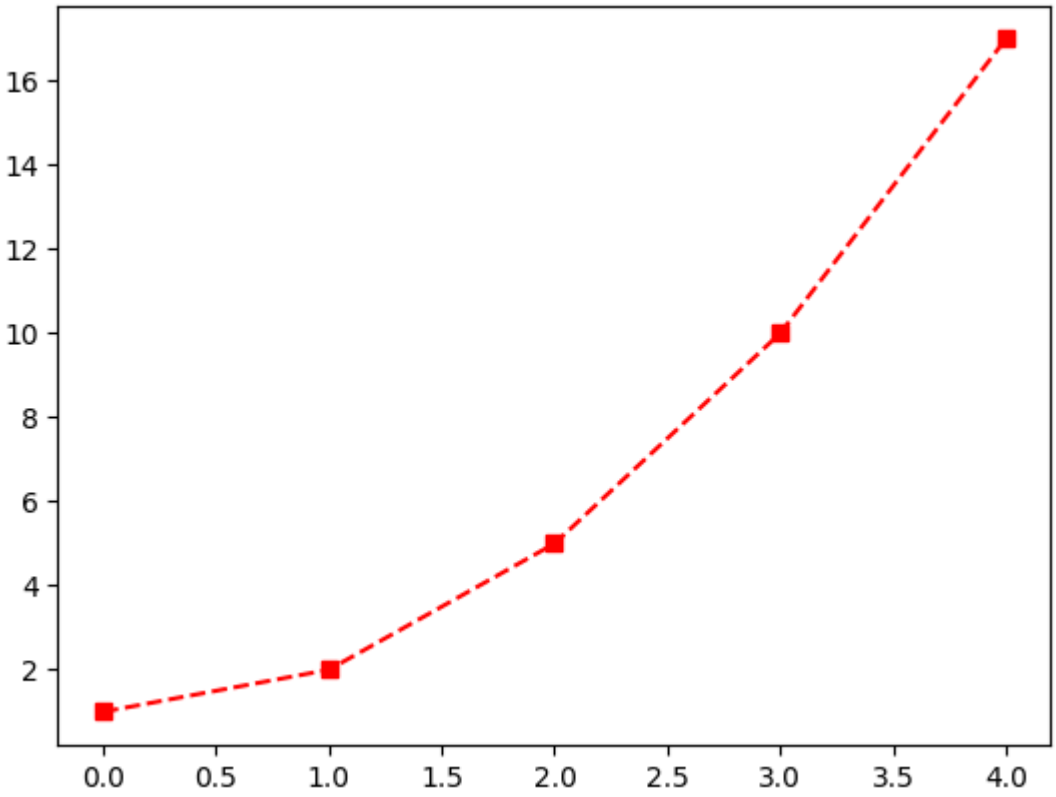


Аннотация

Для любознательных: узнать про `plt.annotate()`.

Сохранение графиков

In [57]:
```
file_name = 'my plot.png'
plt.plot(x, np.array(y)+1, color='r', marker='s', linestyle='--')
plt.savefig(file_name)
```



Несколько графиков на одной картинке

Есть два пути: `plt.subplot` и `plt.subplots`. Выбирем первое --- считай пропало, поэтому рассмотрим только второй.

▶ Спойлер

In [12]:
```
x = np.linspace(0, 10, 100)
```

В следующей *строке* создаём сетку 2x2, игнорируем `fig` (поэтому и не вызываем `plt.figure`, кстати) оставляем axes для расположения графиков, `figsize` задаёт размер картинки.

In [13]:
```
_, axes = plt.subplots(2, 2, figsize=(10, 6))

axes[0, 0].plot(x, np.sin(x)) # 0, 0 -- верхний левый угол
axes[0, 0].set_title("sin(x)")

axes[0, 1].plot(x, np.cos(x)) # 0, 1 -- правый верхний угол
axes[0, 1].set_title("cos(x)")

axes[1, 0].plot(x, np.tan(x))
axes[1, 0].set_title("tan(x)")

axes[1, 1].plot(x, np.sinh(x))
axes[1, 1].set_title("sinh(x)")

plt.tight_layout() # регулирует отступы внутри картинки, меняет размер
plt.show()
```

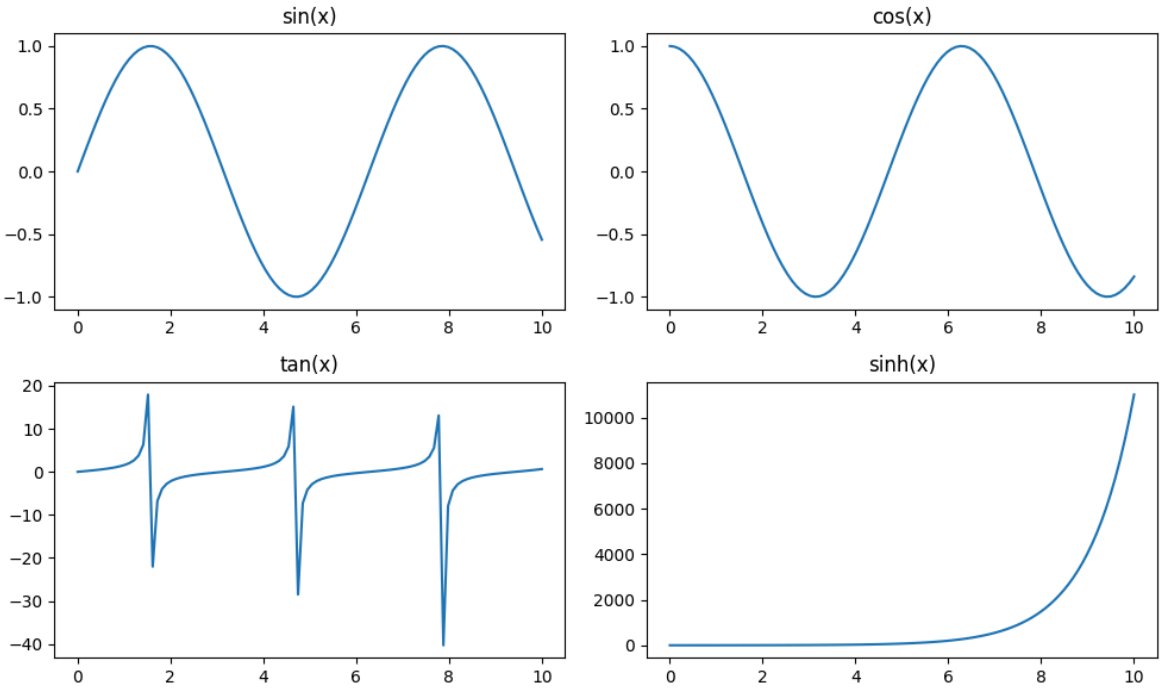
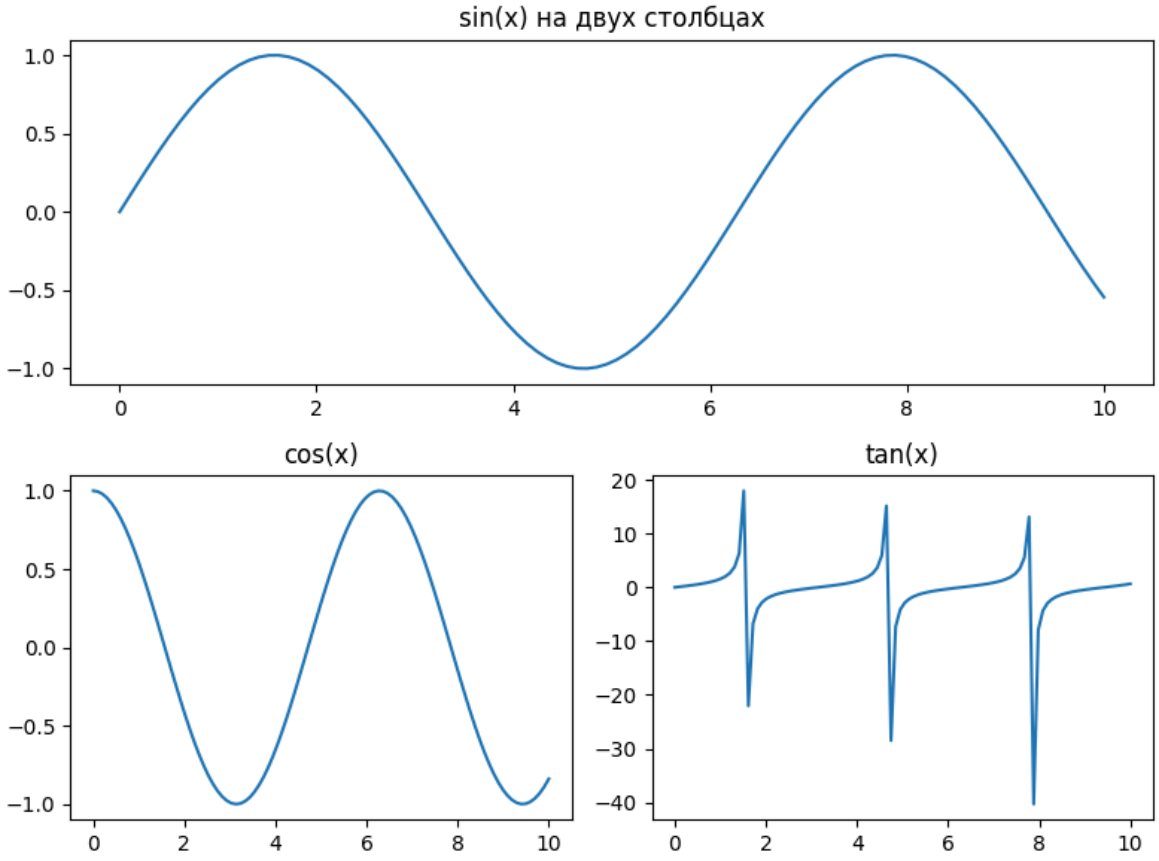


