KARPOV.COURSES >>>
КОНСПЕКТ

> Конспект > 2 урок > Доверительные интервалы

> Оглавление

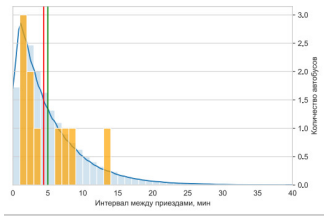
>Доверительные интервалы

- Теория вероятности изучает модели, которые описывают случайности, в то время как математическая статистика учится реальные процессы и явления сводить к математическим моделям.
- Главная задача заключается в том, чтобы узнать что-либо о генеральной совокупности (ГС), что не предоставляется возможным, особенно если генеральная совокупность – всё население земного шара.
- Поэтому мы делаем выборку и оцениваем некие параметры по ней, чтобы в итоге сделать вывод о генеральной совокупности.

Генеральная совокупность	Выборка
Не имеем доступа	Можем «потрогать»
Неизвестные параметры, которые мы хотим оценить	Оцениваем неизвестные параметры

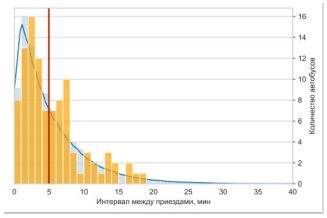
Предположим, хотим измерить средний интервал между автобусами

- Для этого будем стоять на улице и измерять время
- Допустим, мимо нас проехало 11 автобусов
- Средний интервал оказался равным 4.4
- Это точечная оценка– число, которое оценивает параметр генеральной совокупности.
- Но насколько точна эта оценка?



Кажется, что 11 наблюдений недостаточно, чтобы делать выводы о всей совокупности.

- Подождали ещё 90 автобусов
- Теперь средний интервал оказался равным 4.93
- Что стало с оценкой?



Чем больше выборка, тем точнее оценка!

Сгенерируем 100 случайных приездов автобусов(N). Зададим средний интервал равный 5 минутам(t). arrival_times - моменты приезда автобусов.

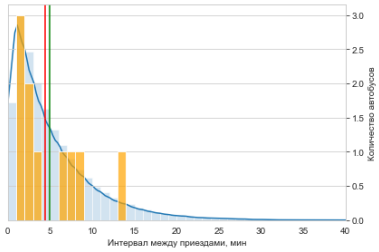
```
N = 100000
t = 5

rand = np.random.seed(42)
arrival_times = (N * t * np.sort(np.random.rand(N))).astype(int)
diff_arrival = np.diff(arrival_times)

diff_arrival[:10]
```

- ->array([2, 1, 3, 6, 1, 8, 1, 2, 7, 13])

```
np.mean(diff_arrival[:10])
```

- ->4.4

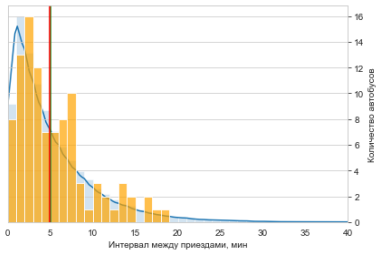
```
sns.histplot(
    diff_arrival, kde=True, bins=np.arange(0, 40, 1),
    stat='probability', alpha=0.2
)

plt.xlim([0, 40])
plt.ylabel('')
plt.grid(None)
plt.yticks([])

plt.xlabel('Интервал между приездами, мин')

plt.twinx()

sns.histplot(
    diff_arrival[:10], bins=np.arange(0, 40, 1),
    color='orange', alpha=0.7)

plt.xlabel('Интервал между приездами, мин')
plt.ylabel('Количество автобусов')
plt.axvline(x=5, color='green')
plt.axvline(x=np.mean(diff_arrival[:10]), color='red')
plt.tight_layout()

plt.savefig('bus10.pdf')
```



Повторим то же самое для 100 наблюдений:

```
sns.histplot(
    diff_arrival, kde=True, bins=np.arange(0, 40, 1),
    stat='probability', alpha=0.2
)

plt.xlim([0, 40])
plt.ylabel('')
plt.grid(None)
plt.yticks([])

plt.xlabel('Интервал между приездами, мин')
```

```
plt.twinx()

sns.histplot(
    diff_arrival[:100], bins=np.arange(0, 40, 1),
    color='orange', alpha=0.7)

plt.xlabel('Интервал между приездами, мин')
plt.ylabel('Количество автобусов')
plt.axvline(x=5, color='green')
plt.axvline(x=np.mean(diff_arrival[:100]), color='red')
plt.tight_layout()

plt.savefig('bus100.pdf')
```




```
np.mean(diff_arrival[:100])
```

- ->4.93

Повторяя эксперимент с еще большим числом наблюдений, заметим, что чем больше наблюдений будем брать, тем ближе выборочное среднее будет приближаться к значению 5.

Но как оценить разброс значений? Для этого нужны доверительные интервалы.

Доверительный интервал – интервал, который с заданной вероятностью покрывает оцениваемый параметр генеральной совокупности.

Доверительный интервал учитывает, что выборка могла быть небольшого размера и плохо оценивать параметр, либо имела большую дисперсию и тогда доверительный интервал будет более широким.

Таким образом мы получаем бóльшую степень уверенности в нашей оценке.

Пример:

• Средний интервал — «от 4.9 до 5.1 минуты с 95% уровнем доверия»

- Средний интервал — «от 4.9 до 5.1 минуты с 95% уровнем доверия»

Таким образом показываем и оценку, и шансы попасть в неё.

Уровень доверия (β) – вероятность, с которой доверительный интервал покрывает оцениваемый параметр.

