> A/B-ТЕСТЫ

> Что такое AB-тест?

На современных веб-сайтах постоянно происходит проверка разного рода гипотез. Какой должна быть кнопка «Купить»: красной или синей? Какой заголовок привлекает больше всего кликов к этой новостной статье? На какую версию рекламы больше кликают?

Чтобы определить оптимальный ответ на эти вопросы, используют A/B-тесты — статистически обоснованный метод сравнения версий A и B.

В ходе тестирования аналитик пытается определить, совпадают ли значения некой величины в группах A и B, или различие в ней на самом деле обусловлено случайностью?

A/B-тестирование основано на классическом понимании статистической значимости. Когда мы придумываем новую функциональность продукта, мы хотим проверить, полезна ли она, прежде чем запускать ее для всей пользовательской базы.

В тесте обычно участвуют две группы: группа, получающая новую функциональность (Treatment), и контрольная группа (Control). Затем мы измеряем ключевой показатель для обеих групп, например:

- среднее время пребывания на сайте (социальная сеть)
- доля постов в новостной ленте, которым поставили like (социальная сеть)
- среднее время до оформления заказа (электронная коммерция)
- CTR или кол-во кликов (онлайн-реклама)

Разница между группами проверяется на предмет статистической значимости.

Классический статистический тест при правильном применении (например z-тест, t-тест) гарантирует, что количество ложных срабатываний (FPR или ошибок первого рода) не превышает α (часто берут 5%). Это означает, что при отсутствии разницы между экспериментальной и контрольной группой тест будет показывать значимое отличие не более чем в 5% случаев.

Часто в тестах группы стараются делать приблизительно одинакового размера.

> Ставим гипотезу

К вам пришла команда ML, которая сделала некоторые новые алгоритмы рекомендаций новостей в ленте. Естественно, ожидается, что новые алгоритмы сделают пользователей счастливее (то есть удлинится LTV, повысятся денежные конверсии и т.д.), а продукт удобнее/приятнее в использовании. Вы как аналитик должны проверить эту гипотезу.

Часто к аналитику заказчик приходит с неким пониманием «что же он хотел изменить», например, «мы сделали в интернет-магазине систему лояльности и ожидаем, что повысится средний чек». В таком случае аналитику необходимо разбить пользователей на группы, выбрать время проведения теста, а целевую метрику, по которой будут сравниваться наши виды функциональностей, ему, получается, уже принесли.

В нашем случае это не совсем так. Надо понять — а что вообще мы будем сравнивать? По каким показателям можно понять, что стало лучше или хуже? На этот вопрос, к сожалению, нет готового универсального ответа. Нет правила вида «я проверяю гипотезу X, значит точно надо смотреть именно такой набор метрик». Это одно из тех мест, где требуется то самое плохо определенное понятие «понимание продукта». Вам необходимо одновременно понимать вопросы «на что влияет ваша гипотеза» и «как работает ваш продукт». Естественно, существуют некоторые понятные взаимосвязи метрик:

```
"Хочу увеличить деньги" – можно смотреть ARPU, Средний чек, LTV
"Хочу увеличить вовлеченность" – Что-то из аудиторных метрик: DAU, Sticky-factor, Time spent
```

Команда ML сказала нам «Рекомендации делают посты интереснее». Сделано было два новых алгоритма рекомендаций:

```
1) Показываем пользователю посты, наиболее похожие на те, которые он лайкал.
2) Показываем пользователю посты, которые лайкали похожие на него пользователи.
```

Нам надо придумать показатель, увеличение которого будет говорит, что посты действительно стали интереснее. Что это может быть (варианты):

- Время, проведенное в ленте
- Количество просмотренных постов в единицу времени
- Количество лайков
- **CTR из просмотров в лайки**

Почему мы взяли именно CTR?

- Есть явное действие, которое показывает заинтересованность – лайк
- Это довольно стандартная метрика (все так делают и мы туда же)

График CTR

> Система сплитования

Хорошо, мы зафиксировали метрику. Что делать теперь?

Сначала давайте ответим на вопрос: зачем нам вообще проводить эксперимент, почему бы просто не выбрать одну из моделей рекомендаций и не добавить ее всем пользователям? Потому что не исключено, что мы чего-то не учли, и в продакшене модель рекомендаций будет вести себя иначе, чем мы ожидаем. Вспомним наш главный дашборд базовых метрик. Было бы странно оставить модель, внедрение которой приведет к падению лайков и просмотров и оттоку активной аудитории. Мы не можем так рисковать. Поэтому перед внедрением таких существенных изменений сначала нужно протестировать их на небольшом проценте пользователей.

Чего мы ожидаем от разбиения на группы

Мы хотим, чтобы:

- Разбиение было случайным

- У каждого пользователя в течение эксперимента группа не менялась
- Способ разбиения должен быть детерминированным (чтобы можно было, если нужно, в точности повторить разбиение)

Зачем мы этого хотим?

Про случайность

Сравнивая две группы друг с другом в эксперименте, мы хотим чтобы они отличались друг от друга только нашим воздействием в рамках теста (в нашем случае – алгоритмом рекомендаций), а все остальное в группах должно быть одинаковым. Например, чтобы не получилось, что в группе A больше пользователей из Москвы, а в группе B — пользователей из регионов, или в группе A больше пользователей, которые пользуются сервисом в будние дни, а в группе B — тех, которые больше пользуются сервисом в выходные. Говоря умными словами, мы хотим, чтобы группы были статистически эквивалентны. Кроме того, мы хотим, чтобы показатели группы отражали показатели в генеральной совокупности. То есть мы хотим провести тест на маленькой группе пользователей, но так, чтобы полученные выводы потом можно было распространить «на всех наших пользователей».

Подводя итог:

Если метрики в группе A окажутся значимо лучше чем в группе B, мы ожидаем, что алгоритм рекомендаций A лучше, чем алгоритм рекомендаций B (в контексте анализируемой метрики, например, CTR).

Про неизменность группы

Одно из наших основных предположений в том, что группы отличаются друг от друга только алгоритмом рекомендаций. Если группа на протяжении теста сохраняться не будет, получится, что часть пользователей видела и тот алгоритм рекомендаций, и другой, что обычно плохо. Более того — нельзя просто так взять и «выкинуть» этих пользователей, а последующий анализ проводить без них, это может испортить выполнение пункта про случайность. То есть результат у нас будет, а раскатить его на всех потом будет нельзя.