График на несколько ячеек

Можно растянуть один график на несколько ячеек, если необходимо.

In [14]:
```
x = np.linspace(0, 10, 100)
plt.figure(figsize=(8, 6))

fig_cells = (2, 2) # размер "матрицы" для графиков

ax1_place = (0, 0) # место откуда график начинает "расти"
# График, занимающий две ячейки по ширине
# Для использования вертикальных ячеек используется `rowspan`
ax1 = plt.subplot2grid(fig_cells, ax1_place, colspan=2)
ax1.plot(x, np.sin(x))
ax1.set_title("sin(x) на двух столбцах")

ax2_place = (1, 0) # (здесь) месторасположение графика
ax2 = plt.subplot2grid(fig_cells, ax2_place) # займёт только 1 ячейку
ax2.plot(x, np.cos(x))
ax2.set_title("cos(x)")

ax3_place = (1, 1)
ax3 = plt.subplot2grid(fig_cells, ax3_place)
ax3.plot(x, np.tan(x))
ax3.set_title("tan(x)")

plt.tight_layout()
plt.show()
```



Продвинутые вещи

Для любознательных:

- Деления:
  - `set_xticklabels`
  - `set_yticklabels`
- Масштаб
  - `set_xscale`
  - `set_yscale`
- Параметры сетки (`grid`)
- 3D графики
- Анимация (`animation` ; для очень любознательных)

Seaborn

Основные возможности `seaborn`

Перво-наперво `seaborn` можно импортировать.

In [2]:
```
import seaborn as sns
```

Загрузим встроенный датасет `tips` . Там следующие колонки: общий счёт (`total_bill`), чаевые (`tips`), пол (`sex`), курящий (`smoker`), день (`day`), время (`time` ; категория, а не `datetime`) и количество поситителей за столом (`size`). На

этом же датафрейме (да, мы не подключали `pandas`, но `tips` всё равно типа `dataframe`) и покажем поддержку `pandas` в `seaborn`.

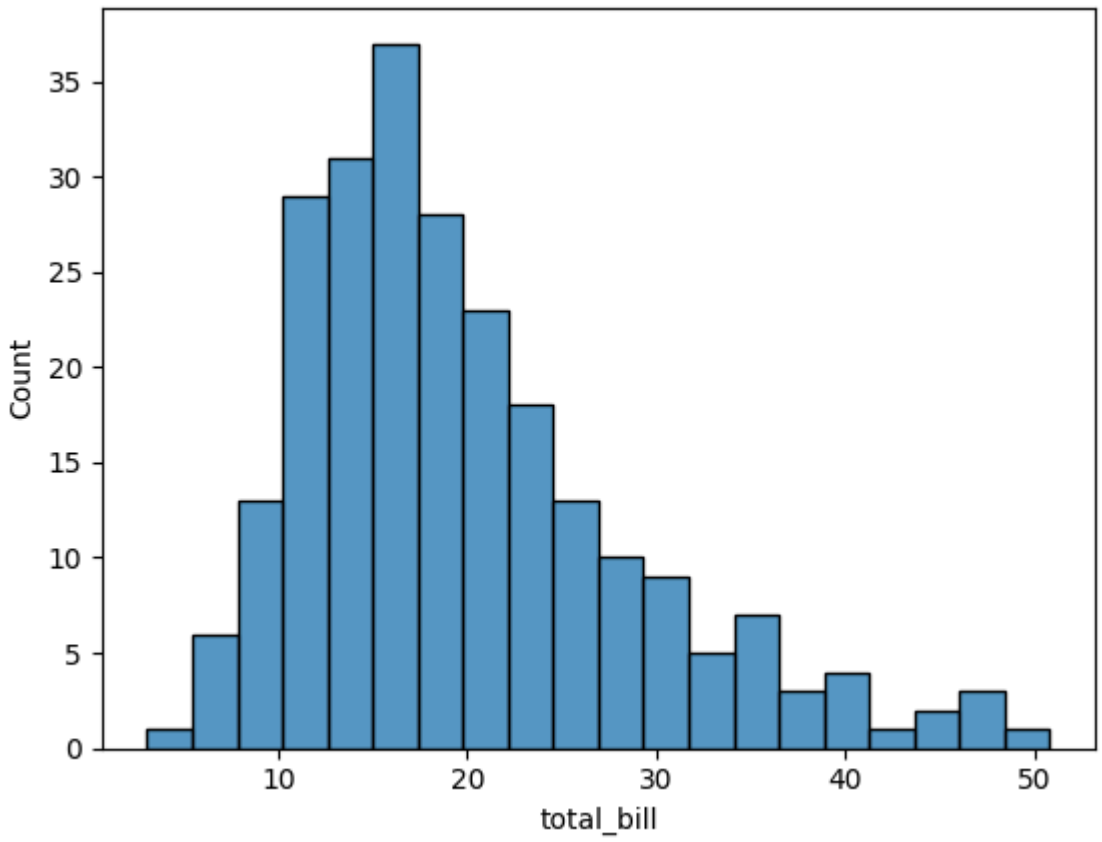
In [3]:
```
tips = sns.load_dataset('tips')
tips.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 244 entries, 0 to 243
Data columns (total 7 columns):
 #   Column      Non-Null Count  Dtype   
---  ------      --------------  -----   
 0   total_bill  244 non-null    float64 
 1   tip         244 non-null    float64 
 2   sex         244 non-null    category
 3   smoker      244 non-null    category
 4   day         244 non-null    category
 5   time        244 non-null    category
 6   size        244 non-null    int64   
dtypes: category(4), float64(2), int64(1)
memory usage: 7.4 KB
```

Гистограмма

Построим гистограмму сумм в общих счетах. По оси `y` у нас количество, по `x` сумма, указанная в счёте (мы передали туда `total_bill`).