На практике обычно берут уровень доверия 95% или 99%

Выбор разного уровня «точности» даёт разные интервалы - если хотим гарантировать большую точность, то интервал нужен шире.

Уровень значимости (α= 1 - β) – вероятность, с которой значение параметра не попадает в доверительный интервал.

Уровень доверия в 95% означает, что если бесконечно повторять эксперимент и каждый раз строить доверительный интервал, то истинное значение параметра в 95% случаев будет оказываться внутри данного интервала.

Доверительным интервалом параметра θ с уровнем доверия β называют интервал (l, r) такой, что

$P(l <= \theta <= r) >= \beta$

,где $l(x), r(x)$ – некоторые функции от выборки.

Доверительный интервал для интервала между автобусами с уровнем доверия 95% — интервал между 4.9 и 5.1, то есть:

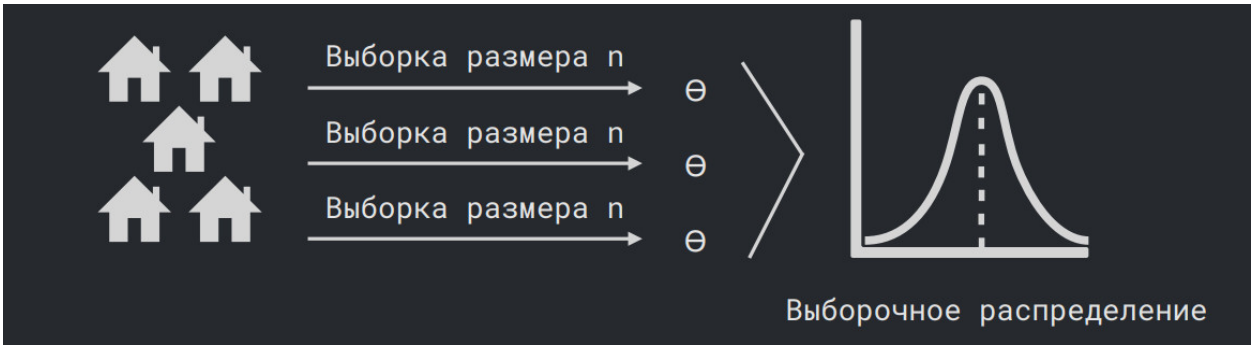
$\mathbb{P}(4.9 \leq t \leq 5.1) = 0.95$

Цель — чтобы этот интервал был максимально узким.

>Методы оценки доверительных интервалов

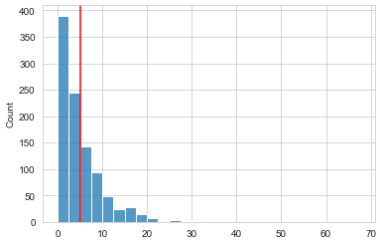
Наивный подход

Взять из генеральной выборки N выборок размера n, оценить по ним выборочное распределение статистики Θ (например среднего) эмпирически



Вернемся к примеру с автобусами:

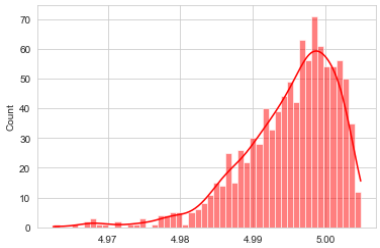
```
t = 5

def get_samples(size):
    arrival_times = (size + 1) * t * np.sort(np.random.rand((size + 1)))
    return np.diff(arrival_times)

data = get_samples(size=1000)
sns.histplot(data, bins=np.arange(0, 70, 2.5))
plt.axvline(x=np.mean(data), color='red')
plt.show()
```



А теперь будем генерировать в цикле такие выборки 1000 раз и считать по ним выборочные средние:

```
means = []
size = 1000
n_iter = 1000
```

```
for _ in range(n_iter):
    samples = get_samples(size=size)
    mean = np.mean(samples)
    means.append(mean)

sns.histplot(means, bins=50, kde=True, color='red', alpha=0.5)
plt.show()
```



Бутстрэп

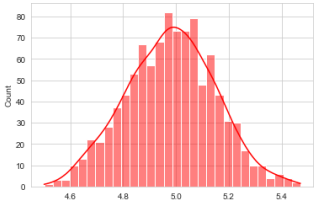
Бутстрэп (Bootstrap) — техника для оценивания распределений статистик с помощью многократной генерации выборок на базе имеющийся выборки.

- $X_1, X_2, \ldots, X_N$ – псевдовыборки размера N из Xn
- $\Theta_1, \Theta_2, \ldots, \Theta_N$ – значения статистик на псевдовыборках
- $F_{\Theta}^{boot}(x)$ – бутстрэп-распределение Θn
- По $F_{\Theta}^{boot}(x)$ можно оценивать доверительные интервалы

Посмотрим, как это работает в коде.

Допустим, мы имеем некоторую выборку. Построим гистограмму распределения значений в этой выборке.

```
size = 1000
data = get_samples(size=size)

means = []
n_iter = 1000

for _ in range(n_iter):
    samples = np.random.choice(data, size=size, replace=True)
    mean = np.mean(samples)
    means.append(mean)

sns.histplot(means, bins=30, kde=True, color='red', alpha=0.5)
plt.show()
```



Мы получили бутстрэп распределение F_θ^{boot}. На основе данного распределения мы уже можем построить доверительный интервал.

Плюсы и минусы построения доверительного интервала через bootstrap:

+ Не требуется знаний о распределениях
+ Можно оценивать любую статистику (медиана, среднее, квантили и прочее)
- Медленно – требуется много времени на генерацию выборок
- Плохо работает для статистик, зависящих от малого числа объектов выборки

>Доверительный интервал через квантили

Идея:

- Мы хотим построить доверительный интервал с уровнем доверия 95% (0.95)
- Отрежем с каждой стороны по 2.5% (0.025) самых экстремальных значений
- Нам понадобится понятие **квантиля**.