Про детерминированность

Представим, что мы решили протестировать некоторую другую функциональность, например, появление дополнительной «карусели» с популярными «историями» (мы хотим, чтобы люди их смотрели). Но прямо сейчас уже идет тест с новыми алгоритмами рекомендаций. Мы не хотим, чтобы наши эксперименты пересеклись друг с другом. Для этого нам надо выделить группы под новый эксперимент таким образом, чтобы можно было полностью исключить тех, кто находится в эксперименте с рекомендациями. Без детерминированности это будет весьма затруднительно.

А как нам разбить пользователей на группы? Стандартная техника для такого разбиения — **хеширование с солью**.

Код из лекции с реализацией «соли»:

```
def ab_split(id, salt='exp_mess_1', n_groups=5):
    test_id = str(id) + '-' + str(salt)
    test_id_digest = hashlib.md5(test_id.encode('ascii')).hexdigest()
    test_id_final_int = int(test_id_digest, 16)
    return test_id_final_int % n_groups
```

> Чем же так хорошо “солёное” хэширование?

Детерминированность есть, случайность есть, есть гарантия, что для одного и того же id будет одна и та же группа (группы не будут меняться).

Основная «фишка» в том, что какие бы id'шки к нам на вход ни пришли, разбиение их по соленому хешу гарантирует, что получившиеся группы будут случайны и приблизительно одинакового размера!

Продемонстрируем на примере:

```
connection = {
    'host': 'https://clickhouse.lab.karpov.courses',
    'password': 'dpo_python_2020',
    'user': 'student',
    'database': 'simulator'
}

q = """
SELECT distinct user_id
FROM {db}.feed_actions
WHERE toDate(time) >= '2021-11-15' and toDate(time) <= '2021-11-21'
"""

# Вытащили пользователей
users_df = pandahouse.read_clickhouse(q, connection=connection)

# Сделали группы
users_df['hash_group'] = users_df.user_id.swifter.apply(ab_split)

# Смотрим на число пользователей в каждой группе
users_df.groupby('hash_group').user_id.nunique().reset_index()
```

	hash_group	user_id
0	0	10151
1	1	9982
2	2	9977
3	3	9963
4	4	10076

Хэшировать можно и прямо в кликхаусе:

```
q = """
SELECT xxHash64(toString(user_id)||'my_salt')%5 as group,
    uniqExact(user_id)
FROM {db}.feed_actions
WHERE toDate(time) >= '2021-11-15' and toDate(time) <= '2021-11-21'
group by group
"""

pandahouse.read_clickhouse(q, connection=connection)
```

	group	uniqExact(user_id)
0	0	10091
1	1	9976
2	2	9966
3	3	10140
4	4	9976

> Проверка корректности работы системы сплитования

Надо удостовериться, что наша система корректно бьет пользователей на группы, то есть размеры групп **примерно одинаковые** и показатели в них сами по себе **не различаются**.

То есть группы, на которые мы побили пользователей, не отличаются друг от друга до тех пор, пока мы не выкатили на них новую функциональность. Это критически важный момент. Для проверки справедливости этого надо провести АА-тест.

Только если группы до нашего воздействия были «одинаковыми», мы позволяем себе утверждать, что возникшее после воздействия отличие вызвано воздействием.

На данных в лекции:

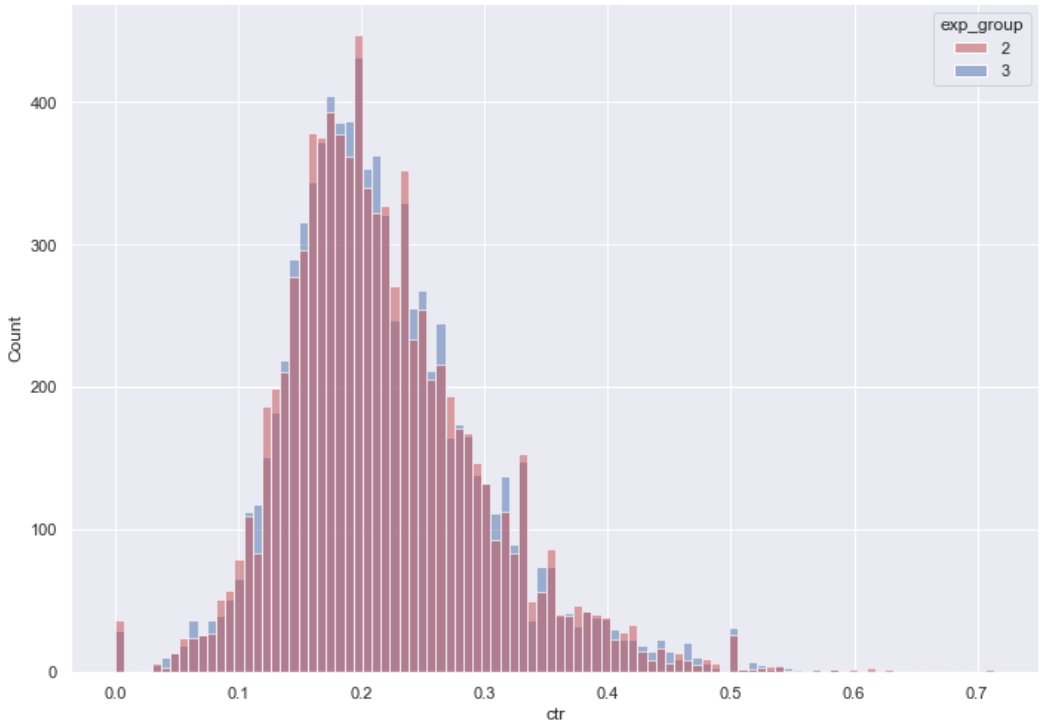
- АА-тест проходил с 8 по 14 ноября 2021
- АБ-тест с нашими системами рекомендаций проходил с 15 по 21 ноября 2021

```
q = """
SELECT exp_group,
    user_id,
    sum(action = 'like') as likes,
    sum(action = 'view') as views,
    likes/views as ctr
FROM {db}.feed_actions
WHERE toDate(time) between '2021-11-08' and '2021-11-14'
    and exp_group in (2,3)
GROUP BY exp_group, user_id
"""

df = pandahouse.read_clickhouse(q, connection=connection)

df.groupby('exp_group').count()
```

exp_group	user_id	likes	views	ctr
2	8480	8480	8480	8480
3	8569	8569	8569	8569

```
# Сделаем графики в seaborn покрупнее
sns.set(rc={'figure.figsize':(11.7,8.27)})

groups = sns.histplot(data = df,
              x='ctr',
              hue='exp_group',
              palette = ['r', 'b'],
```

```
                alpha=0.5,
                kde=False)
```



> AA-тест “по классике”

Мы «на глаз» убедились, что наши группы как будто бы одинаковые. Давайте в этом удостоверимся с помощью готовых тестов :)

Пункт 1. Известное. t-критерий Стьюдента

Плюсы:

1. Интерпретируемый, обычно сравнивает средние
2. Легко получить доверительный интервал
3. Считается быстро (за $O(N_1 + N_2)$, где N_1 и N_2 - длины выборки)

Минусы: смотрим ниже.