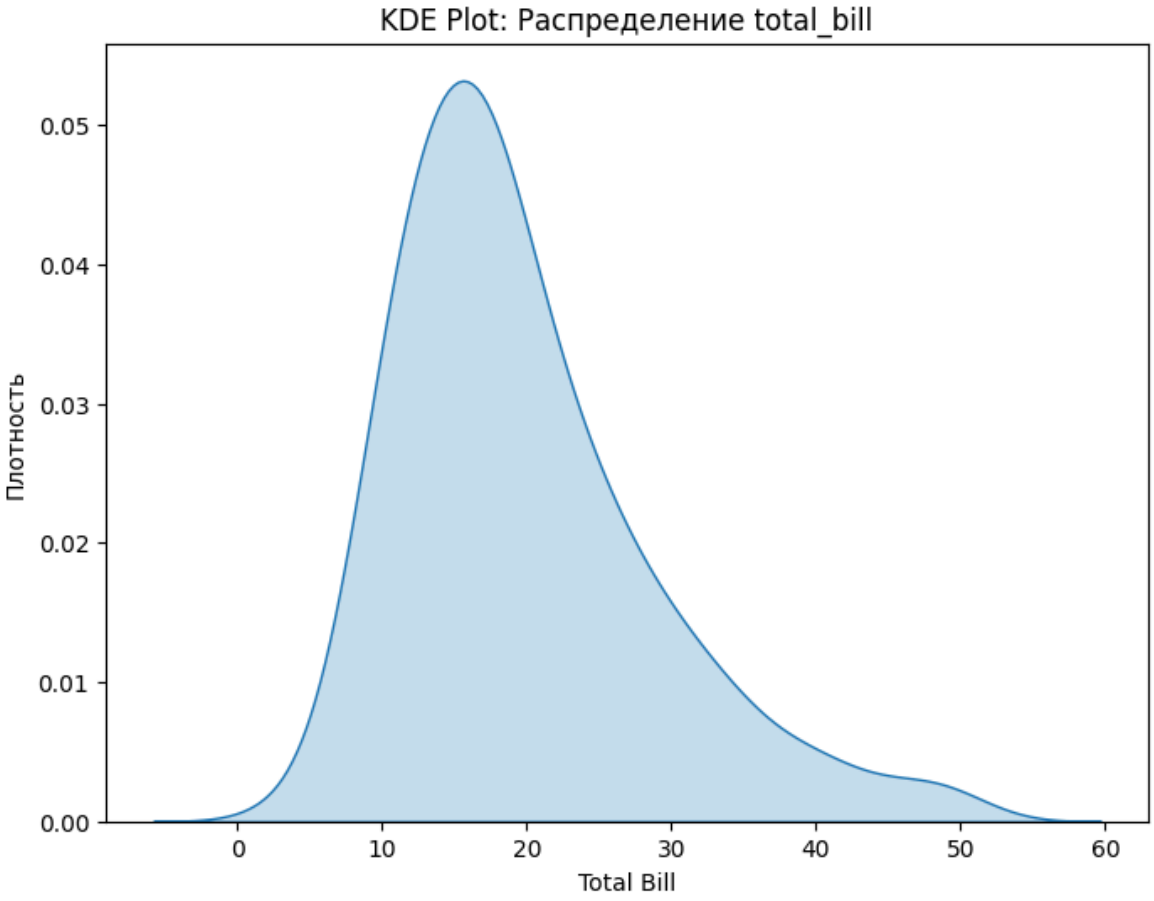
In [9]:
```
sns.histplot(data=tips, x='total_bill', bins=20)
plt.show() # работает и без этой функции, но не забываем про хорошие
           # манеры
```



Оценка плотности

Для оценки плотности распредления можем нарисовать `kdeplot`.

In [21]:
```
plt.figure(figsize=(8, 6))
sns.kdeplot(data=tips, x='total_bill', fill=True)
plt.title('KDE Plot: Распределение total_bill')
plt.xlabel('Total Bill')
plt.ylabel('Плотность')
plt.show()
```



Или `displot`, чтобы нарисовать и гистограмму, и оценку плотности.

In [22]:
```
sns.displot(tips, x='total_bill', bins=20, kde=True, height=6, aspect=1.2)
plt.title('Displot: Распределение total_bill')
plt.xlabel('Total Bill')
plt.ylabel('Частота')
plt.show()
```