α квантиль (quantile) – значение выборки, которое больше α части выборки (например 0.3 квантиль – значение, меньше которого 30% выборки)

Другими словами, некая случайная величина не превосходит квантиль α с вероятностью α:

$P(x <= X_\alpha) = F(X_\alpha) = \alpha$

$F(z)$ – функция распределения случайной величины

Если мы хотим построить доверительный интервал с уровнем доверия 1 - α, нам нужно найти l и r – границы интервала.

$P(l <= \theta <= r) >= 1 - \alpha$

• Как связать α с границами интервала?

• Нужно представить эти границы через квантили распределения θ

• Для квантилей $\frac{\alpha}{2}$ и $\frac{1-\alpha}{2}$ будет справедливо следующее равенство:

$$\mathbb{P}(\theta \leq \Theta_{\frac{\alpha}{2}}) = \frac{\alpha}{2} \qquad \mathbb{P}(\theta \leq \Theta_{1-\frac{\alpha}{2}}) = 1 - \frac{\alpha}{2}$$

Объединим эти выражения и посчитаем, с какой вероятностью параметр θ будет лежать в интервале $[\theta_{\frac{\alpha}{2}}, \theta_{\frac{1-\alpha}{2}}]$:

$$\mathbb{P}(\Theta_{\frac{\alpha}{2}} \leq \theta \leq \Theta_{1-\frac{\alpha}{2}}) =$$
$$= \mathbb{P}(\theta \leq \Theta_{1-\frac{\alpha}{2}}) - \mathbb{P}(\theta \leq \Theta_{\frac{\alpha}{2}}) =$$
$$= 1 - \frac{\alpha}{2} - \frac{\alpha}{2} = 1 - \alpha$$

- **Предсказательный интервал** – интервал, в который попадает случайная величина с нужной нам вероятностью.

$P(\Theta_{\frac{\alpha}{2}}) <= \theta <= \Theta_{1-\frac{\alpha}{2}}$

- Квантиль определяют также как обратную функцию от распределения:

$\Theta_\alpha = F_\theta^{-1}(\alpha)$

- Посчитаем выборочные квантили бутстрэп распределения и получим интервал:

$$\mathbb{P}\left((F^{boot}_{\hat{\Theta}_n}(\frac{\alpha}{2}))^{-1} \leq \theta \leq (F^{boot}_{\hat{\Theta}_n}(1-\frac{\alpha}{2}))^{-1}\right) \approx 1 - \alpha$$

$$\left((F^{boot}_{\hat{\Theta}_n}(\frac{\alpha}{2}))^{-1}, (F^{boot}_{\hat{\Theta}_n}(1-\frac{\alpha}{2}))^{-1}\right)$$

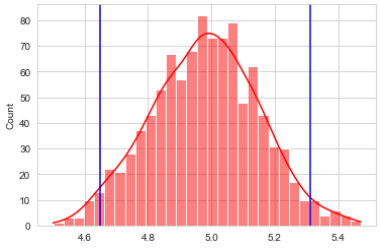
Вернемся к коду и построим 95% доверительный интервал для нашего бутстрэп распределения:

```
alpha = 0.05
ci = np.quantile(means, q=[0.025, 0.975])
ci
```

- ->array([4.65009086, 5.31194005])

```
sns.histplot(means, bins=30, kde=True, color='red', alpha=0.5)
plt.axvline(x=ci[0], color='blue')
```

```
plt.axvline(x=ci[1], color='blue')
plt.show()
```



Также можно воспользоваться встроенным в библиотеку scipy.stats методом bootstrap:

```
from scipy.stats import bootstrap
bootstrap(
    data=(data,),
    statistic=np.mean,
    confidence_level=0.95,
    n_resamples=1000,
    method='percentile',
).confidence_interval
```

- ->ConfidenceInterval(low=4.657233480111648, high=5.307853154781098)

>Виды распределений случайных величин

Дискретные случайные величины – величины, имеющие счётное множество значений.

Определяется вероятность каждого их возможных значений.

Примерами распределений дискретных величин являются:

· Распределение Бернулли

· Биноминальное распределение

· Распределение Пуассона

· Равномерное распределение

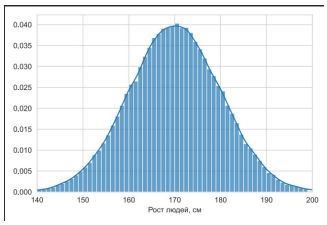
Непрерывные случайные величины – величины, которые имеют вещественное множество значений.

· Каждое отдельное значение имеет нулевую вероятность, но есть вероятность получить значение в некотором интервале.

· Если ширина интервала стремится к нулю, получается плотность распределения:

$$\rho(x) = \lim_{\delta \to 0} \mathbb{P}(x < \theta < x + \delta)$$

· Этой плотностью распределения и описывают распределения случайных непрерывных величин.



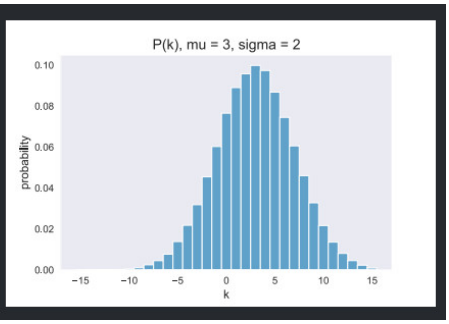
Нормальное распределение — симметричное распределение непрерывной случайной величины.