Описание. T-критерий (дисперсии нам неизвестны)

Выборка $X_1^{n_1} = X_{11}...X_{1n_1}$ из распределения X_1

Выборка $X_2^{n_2} = X_{21}...X_{2n_2}$ из распределения X_2

Дисперсии σ_1^2 и σ_2^2 неизвестны.

Нулевая гипотеза $H_0 : \mu 1 = \mu 2$

Альтернативная гипотеза $H_1 : \mu 1 \neq \mu 2$

Статистика $T_n = \frac{\bar{X_1} - \bar{X_2}}{\sqrt{\frac{S_1^2}{n_1} + \frac{S_2^2}{n_2}}}$

Нулевое распределение $Z_n(X_1^{n_1}, X_2^{n_2}) \approx\sim Student(\xi)$

t-тест Стьюдента в такой формулировке сравнивает **средние значения** в двух выборках. В интернете можно встретить бесконечное количество войн на тему «должны ли выборки в t-критерии Стьюдента быть нормальными». Что про эту статистику надо понимать:

1. t-статистика — это штука, у которой в числителе стоит нормальное распределение, а в знаменателе — оценка его дисперсии (которая должна иметь распределение хи-квадрат)
2. Если у вас достаточно много наблюдений (а у нас их тысячи), среднее значение по выборке часто будет оказываться нормальным (благодаря Центральной Предельной Теореме).
3. Если у вас достаточно много наблюдений (а у нас их тысячи), выборочная оценка дисперсии (которая стоит в знаменателе) часто будет сходиться (aka будет очень похожа на истинную дисперсию распределения)

Проблема в том, что пункты 2 и 3 могут быть «испорчены» и на определенных видах данных «не работать», если:

1. Распределение очень сильно скошено.
2. В распределении много выбросов.
3. В распределении очень высока вероятность больших по модулю отклонений (жирные выбросы — часть распределения).

Главная проблема — t-тест требует явной оценки дисперсии исследуемой метрики (которая в знаменателе), а построить ее не всегда получается, например, если вы исследуете значение квантиля некоторого неизвестного заранее распределения.

```
stats.ttest_ind(df[df.exp_group == 2].ctr,
                df[df.exp_group == 3].ctr,
                equal_var=False)
```

Пункт 2. Тоже известное. тест Манна-Уитни

Плюсы

- Почти так же чувствителен, как t-тест
- Приблизительно умеет проверять различие в средних
- Не слишком привязан к распределению вашей метрики

Минусы

- Не очень удобно интерпретируемая нулевая гипотеза. Это **НЕ равенство средних, НЕ равенство медиан** и так далее
- Из-за неудобной нулевой гипотезы может «краситься» даже при равенстве средних (но это, на самом деле, небесполезно)
- Работает подольше чем тест Стьюдента: $(O((N_1 + N_2) * log(N_1 + N_2))$

Описание. критерий Манна-Уитни (непараметрический)

Выборка $X_1^{n1} = X_{11}...X_{1n_1}$ из распределения X_1,

Выборка $X_2^{n2} = X_{21}...X_{2n_2}$ из распределения X_2

Нулевая гипотеза $H_0 : P(X > Y) = P(Y > X)$

Альтернативная гипотеза $H_1 : P(X > Y) \neq P(Y > X)$

Статистика Штука, составленная из рангов

Нулевое распределение Табличное, но есть нормальная подгонка при больших N

```
print(stats.mannwhitneyu(df[df.exp_group == 2].ctr,
                df[df.exp_group == 3].ctr))

print('Отличие не прокрасилось')

A_gt_B = 0
for _ in range(10000):
    A_gt_B+= df[df.exp_group == 2].ctr.sample().values[0] > df[df.exp_group == 3].ctr.sample().values[0]

print('B', A_gt_B/100, '% случаев A > B. Должно получиться около 50%')
```

Output:

```
MannwhitneyuResult(statistic=36124728.5, pvalue=0.25887403071687204)
Отличие не прокрасилось
В 49.79 % случаев A > B. Должно получиться около 50%
```

> Наконец-то AB-тест

У нас параллельно идут два теста, оба – с 15 по 21 ноября 2021

Первый

```
exp_group = 1 - Все по-старому
exp_group = 2 - Рекомендации "похожих на лайкнутые постов"
```

Второй

```
exp_group = 0 – Все по-старому
exp_group = 3 - Рекомендации "постов, которые лайкали похожие на вас люди"
```

Кто-то из мог слышать о проблеме множественных сравнений, но мы на этом останавливаться не будем.

Те же самые данные понадобятся нам для домашнего задания.