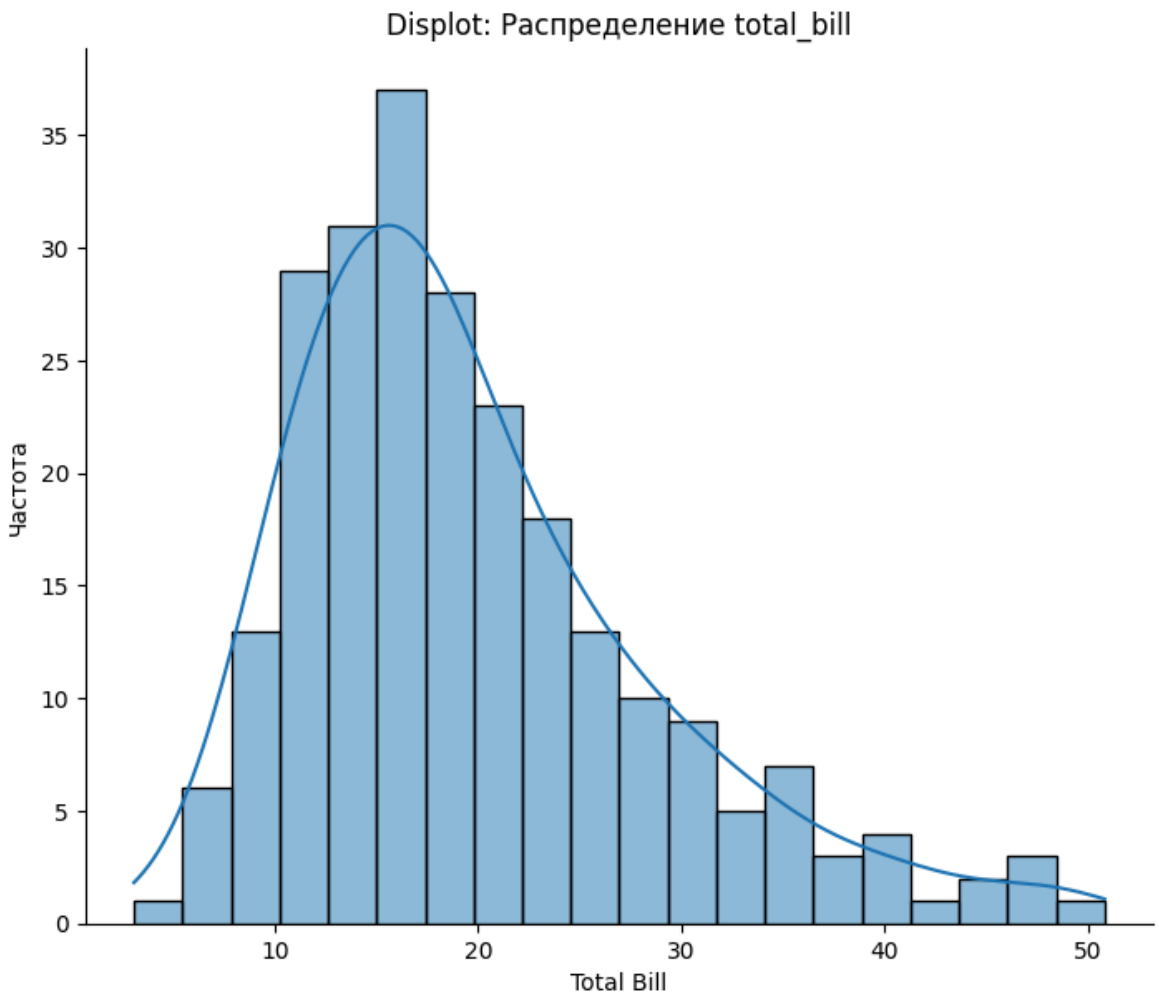
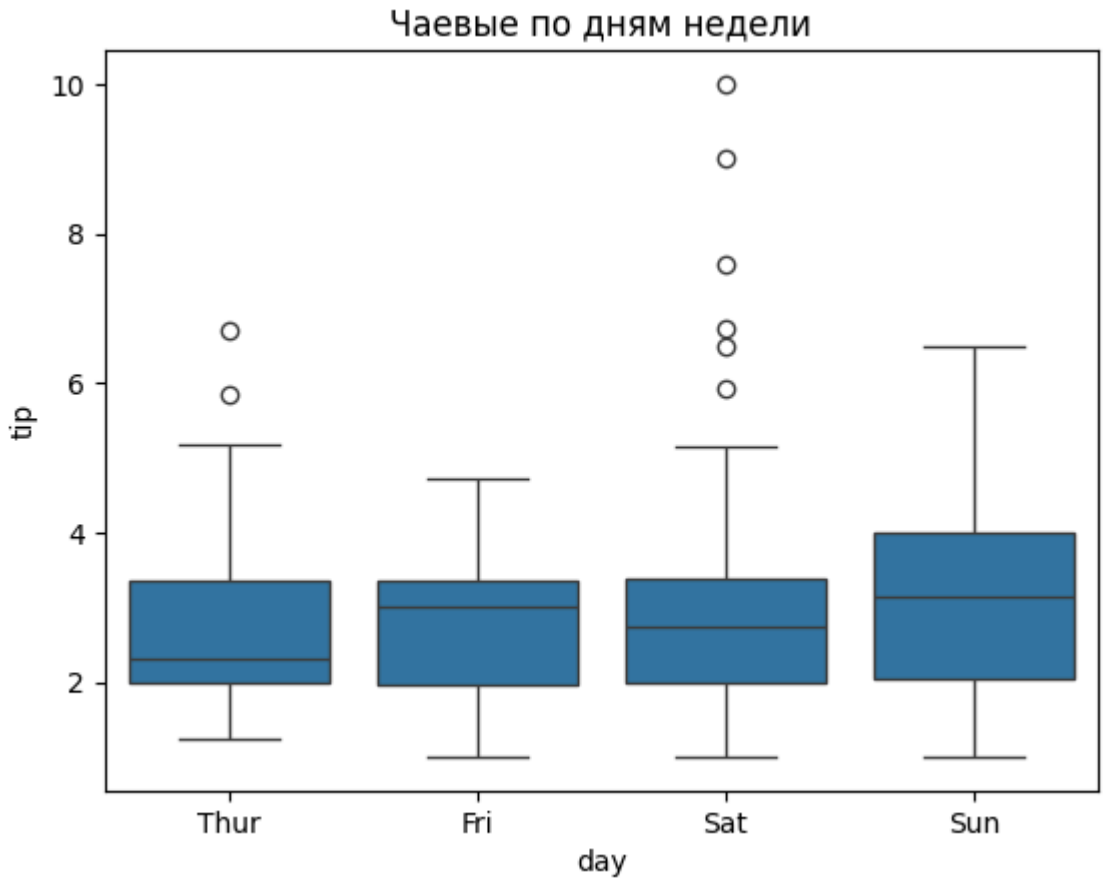


Диаграмма "ящик с усами"

Показывает медиану, квартили, наблюдаемый минимум и максимум, а также выбросы. По оси `x` категориальный признак день недели, по `y` у нас чаевые.

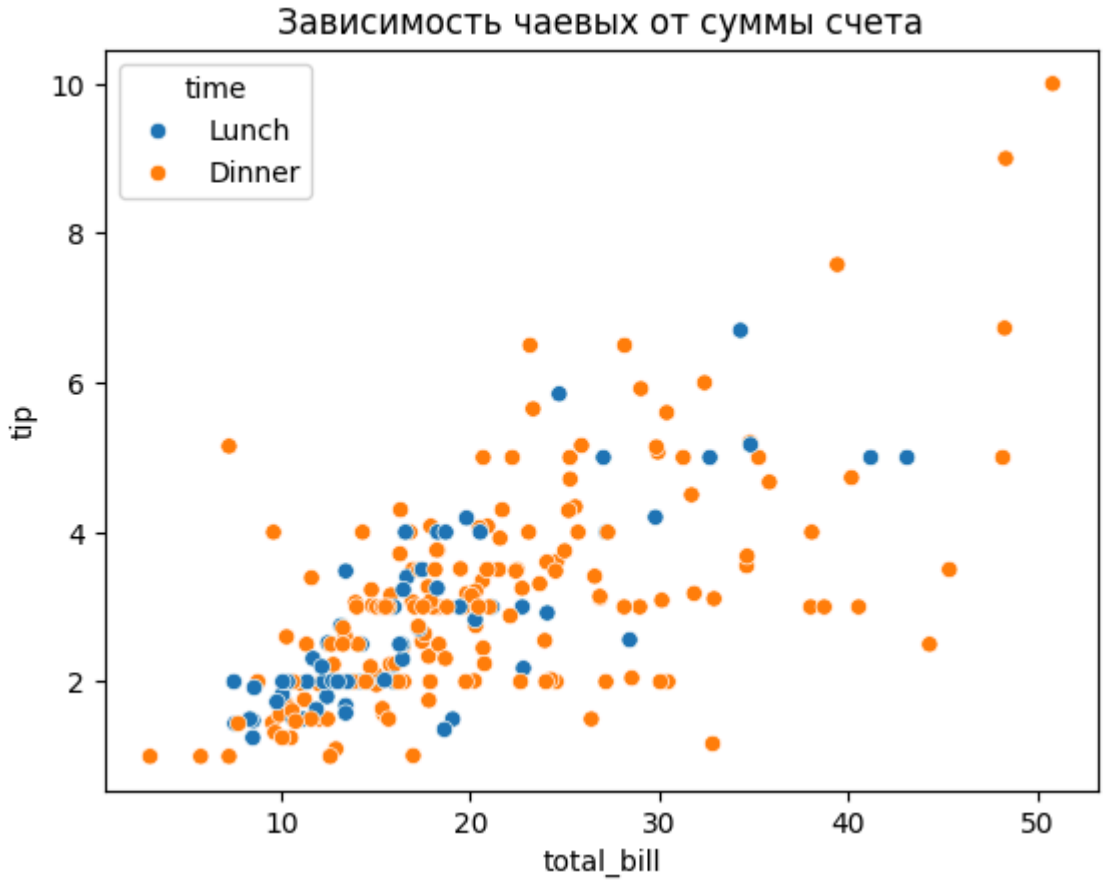
In [10]:
```
sns.boxplot(data=tips, x='day', y='tip')
plt.title('Чаевые по дням недели')
plt.show()
```



Скаттерплот

Построим скаттерплот, по `x` общая сумма счёта (`total_bill`), по `y` чаевые (`tips`), также расскрасим (`hue`) каждую точку в зависимости от времени (`time`): ланч (`Lunch`) или ужин (`Dinner`).