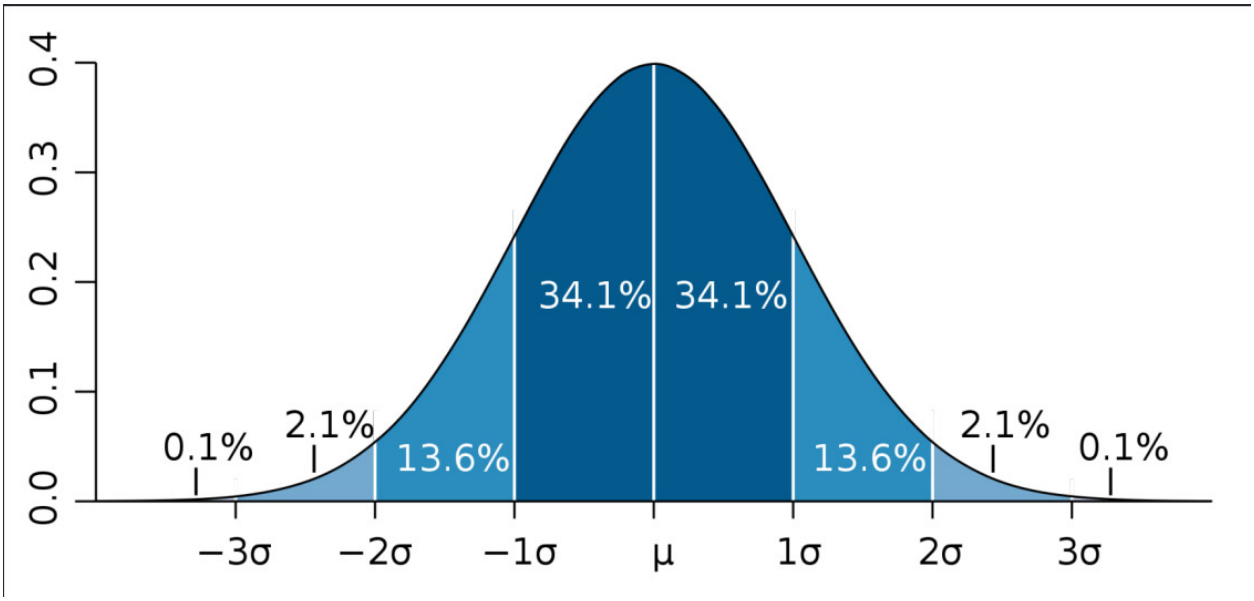
Характеризуется параметрами μ – математическое ожидание (expectation),

σ – среднеквадратичное отклонение (standard deviation).

$$\rho(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

$$\mathbb{N}(\mu, \sigma^2)$$





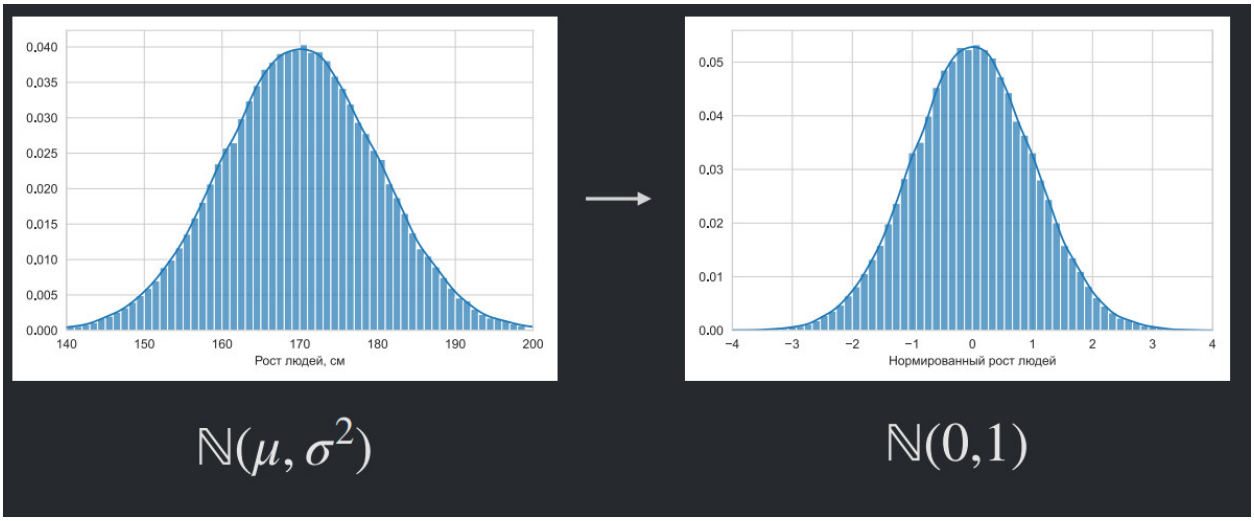
Распределение значений в нормальном распределении

Стандартное распределение — нормальное распределение, в котором

$\mu = 0, \sigma = 1$

```
np.random.seed(125)
hh = np.random.normal(loc=170, scale=10, size=100000)

new_hh = (hh - np.mean(hh)) / np.std(hh)
```

```
sns.histplot(new_hh, kde=True, bins=np.linspace(-4, 4, 61),
             stat='probability', alpha=0.7)

plt.xlim([-4, 4])
plt.ylabel('')

plt.xlabel('Нормированный рост людей')

plt.tight_layout()
plt.savefig('normal_h_normed.pdf')
```



>Центральная предельная теорема (ЦПТ)

· Если мы возьмем N независимых одинаково распределенных случайных величин с матожиданием μ и среднеквадратичным отклонением σ

· Сложим эти случайные величины в S_N (сложим значения на всех объектах)

· Получим нормальное распределение S_N

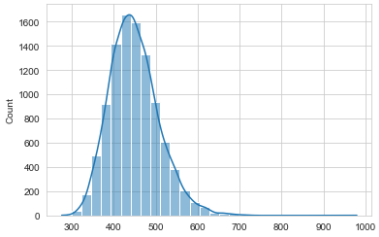
$$\frac{S_N - \mu N}{\sigma\sqrt{N}} \to \mathbb{N}(0,1)$$

· При этом не важно, из какого распределения были наши случайные величины!