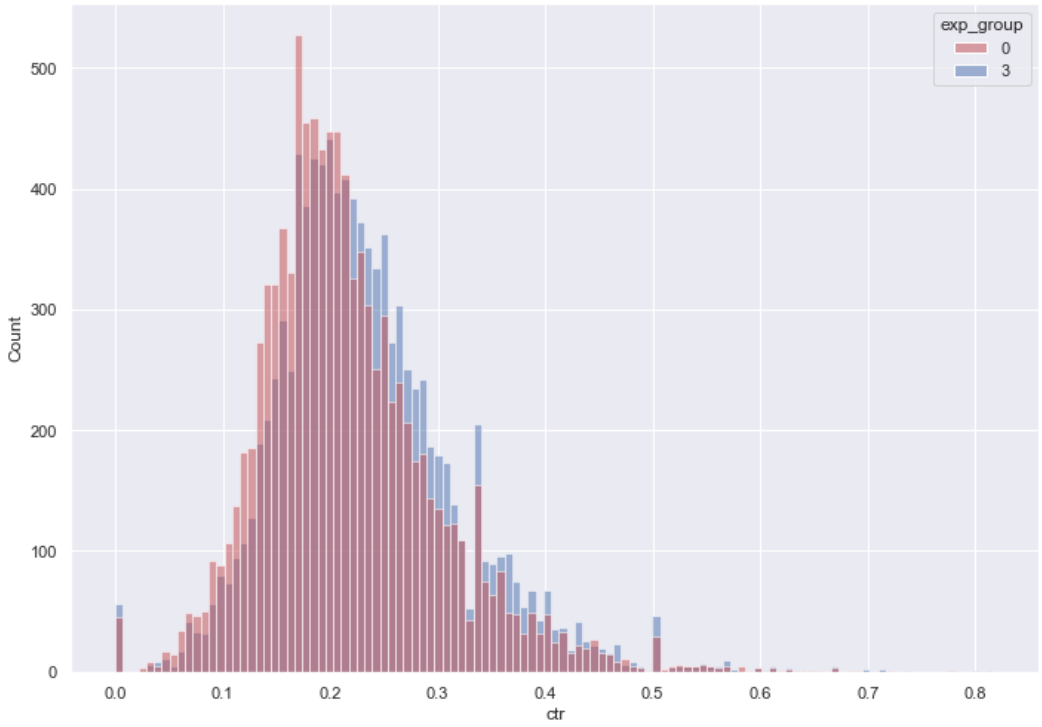
```
q = """
SELECT exp_group,
    user_id,
    sum(action = 'like') as likes,
    sum(action = 'view') as views,
    likes/views as ctr
FROM {db}.feed_actions
WHERE toDate(time) between '2021-11-15' and '2021-11-21'
    and exp_group in (0,3)
GROUP BY exp_group, user_id
"""

df = pandahouse.read_clickhouse(q, connection=connection)

#T-тест

stats.ttest_ind(df[df.exp_group == 0].ctr,
                df[df.exp_group == 3].ctr,
                equal_var=False)
```

Output:

```
Ttest_indResult(statistic=-13.89687072190407, pvalue=1.055849414662529e-43)
```

```
groups = sns.histplot(data = df,
              x='ctr',
              hue='exp_group',
              palette = ['r', 'b'],
```

```
                alpha=0.5,
                kde=False)
```



Судя по всему, различия есть. Давайте сравним с Манна-Уитни:

```
stats.mannwhitneyu(df[df.exp_group == 0].ctr,
                   df[df.exp_group == 3].ctr,
                   alternative = 'two-sided')
```

Output:

```
MannwhitneyuResult(statistic=43682792.0, pvalue=1.1952055473582936e-57)
```

Это всё можно сделать, не выходя из кликхауса:

```
--Для t-теста
WITH welchTTest(ctr, exp_index) as ttest
select tupleElement(ttest,1) as statistic, tupleElement(ttest,2) as p_value
FROM (SELECT exp_group,
```

```
        case when exp_group = 0 then 0 else 1 end as exp_index,
        user_id,
        sum(action = 'like') as likes,
        sum(action = 'view') as views,
        likes/views as ctr
      FROM {db}.feed_actions
      WHERE toDate(time) between '2021-11-15' and '2021-11-21'
        and exp_group in (0,3)
      GROUP BY exp_group, user_id)

--Для Манна-Уитни
WITH mannWhitneyUTest('two-sided')(ctr, exp_index) as mw
select tupleElement(mw,1) as statistic, tupleElement(mw,2) as p_value
FROM (SELECT exp_group,
        case when exp_group = 0 then 0 else 1 end as exp_index,
        user_id,
        sum(action = 'like') as likes,
        sum(action = 'view') as views,
        likes/views as ctr
      FROM {db}.feed_actions
      WHERE toDate(time) between '2021-11-15' and '2021-11-21'
        and exp_group in (0,3)
      GROUP BY exp_group, user_id)
```

> Сглаженный CTR

Проблема t-теста на пользовательском CTR в том, что у нас есть пользователи с небольшим количеством просмотров, а их CTR сильно зашумлен. Мы почти ничего не знаем о CTR пользователей, у которых всего 3 просмотра, независимо от количества их кликов. Когда у пользователя есть 20 просмотров, мы можем сделать приемлемую оценку CTR этого пользователя. А если у пользователя 200 просмотров, мы можем \pm± точно оценить его CTR.

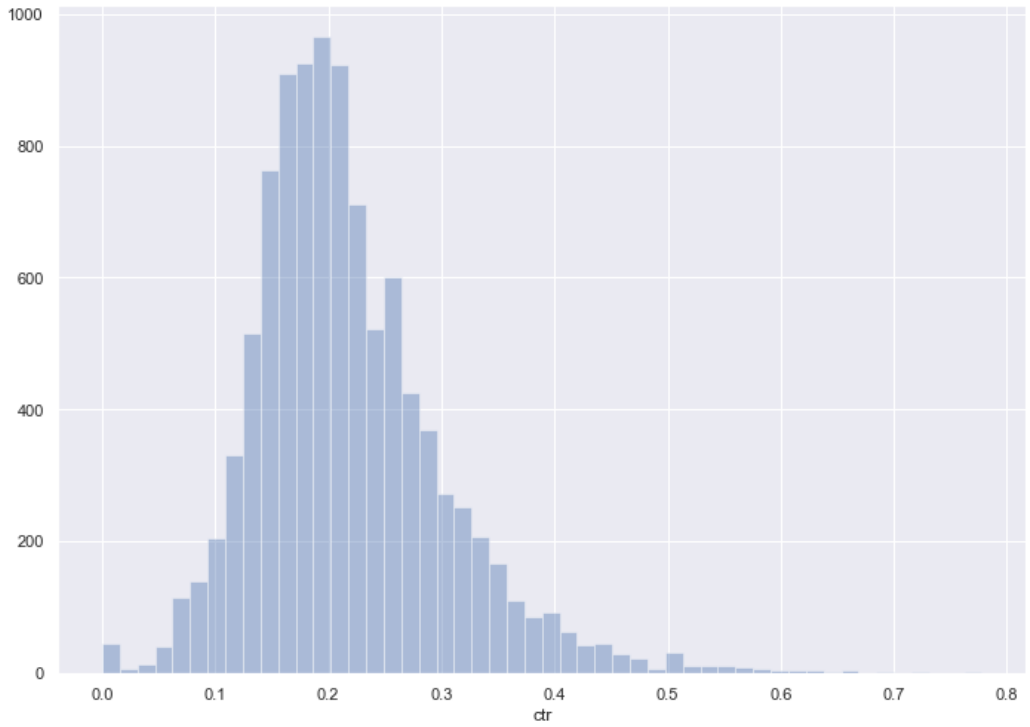
Давайте «сгладим» наши оценки CTR (процедура называется **сглаживанием Лапласа**, и у него интересная история которую можно прочитать тут):