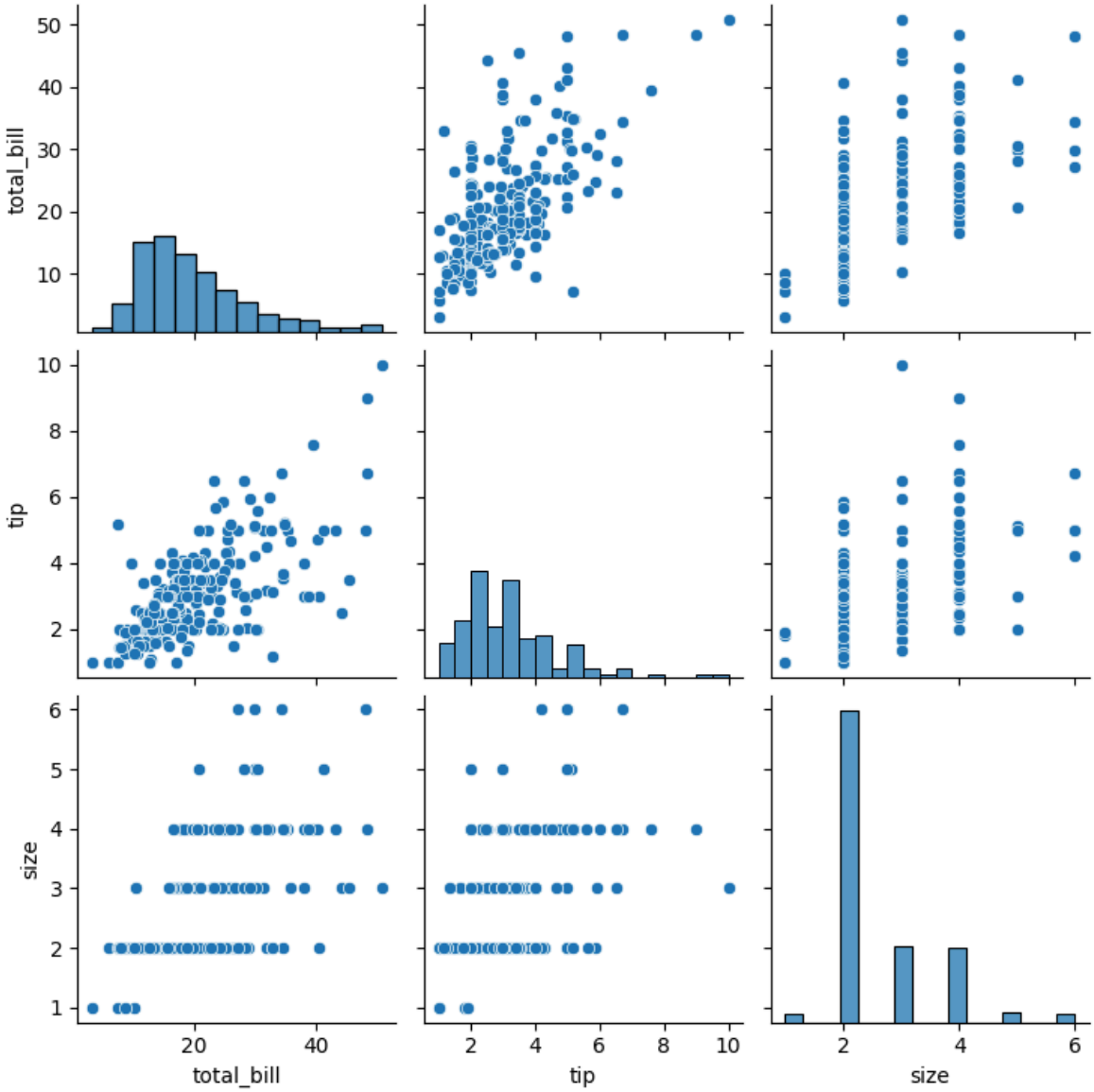
In [11]:
```
sns.scatterplot(data=tips, x='total_bill', y='tip', hue='time')
plt.title('Зависимость чаевых от суммы счета')
plt.show()
```



`Pairplot`

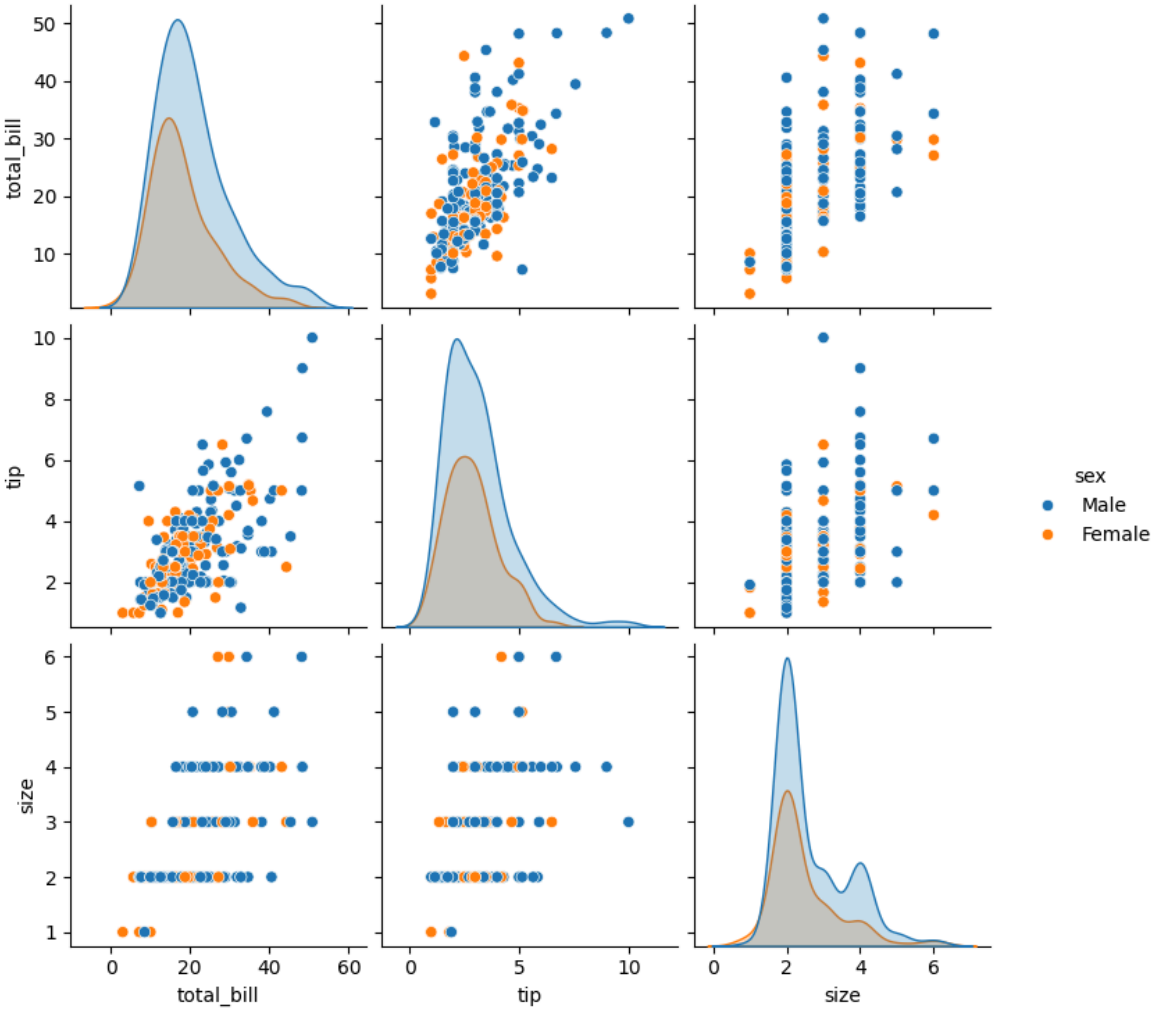
Есть прекрасная и очень удобная штука `Pairplot`. Берутся числовые признаки. По диагонали строится гистограмма каждого признака, внедиагонали скаттерплот одного признака с другим, причём необходимо обратить внимание, что получившаяся матрица симметричная, т.е. если во втором столбце первой строки скаттерплот первого признака от второго (не отключаемся), то в первом столбце второй строки будет скаттерплот второго признака в зависимости от первого.