· Это значит, что если мы возьмем сумму случайных величин из некоторого одного распределения, то сможем узнать, как распределена эта сумма

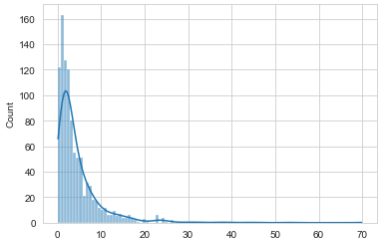
Рассмотрим на примере. Будем делать 1000 выборок размера 100 и подсчитывать суммы случайных величин и посмотрим на распределение этих сумм:

```
sums = []
size = 100
n_iter = 10000

for _ in range(n_iter):
    samples = np.random.normal(mean=2, sigma=5, size=size)
    sum_ = np.sum(samples)
    sums.append(sum_)

sns.histplot(sums, bins=30, kde=True)
plt.show()
```

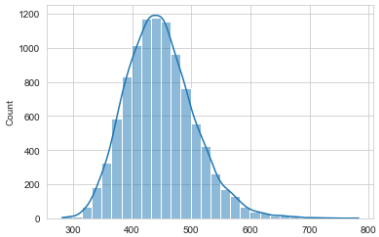


Возьмем другой вид распределения - логнормальное и повторим эксперимент:

```
size = 1000
samples = np.random.lognormal(mean=1, sigma=1, size=size)

sns.histplot(samples, bins=100, kde=True)
plt.show()
```




```
sums = []
size = 100
n_iter = 10000

for _ in range(n_iter):
    samples = np.random.lognormal(mean=1, sigma=1, size=size)
    sum_ = np.sum(samples)
    sums.append(sum_)

sns.histplot(sums, bins=30, kde=True)
plt.show()
```



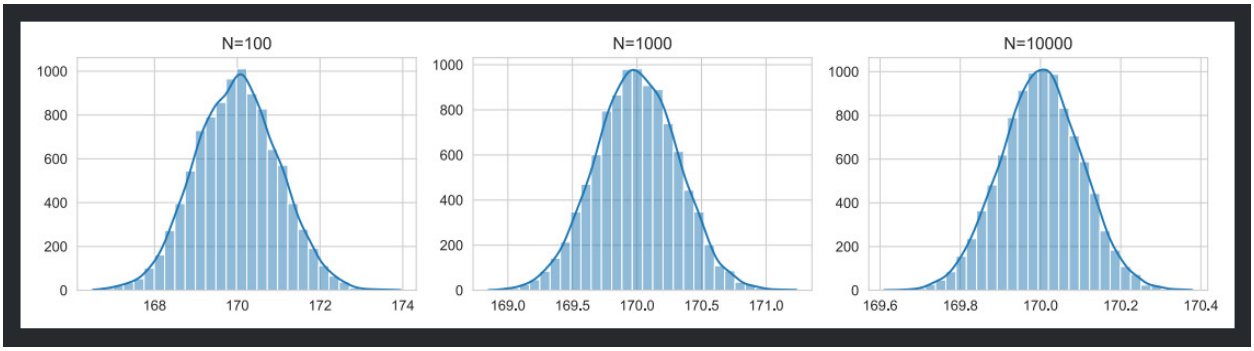
Аналогично можно проделать для любого другого распределения!

Таким образом можно также оценивать распределение выборочного среднего.

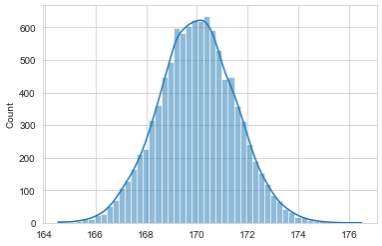
$$\overline{X} = \frac{1}{N} \sum_i X_i$$

$$\overline{X} \sim \mathbb{N}(\mu, \frac{\sigma^2}{N})$$

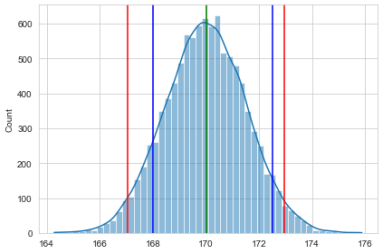
```
n_iter = 10000
plt.figure(figsize=(12, 3))

plt.subplot(1, 3, 1)
means = []
size = 100
for _ in range(n_iter):
    samples = np.random.normal(loc=170, scale=10, size=size)
    mean_ = np.mean(samples)
    means.append(mean_)
sns.histplot(means, bins=30, kde=True)
plt.ylabel('')
plt.title('N=100')

plt.subplot(1, 3, 2)
means = []
size = 1000
for _ in range(n_iter):
    samples = np.random.normal(loc=170, scale=10, size=size)
    mean_ = np.mean(samples)
    means.append(mean_)
sns.histplot(means, bins=30, kde=True)
plt.ylabel('')
plt.title('N=1000')

plt.subplot(1, 3, 3)
means = []
size = 10000
for _ in range(n_iter):
    samples = np.random.normal(loc=170, scale=10, size=size)
    mean_ = np.mean(samples)
    means.append(mean_)
sns.histplot(means, bins=30, kde=True)
plt.ylabel('')
plt.title('N=10000')

plt.tight_layout()
plt.savefig('cpt.pdf')
plt.show()
```




```
mu, sigma = 170, 15
means = []
size = 100
n_iter = 10000

for _ in range(n_iter):
    samples = np.random.normal(loc=mu, scale=sigma, size=size)
    mean_ = np.mean(samples)
    means.append(mean_)

sns.histplot(means, bins=50, kde=True)
plt.show()
```




```
sns.histplot(means, bins=50, kde=True, alpha=0.5)
plt.axvline(x=mu - 1.96 * se, color='red')
plt.axvline(x=mu + 1.96 * se, color='red')
plt.axvline(x=mu, color='green')
plt.axvline(x=168, color='blue')
plt.axvline(x=172.5, color='blue')
plt.tight_layout()
plt.savefig('cpt_dist_2s.pdf')
plt.show()
```