$$smoothed\_CTR_u = \frac{clicks_u + \alpha * globalCTR}{views_u + \alpha}$$

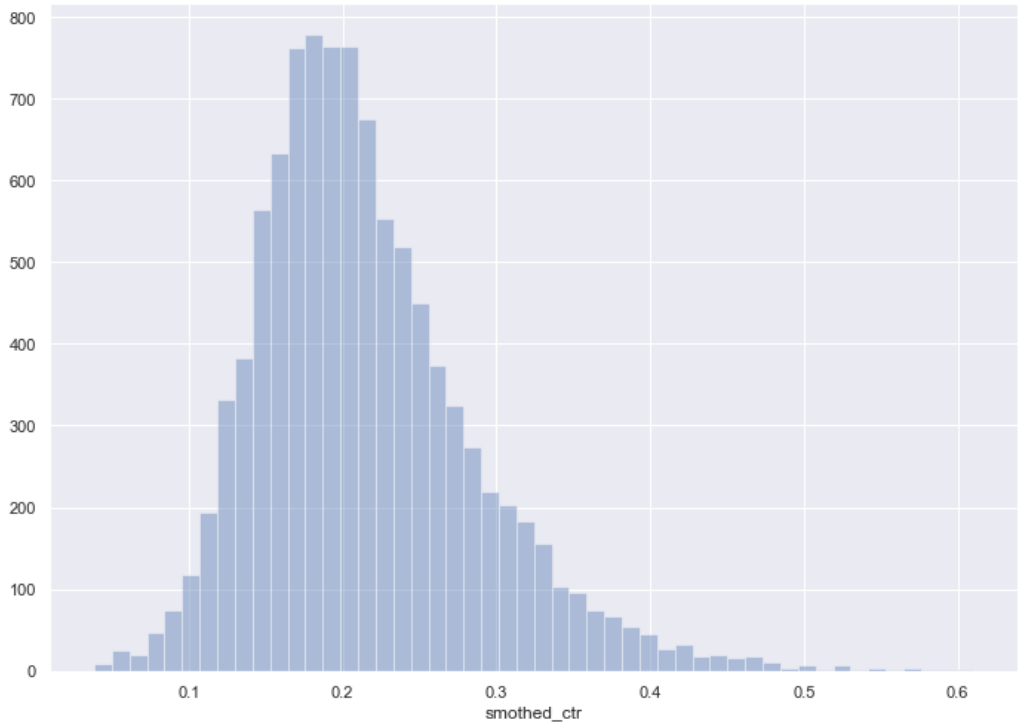
α — гиперпараметр. Идея проста: когда просмотров много, сглаженный CTR почти равен CTR пользователя. Когда просмотров мало, сглаженный CTR почти равен общегрупповому CTR. Иначе говоря, если у пользователя много просмотров, мы можем быть уверены, что клики / просмотры — хорошая оценка его CTR, а когда у пользователя просмотров мало, мы устанавливаем в качестве оценки общегрупповой CTR.

Хоть идея сглаженного CTR интуитивно понятна, теоретической гарантии, что его направленность совпадает с направленностью общего количества кликов, нет. Для кого-то это основная причина такое НЕ использовать, но идея интересная.

```
def get_smothed_ctr(user_likes, user_views, global_ctr, alpha):
    smothed_ctr = (user_likes + alpha * global_ctr) / (user_views + alpha)
    return smothed_ctr

q = """
SELECT exp_group,
    user_id,
    sum(action = 'like') as likes,
    sum(action = 'view') as views,
    likes/views as ctr
FROM {db}.feed_actions
WHERE toDate(time) between '2021-11-15' and '2021-11-21'
    and exp_group in (0,3)
GROUP BY exp_group, user_id
"""

df = pandahouse.read_clickhouse(q, connection=connection)

global_ctr_1 = df[df.exp_group == 0].likes.sum()/df[df.exp_group == 0].views.sum()
global_ctr_2 = df[df.exp_group == 3].likes.sum()/df[df.exp_group == 3].views.sum()

group1 = df[df.exp_group == 0].copy()
sns.distplot(group1.ctr,
             kde = False)
```



```
group1['smothed_ctr'] = df.apply(
    lambda x: get_smothed_ctr(x['likes'], x['views'], global_ctr_1, 5), axis=1)

sns.distplot(group1.smothed_ctr,
             kde = False)
```



> Бутстреп и CTR

До сих пор мы считали тесты на CTR по пользователям. Это хороший подход, но на самом деле нас интересует общий показатель по ВСЕЙ группе. Если бы мы были в мультивселенной, мы могли бы провести наш тест 100500 раз и сравнить получившиеся CTR в группах эти же 100500 раз. Та группа, которая выигрывала чаще, признавалась бы победителем. К мультивселенным мы доступа не имеем, поэтому приходится выкручиваться.

Непараметрический бутстреп

Очень медленно работающий метод, который в каком-то смысле является «серебряной пулей» в AB-тестировании. Он не требователен к распределению, может считать почти любые экзотические статистики, а еще может строить доверительные интервалы.

Идея

Мы хотим сравнить общегрупповой CTR в группах A и B. Проблема в том, что это всего лишь два числа. А нам хочется, чтобы это было распределение (как если бы были мультивселенные).

Предположение

Наши выборки из групп являются хорошей моделью генеральной совокупности.

Что делаем

У нас есть выборка из группы A (все по-старому) и выборка из группы B (выборка с новой системой рекомендаций). Нагенерируем из каждой из этих выборок псевдовыборки с возвращением.