In [12]:
```
sns.pairplot(tips)
plt.show()
```



Также мы можем расскрасить наши графики в зависимости от пола (`sex`). Ну и про другие модификации, которые были описаны в разделе про `matplotlib` тоже не забывайте, мы же можем маркеры ещё менять и др. вещи делать.

In [29]:
```
sns.pairplot(tips, hue='sex')
plt.show()
```



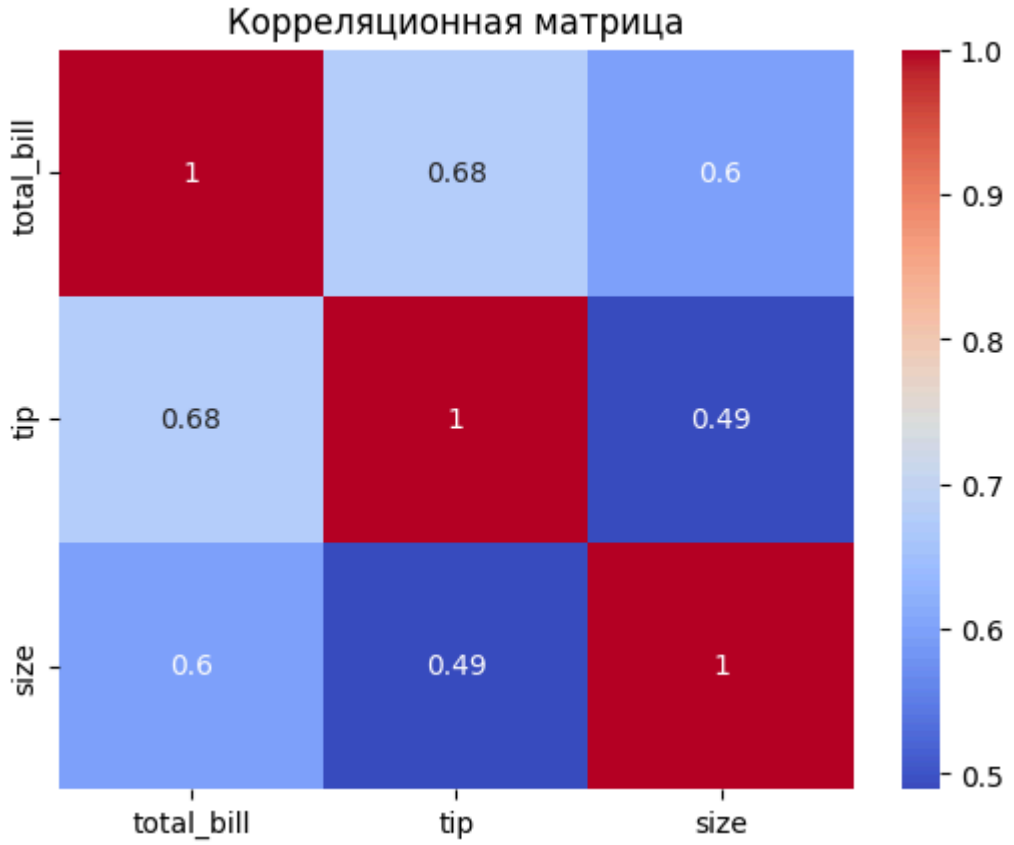
Heatmap / Тепловая карта

Покажем пример тепловой карты на примере корреляционной матрицы. Для построенния корреляционной матрицы её нужно посчитать. Сделать это довольно легко, поскольку `tips` --- это у нас пандасовский датафрейм, а значит содержит методы. Используем метод `corr`.

Почему-то у меня эта функция выдала ошибку, но раньше она сама выбирала числовые признаки и игнорировала категориальные. Мне пришлось немного закостылить и я выбрал числовые признаки с помощью `select_dtypes(include=['number'])`.

Тепловую карту делает тепловой отображение "температуры", чем выше значение, тем теплее цвет и наоборот, чем меньше значение, тем холоднее цвет.

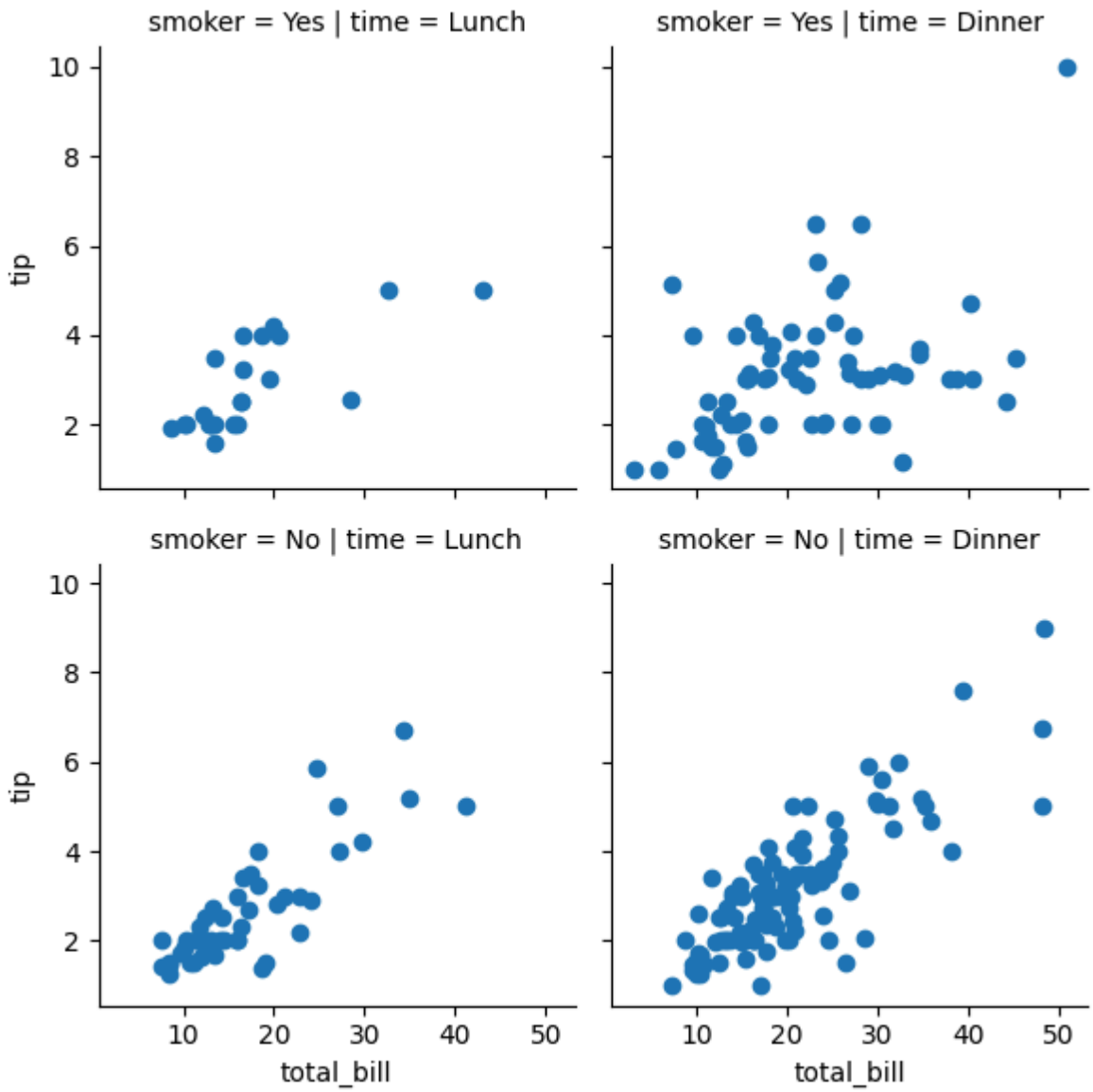
In [19]:
```
corr = tips.select_dtypes(include=['number']).corr()
sns.heatmap(corr, annot=True, cmap='coolwarm')
plt.title('Корреляционная матрица')
plt.show()
```