- Для 95%-го уровня доверия мы отступали от среднего генеральной совокупности на 2σ, чтобы покрыть 95% распределения выборочных средних
- Но ведь работает и в обратную сторону!

То есть теперь уже среднее генеральной совокупности заменяем на наше среднее выборочное, а среднеквадратичное отклонение можно (с оговоркой) заменить на среднеквадратичное отклонение выборки

```
mu, sigma = 170, 15
size = 100

data = np.random.normal(loc=mu, scale=sigma, size=size)
mean = np.mean(data)
std = np.std(data)

mean - 1.96 * std / size ** 0.5, mean + 1.96 * std / size ** 0.5
```

- ->(168.0619331978277, 173.78168682942712)

>Правило «двух сигм»

Зная параметры нормального распределения, можем сказать, где лежит основная масса наблюдений

· Квантили стандартного распределения равны, соответственно:

$z_{\frac{\alpha}{2}} = -z_{\frac{1-\alpha}{2}}$

.

Если нам известна дисперсия, то можем подставить ее и рассчитать доверительный интервал с уровнем доверия $(1-\alpha)$

$$\left(\overline{X} - z_{1-\frac{\alpha}{2}}\frac{\sigma}{\sqrt{N}}, \overline{X} + z_{1-\frac{\alpha}{2}}\frac{\sigma}{\sqrt{N}}\right)$$

Вернемся к практике:

```
mu, sigma = 170, 15
size = 100
data = np.random.normal(loc=mu, scale=sigma, size=size)

alpha = 0.05
mean = np.mean(data)
z_alpha2 = norm().ppf(q=1-alpha/2)

mean - z_alpha2 * sigma / size ** 0.5, mean + z_alpha2 * sigma / size ** 0.5
```

—>(166.78300276580276, 172.66289471942295).

- Ранее для перехода к нормальному стандартному распределению использовали следующую формулу:

$$z = \frac{x - \mu}{\sigma}$$

- С учётом ЦПТ было верно для выборочного среднего:

$$z = \frac{\overline{X} - \mu}{\frac{\sigma}{\sqrt{N}}}$$

- Можем ли мы взять среднее по выборке?

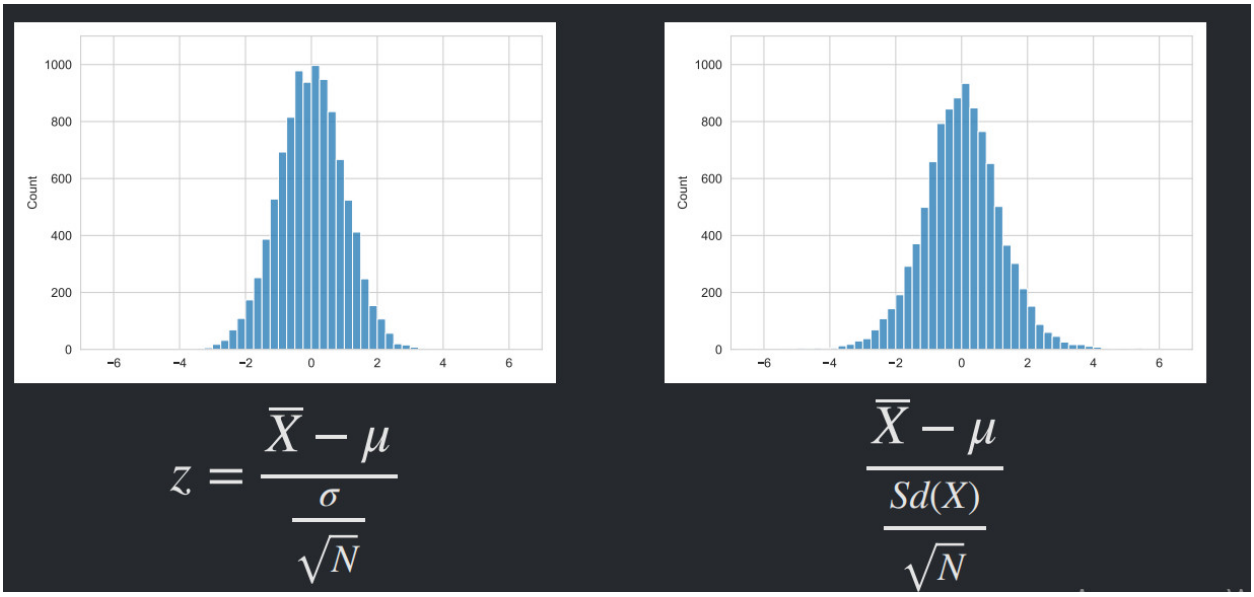
$$\frac{\overline{X} - \mu}{\frac{Sd(X)}{\sqrt{N}}} \sim ?$$

```
mu, sigma = 170, 15
zs = []
size = 10
n_iter = 10000

for _ in range(n_iter):
    samples = np.random.normal(loc=mu, scale=sigma, size=size)
    mean_ = np.mean(samples)
    zs.append((mean_ - mu) / (sigma / size ** 0.5))
```

```
sns.histplot(zs, bins=np.arange(-7.5, 7.6, 0.25), kde=False)
plt.xlim([-7, 7])
plt.ylim([0, 1100])
plt.tight_layout()
plt.savefig('znorm.pdf')
plt.show()

Select LanguageHTMLCSSJavaScriptTypeScriptBashCC#C++CMakeClojureDiffDNS zone fileDockerExcel FormulaGoGroovyHaskellHTTPIniJavaJSONJuliaKotl
zs = []
size = 10
n_iter = 10000

for _ in range(n_iter):
```

```
    samples = np.random.normal(loc=mu, scale=sigma, size=size)
    mean_ = np.mean(samples)
    zs.append((mean_ - mu) / (np.std(samples) / size ** 0.5))
```

```
sns.histplot(zs, bins=np.arange(-7.5, 7.6, 0.25), kde=False)
plt.xlim([-7, 7])
plt.ylim([0, 1100])
plt.tight_layout()
plt.savefig('tnorm.pdf')
plt.show()
```