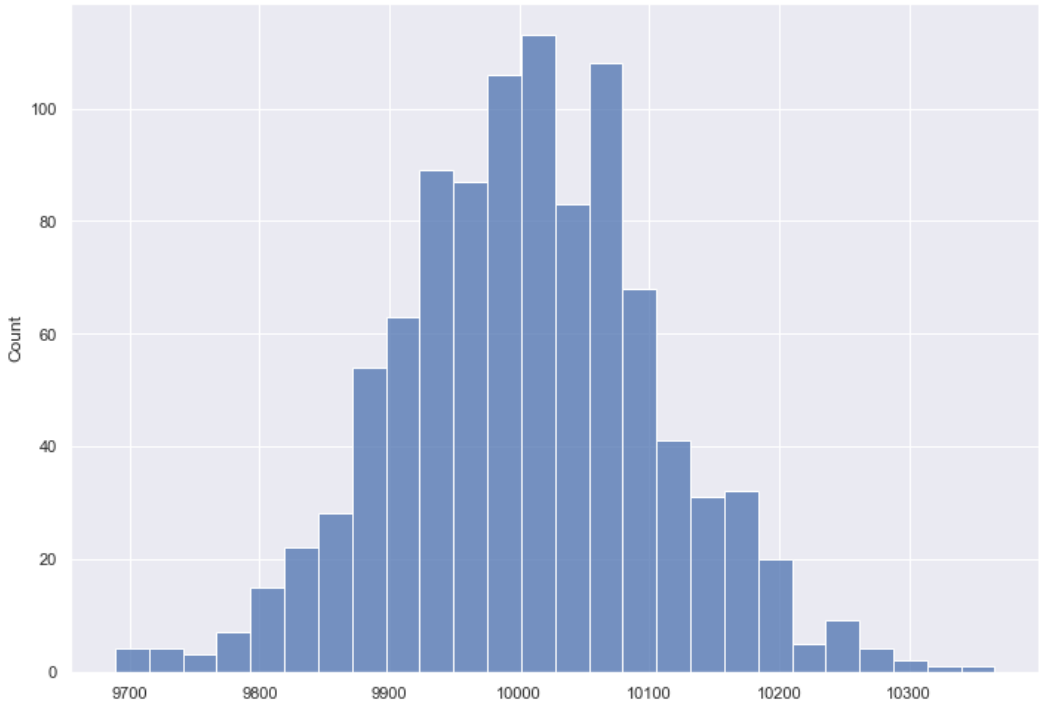
Как это сделать?

```
# Есть выборка

df = pd.DataFrame()
df['data'] = [1,2,3,4,5,6,7,8,9,10]

# генерю псевдовыборку
[np.random.choice(df.data) for _ in range(len(sample))]

# То же самое на pandas
df.data.sample(10, replace=True).tolist()
```

А можно по-другому. Мы можем считать «сколько раз в псевдовыборку попадет наше наблюдение». Такая величина будет иметь биномиальное распределение вида:

$$k = Binomial(n, 1/n)$$

```
df = pd.DataFrame()
df['data'] = np.random.randint(1,100000,10000)

sample_length = 10000
p_to_be_taken = 1/sample_length
number_of_samples = 1000

sums = [sum(stats.binom(sample_length,
                        p_to_be_taken).rvs(sample_length)) for _ in range(number_of_samples)]

sns.histplot(sums)
```



Считать «сколько раз наблюдение попадет в выборку» намного быстрее, чем последовательно проверять «попало - не попало» много раз. И несмотря на то, что не все выборки получатся в точности одинаковой длины, нам это не помешает.

Ускоримся еще немножко. При большой длине выборки наше биномиальное распределение – почти то же самое, что распределение Пуассона:

$$k = Binomial(n, 1/n) \sim Poisson(1)$$

Тут, как видите, никакой зависимости от n нет. И считается оно чуть побыстрее.

Пуассоновский бутстреп для подсчета разницы в CTR

- Берем выборку из группы A (у нас это $user\_id$ с его $views$ и $clicks$)
- Из распределения Пуассона генерируем «сколько раз этот пользователь попал в псевдовыборку». Получается колонка, отражающая «вес» пользователя. Назовем ее $weights_1$
- Считаем «глобальный CTR» по псевдовыборке для A*A* :

$$globalCTR_A = \frac{sum(clicks_A * weights_1)}{sum(views_A * weights_1)}$$

- Берем выборку из группы B (у нас это $user\_id$ с его $views$ и $clicks$)
- Из распределения Пуассона генерируем «сколько раз этот пользователь попал в псевдовыборку». Получается колонка, отражающая «вес» пользователя. Назовем ее $weights_2$
- Считаем «глобальный CTR» по псевдовыборке для B*B*:

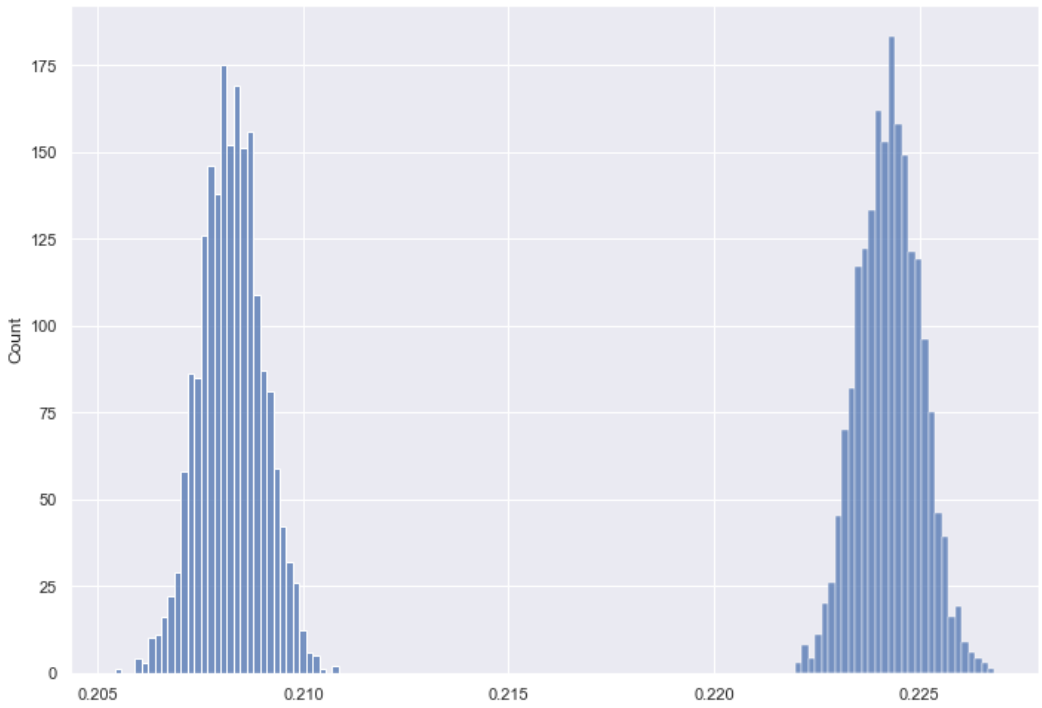
$$globalCTR_B = \frac{sum(clicks_B * weights_2)}{sum(views_B * weights_2)}$$