FacetGrid

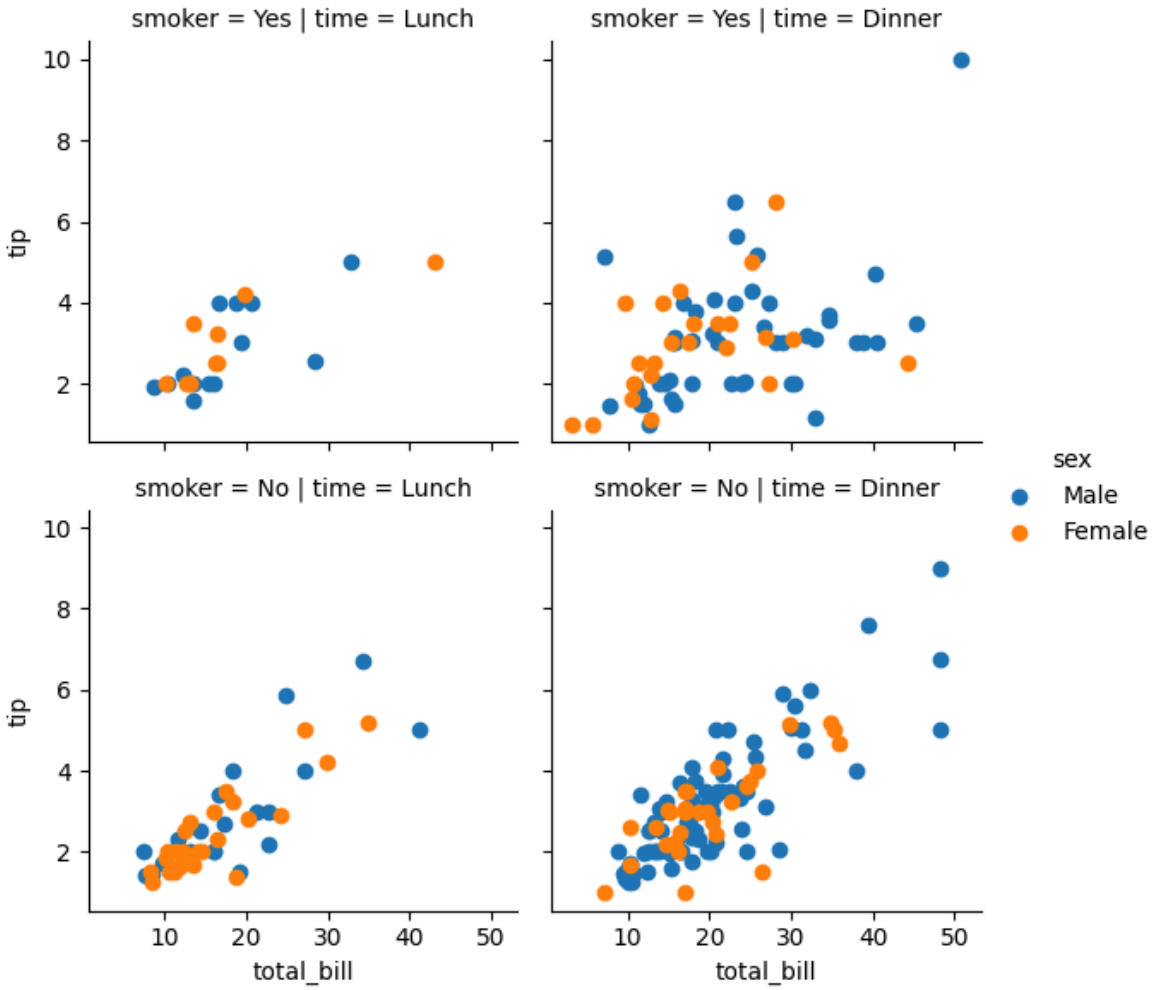
Есть ещё такая шутка для визуализации. Здесь мы взяли две категории (в каждой из корых есть два признака): курящий и обед/ужин и построили скаттерплот для всех возможных четырёх вариантов.

In [28]:
```
g = sns.FacetGrid(tips, col="time", row="smoker")
g.map(plt.scatter, "total_bill", "tip")
plt.show()
```



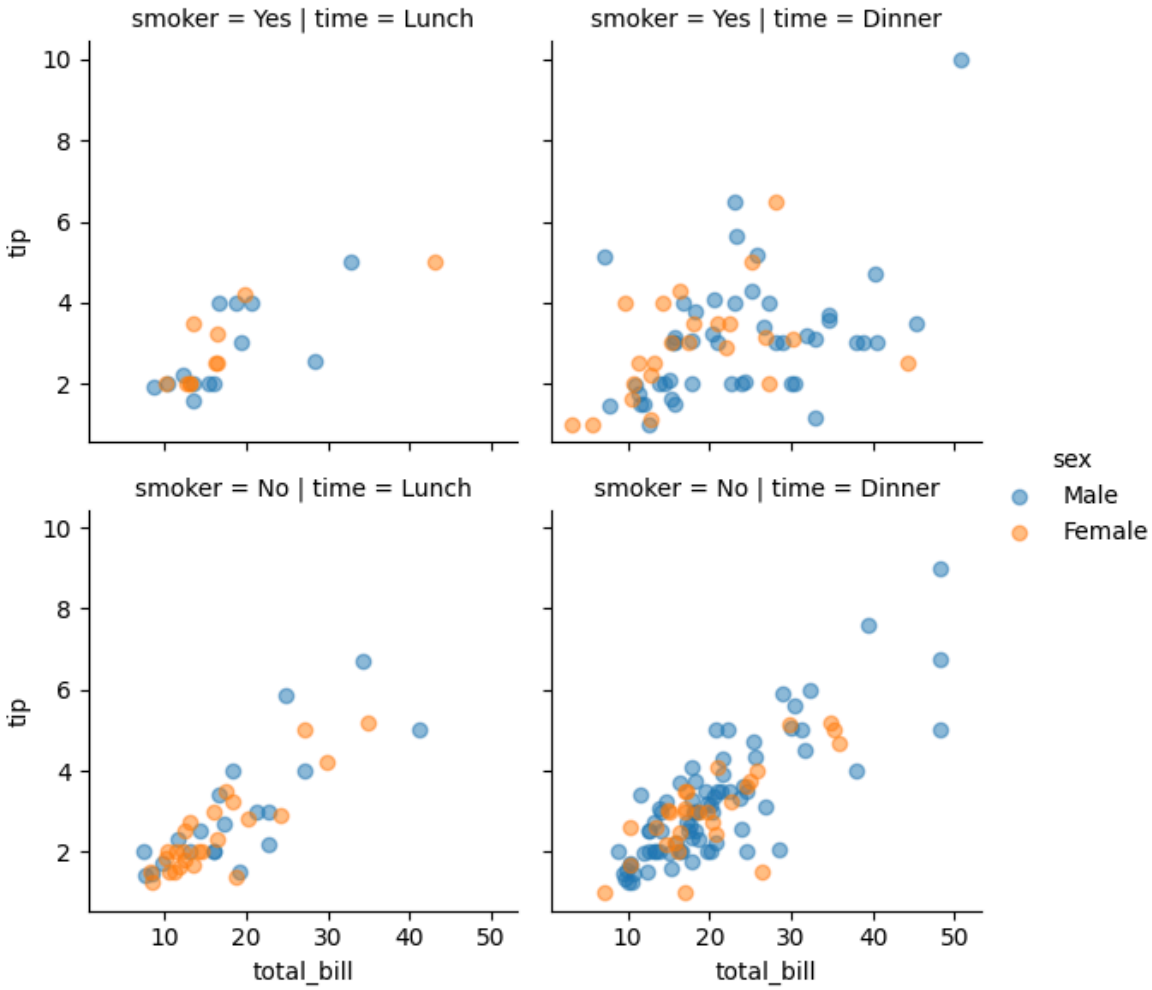
Вместо того, чтобы добавлять ещё один признак (так можно, но выйдет уже 8x3, что не слишком удобно читать) мы можем снова раскрасить каждую точку в зависимости от пола.