Можно заметить, что график со стандартным отклонением по выборке получился несколько шире и ниже, так как мы снесли некоторый шум в данные и увеличили разброс.

Что же делать в том случае, когда дисперсия нам не известна?

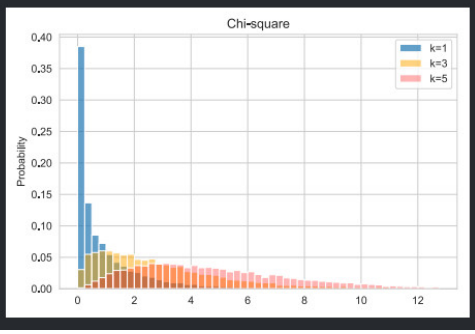
Познакомимся с новыми распределениями!

>Хи-квадрат и распределение Стьюдента (t-распределение)

Хи-квадрат с k степенями свободы – распределение суммы
квадратов k случайных величин со стандартным нормальным распределением.

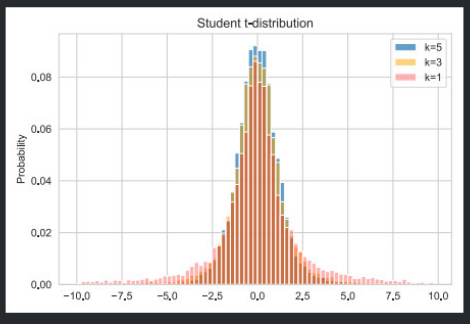
Чем больше степеней свободы, тем распределение шире.

$$X_1, \ldots, X_k \sim N(0,1) \qquad X = \sum_{i=1}^{k} X_i^2 \qquad X \sim \chi_k^2$$



Предположим, у нас есть одна величина, распределенная нормально и стандартно, и вторая величина – хи-квадрат с k степенями свободы. Если первую разделить на корень из второй, деленной на число степеней свободы, получим **распределение Стьюдента** с k степенями свободы:

$$X \sim N(0,1) \qquad \phi = \frac{X}{\sqrt{\frac{Y}{k}}} \qquad \phi \sim St(k)$$
$$Y \sim \chi_k^2$$



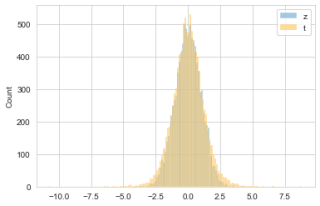

```
from scipy.stats import t

alpha = 0.05

mean = np.mean(data)
std = np.std(data)
t_alpha2 = t.ppf(q=1-alpha/2, df=N-1)

print(mean - t_alpha2 * std / size ** 0.5, mean + t_alpha2 * std / size ** 0.5)
```

- —> (166.2107638332103 173.2351336520154)

Вернемся к примеру, в котором генерируем выборки по 10 человек, считать среднее. В первом случае возьмем известную дисперсию и сделаем z-преобразование, а во втором - стандартное отклонение выборки и посчитаем t-статистику.

```
mu, sigma = 170, 15
ts, zs = [], []
```

```
size, n_iter = 10, 10000
for _ in range(n_iter):
    samples = np.random.normal(loc=mu, scale=sigma, size=size)
    mean_ = np.mean(samples)
    zs.append((mean_ - mu) / (sigma / size ** 0.5))
    ts.append((mean_ - mu) / (np.std(samples) / size ** 0.5))

sns.histplot(zs, alpha=0.4, label='z')
sns.histplot(ts, alpha=0.4, label='t', color='orange')
plt.legend(loc=1)
plt.show()
```



Как видно графики почти идентичны, но t статистика имеет более высокие хвосты. При этом с ростом размера выборки распределение Стьюдента максимально приближается к нормальному.