- Считаем разницу между $globalCTR_A - globalCTR_B$ и записываем эту разность, например, в список
- Повторяем процедуру много раз, например, 2000

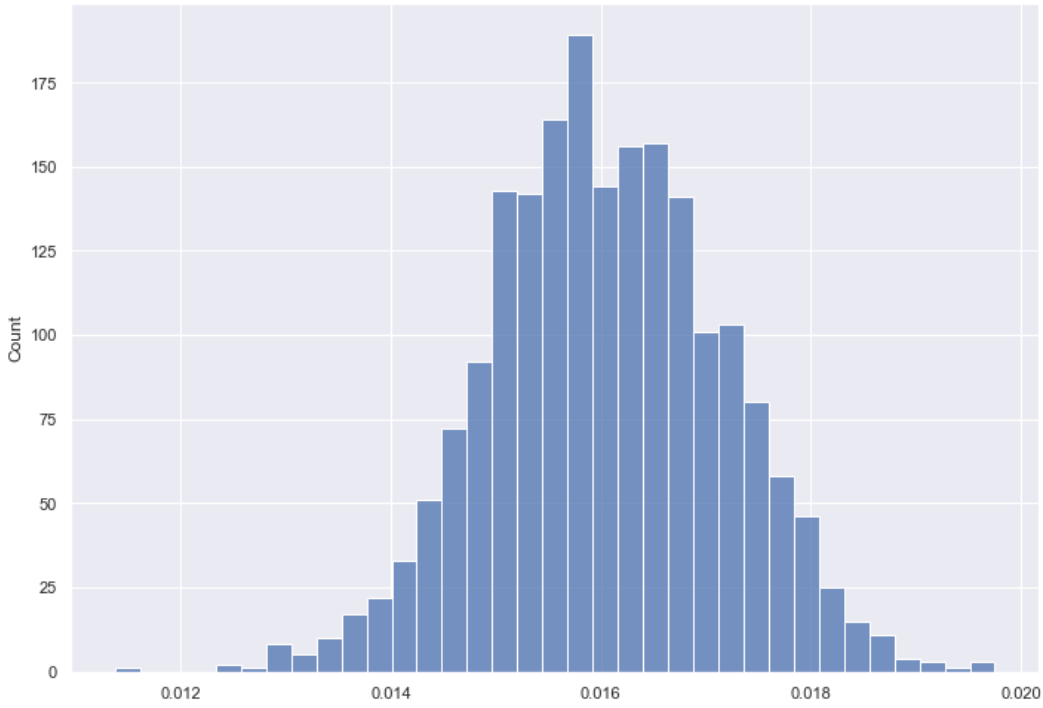
Таким образом, мы набираем распределение для разницы в глобальном CTR.

```
def bootstrap(likes1, views1, likes2, views2, n_bootstrap=2000):

    poisson_bootstraps1 = stats.poisson(1).rvs(
        (n_bootstrap, len(likes1))).astype(np.int64)

    poisson_bootstraps2 = stats.poisson(1).rvs(
            (n_bootstrap, len(likes2))).astype(np.int64)

    globalCTR1 = (poisson_bootstraps1*likes1).sum(axis=1)/(poisson_bootstraps1*views1).sum(axis=1)

    globalCTR2 = (poisson_bootstraps2*likes2).sum(axis=1)/(poisson_bootstraps2*views2).sum(axis=1)

    return globalCTR1, globalCTR2


q = """
SELECT exp_group,
    user_id,
    sum(action = 'like') as likes,
    sum(action = 'view') as views,
    likes/views as ctr
FROM {db}.feed_actions
WHERE toDate(time) between '2021-11-15' and '2021-11-21'
    and exp_group in (0,3)
GROUP BY exp_group, user_id
"""


df = pandahouse.read_clickhouse(q, connection=connection)

likes1 = df[df.exp_group == 0].likes.to_numpy()
views1 = df[df.exp_group == 0].views.to_numpy()
likes2 = df[df.exp_group == 3].likes.to_numpy()
```

```
views2 = df[df.exp_group == 3].views.to_numpy()

ctr1, ctr2 = bootstrap(likes1, views1, likes2, views2)

sns.histplot(ctr1)
sns.histplot(ctr2)
```




```
#Разница между глобальными CTR

sns.histplot(ctr2 - ctr1)
```



> Бакетное преобразование

Еще один подход, который очень часто применяется — бакетное преобразование. Надо разбить пользователей на n «корзинок» («бакетов»). Нужно гарантировать, что каждый пользователь попадет только в один бакет и пользователи по бакетам будут распределены равномерно. Это можно гарантировать с помощью того же хеширования с солью. О применении такого в Авито можно прочитать тут. Каждый бакет теперь становится как бы «метапользователем», участвующим в эксперименте. И внутри такого «метапользователя» показатель нужно как-то агрегировать.

Обычно вычисляют среднюю метрику в бакете, но можно, например, подсчитать медиану или какой-то квантиль. Скажем, если у нас 50 бакетов для теста и контроля, получается будто мы тест проводим на группах по 50 метапользователей в каждой :)

В чем тут логика

Когда мы делим нашу исходную выборку на какое-то количество бакетов и считаем в каждом бакете, например, среднее, на выходе мы получаем выборку из 50 значений среднего по этим «метапользователям». Это как будто мы сделали бутстреп, но только не сгенерировав псевдовыборку, а откусив правильным образом кусок из имеющейся.