In [15]:
```
g = sns.FacetGrid(tips, col="time", row="smoker", hue='sex')
g.map(plt.scatter, "total_bill", "tip")
g.add_legend() # !!! Обратить внимание на добавление легенды
plt.show()
```



Ещё раз напомним про возможность настраивать прозрачность, что позволяет точнее определять скопление точек (при пересечнии маркеров они становятся менее прозрачными).

In [16]:
```
g = sns.FacetGrid(tips, col="time", row="smoker", hue='sex')
g.map(plt.scatter, "total_bill", "tip", alpha=0.5)
g.add_legend()
plt.show()
```

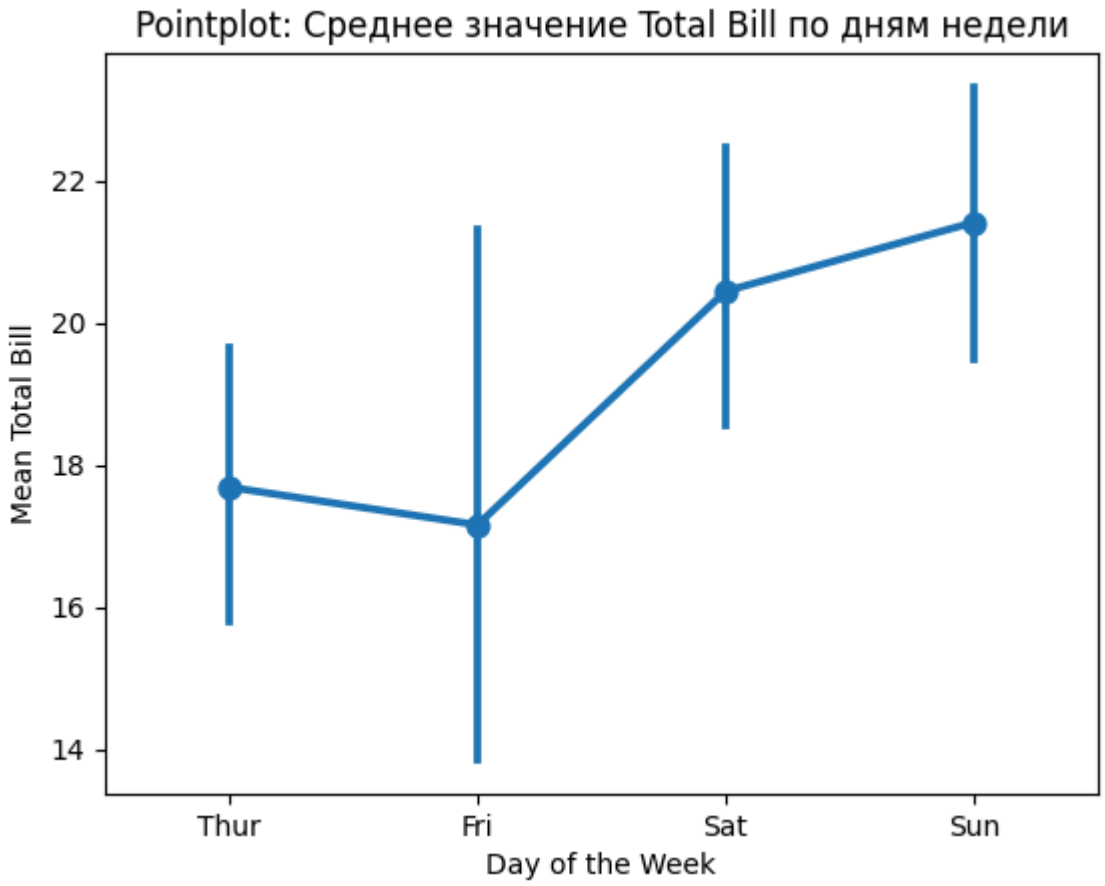


Pointplot

Этот график показывает среднее значение и доверительный интервал этого значения. Удобно для сравнения значения для каждого варианта категориальной переменной.

Для любознательных: прочитать про `errorbar` в `pointplot`.

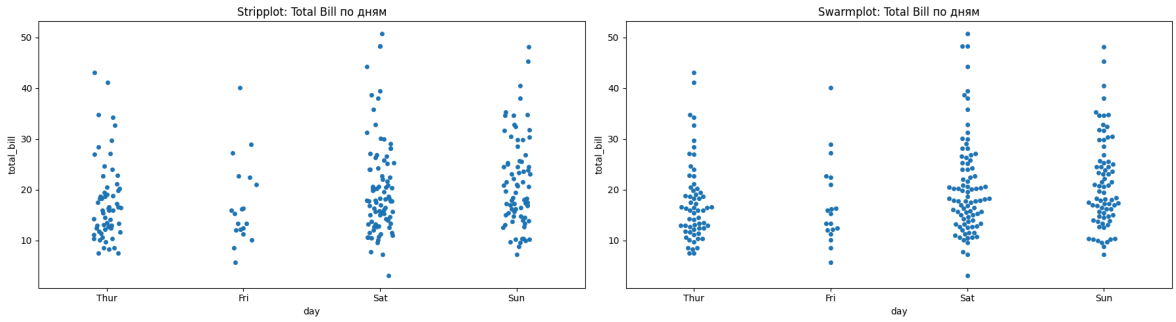
In [17]:
```
sns.pointplot(x="day", y="total_bill", data=tips)
plt.xlabel("Day of the Week")
plt.ylabel("Mean Total Bill")
plt.title("Pointplot: Среднее значение Total Bill по дням недели")
plt.show()
```



Два графика на одном изображении | Striplot vs Swarmplot

Оба этих графиков выполняют одну и ту же функцию, поймите в чём их отличие и когда их нужно применять.

In [5]:
```
fig, axes = plt.subplots(1, 2, figsize=(18, 5))

sns.stripplot(x="day", y="total_bill", data=tips, ax=axes[0])
sns.swarmplot(x="day", y="total_bill", data=tips, ax=axes[1])

axes[0].set_title("Stripplot: Total Bill по дням")
axes[1].set_title("Swarmplot: Total Bill по дням")

plt.tight_layout()
```



Для любознательных

(напоминание, ссылки внутри документа почему-то работают только в html и pdf версии, VS Code не хочет)

1. Здесь
2. Здесь
3. Здесь