При $N > 30$ мы можем использовать выборочную дисперсию и квантили нормального распределения:

$$\left(\overline{X} - t_{N-1,1-\frac{\alpha}{2}}\frac{S}{\sqrt{N}}, \overline{X} + t_{N-1,1-\frac{\alpha}{2}}\frac{S}{\sqrt{N}}\right)$$

```
mean - t_alpha2 * std / size ** 0.5, mean + t_alpha2 * std / size ** 0.5
```

- -> (168.61229835487444 170.83359913035127)

```
mean - z_alpha2 * std / size ** 0.5, mean + z_alpha2 * std / size ** 0.5
```

- -> (168.62587376916923 170.82002371605648)

Сравним со значениями, полученными с помощью бутстрэпа:

```
bootstrap(
    data=(data,),
    statistic=np.mean,
    confidence_level=0.95,
    n_resamples=1000,
    method='percentile',
).confidence_interval
```

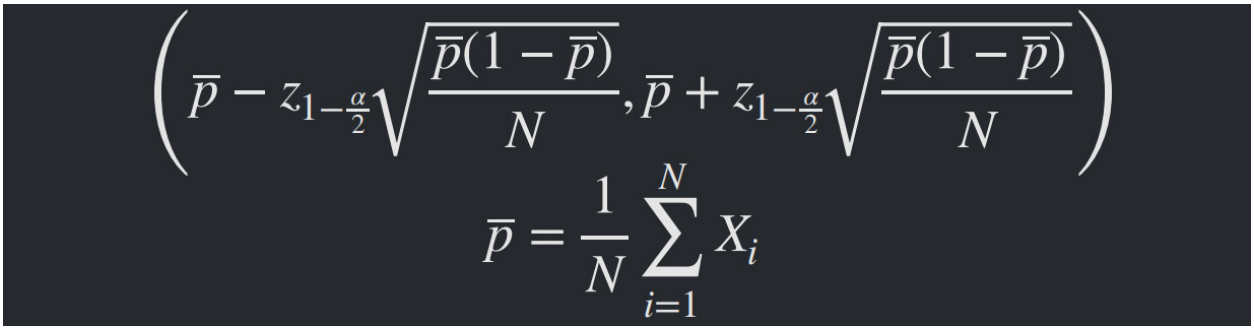
- ->ConfidenceInterval(low=166.12456917789302, high=173.22649571834188)
- При переходе от нормального стандартного распределения с z к распределению Стьюдента с t мы добавили новое условие на нормальность исходных данных, иначе нельзя гарантировать корректность полученных результатов.

- К счастью, на практике t-статистики устойчивы к отклонениям

>Доверительный интервал для доли

На практике мы часто сталкиваемся со случайной величиной,
которая описывается **распределением Бернулли**, например, сделал пользователь заказ в нашем приложении или нет, то есть величина принимает всего два значения (0,1) с некоторой вероятностью. Доля — среднее для случайной величины Бернулли.

Для бинарной случайной величины мы также можем применить
ЦПТ:



Предположим мы подкинули монетку 100 раз и 45 раз выпал орел. У нас есть предположение, что это обычная монетка и для нее вероятность выпадения орла 0.5. Построим ДИ и проверим, покрывает ли он значение 0.5:

```
N = 100
N_pos = 45

alpha = 0.05
p = N_pos / N
sigma = np.sqrt(p * (1 - p))
z_alpha2 = norm.ppf(q=1-alpha/2)

print(p - z_alpha2 * sigma / np.sqrt(N), p + z_alpha2 * sigma / np.sqrt(N))
```

- ->(0.3524930229100607 0.5475069770899393)

То же самое с использованием библиотеки statsmodels:

```
from statsmodels.stats.proportion import proportion_confint

proportion_confint(N_pos, N, alpha=0.05, method='normal')
```

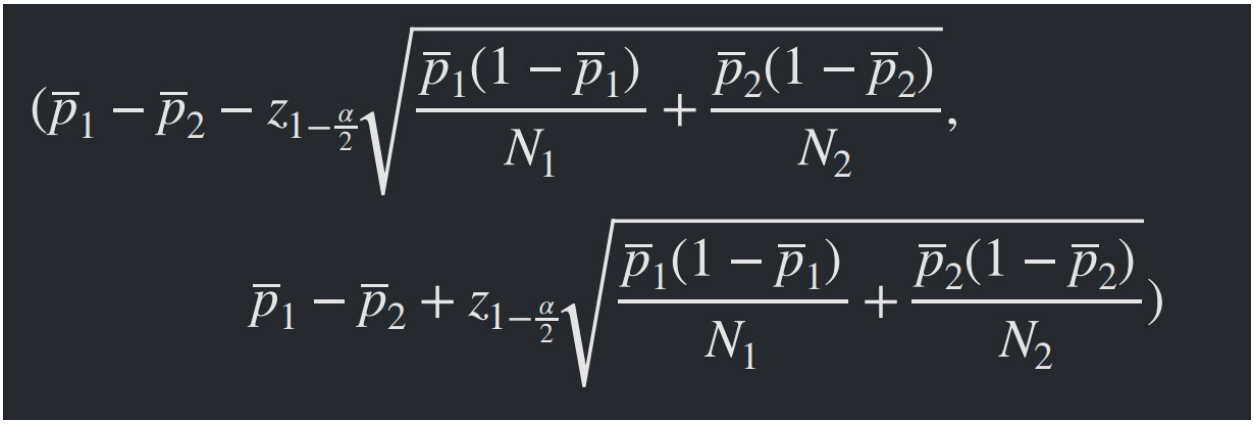
- ->(0.3524930229100606, 0.5475069770899395)

Таким образом не можем отвергнуть гипотезу о том, что монетка самая обычная.

Есть более точный способ рассчитать доверительный интервал
для доли – интервал Уилсона (Wilson).

Для такого распределения нам не нужно знать всю выборку, достаточно
знать число единичек и размер выборки. Данный метод реализован в библиотеке scipy.

На практике перед нами часто возникает задача сравнить между
собой две выборки. Мы могли бы построить для каждой доверительный интервал и сравнить между собой, но можно сразу построить доверительный интервал для разности долей двух выборок по следующей формуле:



```
ci_wilson = proportion_confint(N_pos, N, alpha=0.05, method='wilson')
ci_wilson
```

- ->(0.35614537979511973, 0.5475539700255787)

Доверительный интервал получился немного более узким, а значит более точным.