Поверх бакетного преобразования можно, например, применить стандартный t-тест (если примерно выполнены его предположения) или тест Манна-Уитни.

```
q = """

SELECT exp_group, bucket,
    sum(likes)/sum(views) as bucket_ctr,
    quantileExact(0.9)(ctr) as ctr9
FROM (SELECT exp_group,
        xxHash64(user_id)%50 as bucket,
        user_id,
        sum(action = 'like') as likes,
        sum(action = 'view') as views,
        likes/views as ctr
    FROM {db}.feed_actions
    WHERE toDate(time) between '2021-11-15' and '2021-11-21'
        and exp_group in (0,3)
    GROUP BY exp_group, bucket, user_id)
GROUP BY exp_group, bucket
"""

df = pandahouse.read_clickhouse(q, connection=connection)

#тест Манна-Уитни видит отличие
stats.mannwhitneyu(df[df.exp_group == 0].bucket_ctr,
                   df[df.exp_group == 3].bucket_ctr,
                   alternative = 'two-sided')
```

Output:

```
MannwhitneyuResult(statistic=45.0, pvalue=1.0099815060146218e-16)
```

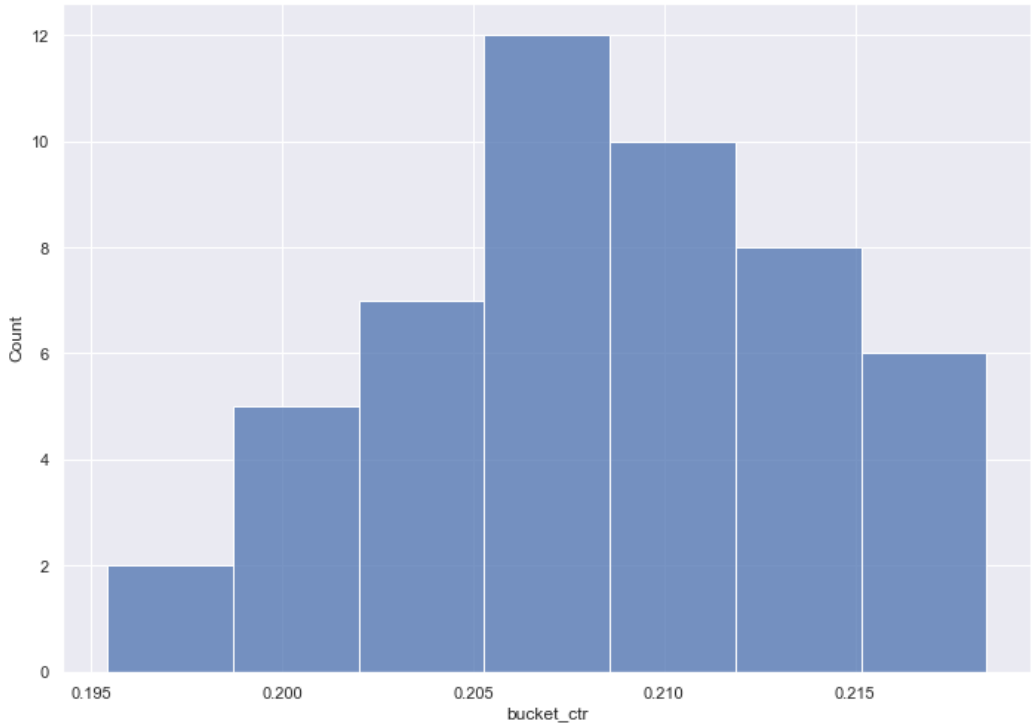
```
#и t-тест тоже

stats.ttest_ind(df[df.exp_group == 0].bucket_ctr,
                   df[df.exp_group == 3].bucket_ctr,
                   equal_var = False)
```

Output:

```
Ttest_indResult(statistic=-14.755503007460343, pvalue=1.2278707215537128e-26)
```

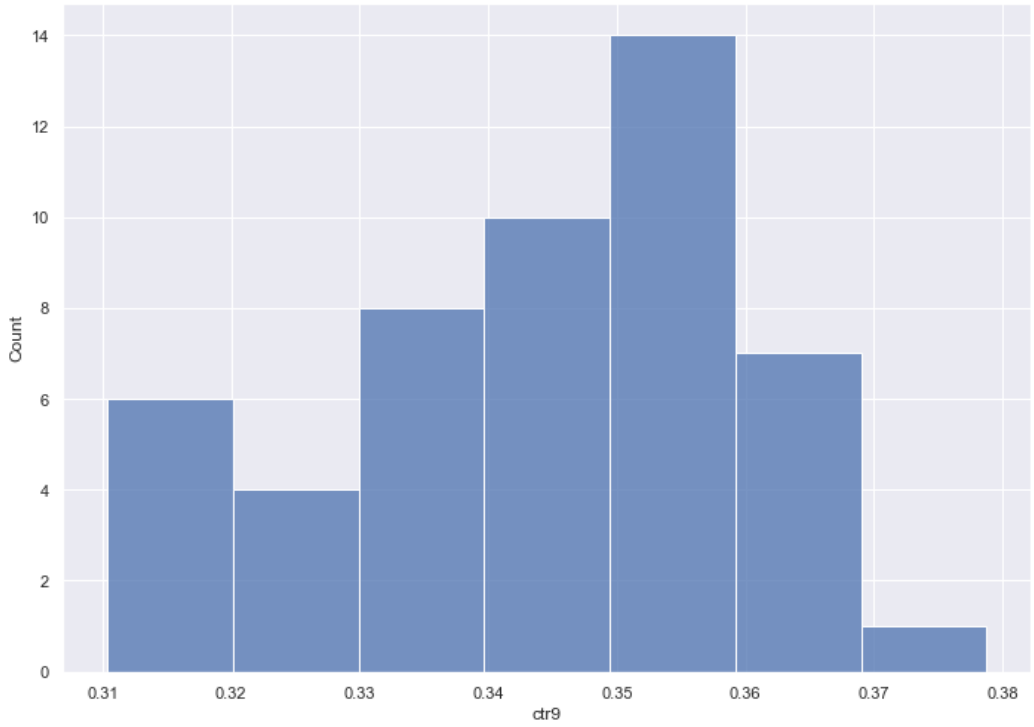
Распределение бакетного CTR даже более-менее похоже на нормальное!

```
sns.histplot(df[df.exp_group == 0].bucket_ctr)
```



А вот с квантилями пользовательских CTR стоит быть поосторожнее, t-тест тут может работать не очень хорошо:

```
sns.histplot(df[df.exp_group == 3].ctr9)
```


14
12
10
8
6
4
2
0
Count
0.31
0.32
0.33
0.34
0.35
0.36
0.37
0.38
ctr9