**Калькуляторы, Монте-Карло**

# Импорты

import pandahouse

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

import numpy as np

import hashlib

import swifter

from scipy import stats

from scipy.stats import norm, ttest\_ind

%matplotlib inline

# Подключаемся

connection = {

'host': 'https://clickhouse.lab.karpov.courses',

'password': 'dpo\_python\_2020',

'user': 'student',

'database': 'simulator'

}

# Функция формулы калькулятора

def compute\_sample\_size\_abs(epsilon, std1, std2, alpha=0.05, beta=0.2):

t\_alpha = norm.ppf(1 - alpha / 2, loc=0, scale=1)

t\_beta = norm.ppf(1 - beta, loc=0, scale=1)

z\_scores\_sum\_squared = (t\_alpha + t\_beta) \*\* 2

sample\_size = int(

np.ceil(

z\_scores\_sum\_squared \* (std1 \*\* 2 + std2 \*\* 2) / (epsilon \*\* 2)

)

)

return sample\_size

# Выбираем данные и загружаем в views\_distribution

q = '''

select views, count() as users

from (select

exp\_group,

user\_id,

sum(action = 'view') as views

from simulator\_20250520.feed\_actions

where toDate(time) between '2025-04-25' and '2025-05-01'

and exp\_group in (1, 2, 3, 4, 5)

group by exp\_group, user\_id

)

group by views

order by views

'''

views\_distribution = pandahouse.read\_clickhouse (q, connection=connection)

# Отношение произведения юзеров на просмотры к юзерам = распределение просмотров

(views\_distribution.users \* views\_distribution.views).sum()/views\_distribution.users.sum()



views\_distribution['p'] = views\_distribution['users']/views\_distribution.users.sum()

# Сортируем по значению распределения

views\_distribution.sort\_values(by = 'p', ascending =False)

# Считаем среднее значение распределения

(views\_distribution.views \* views\_distribution.p).sum()

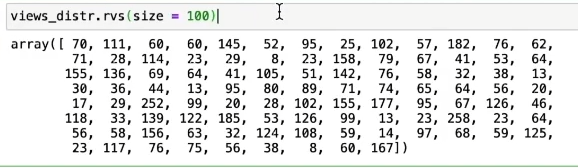
# Настоящее распределение просмотров

views\_distr = stats.rv\_discrete(name = 'view\_distr', values=(views\_distribution['views'], views\_distribution['p']))

np.mean(views\_distr.rvs(size = 10))



# Сэмплировать так



# Сэмплировать и посчитать среднее



# Теперь CTR. Подключаемся

q1 = '''

select floor(ctr, 2) as ctr, count() as users

from (

select

toDate(time) as dt,

exp\_group,

user\_id,

sum(if(action = 'like', 1, 0)) / sum(if(action = 'view', 1, 0)) as ctr

from simulator\_20250520.feed\_actions

where dt between '2025-04-25' and '2025-05-01'

and exp\_group in (1, 2, 3, 4, 5)

group by dt, exp\_group, user\_id

)

group by ctr

'''

ctr\_distribution = pandahouse.read\_clickhouse (q1, connection=connection)

ctr\_distribution['p'] = ctr\_distribution['users'] / ctr\_distribution.users.sum()

# Делаем распределение

probabilities = ctr\_distribution['p']

values = ctr\_distribution['ctr']

distrib = stats.rv\_discrete(values=(range(len(probabilities)), probabilities))

def get\_ctrs(x, y):

idx = distrib.rvs(size=x\*y)

result = values[idx]

return np.array(result).reshape(x,y)

# Переходим к синтетическому АА тесту

q3 = '''

select stddevPop(ctr) as ctr\_std

from (select exp\_group,

user\_id,

sum(action = 'like')/sum(action = 'view') as ctr

from simulator\_20250520.feed\_actions

where toDate(time) between '2025-04-25' and '2025-05-01'

and exp\_group in (1,2,3,4,5)

group by exp\_group, user\_id

)

'''

#Стандартное отклонение CTR

std = pandahouse.read\_clickhouse (q3, connection=connection).values[0][0]

std



# Считаем, какое получится n

compute\_sample\_size\_abs(0.01, std, std)



# **Попробуем 1447, 1600, 1800**

# Приведение к int64

group\_A\_views = views\_distr.rvs(size=(2000, 1100)).astype(np.int64)

group\_B\_views = views\_distr.rvs(size=(2000, 1100)).astype(np.int64)

# Получение CTR и контроль, чтобы не превышал 1.0

group\_A\_ctr = np.minimum(get\_ctrs(2000, 1100) \* 1.048, 1.0)

group\_B\_ctr = get\_ctrs(2000, 1100)

# Генерация количества кликов на основе биномиального распределения

clicks\_A = stats.binom.rvs(n=group\_A\_views, p=group\_A\_ctr)

clicks\_B = stats.binom.rvs(n=group\_B\_views, p=group\_B\_ctr)

# Среднее

* 1. / get\_ctrs(2000, 1800).mean()



def t\_test(a, b):

"""

Считает p-value для t-теста с двусторонней альтернативой

:param a: np.array вида (n\_experiments, n\_users), значения метрик в контрольных группах

:param b: np.array вида (n\_experiments, n\_users), значения метрик в тестовых группах

:return: np.array вида (n\_experiments), посчитанные p-value t-теста для всего списка экспериментов

"""

result = list(map(lambda x: stats.ttest\_ind(x[0], x[1]).pvalue, zip(a, b)))

return np.array(result)

np.sum(t\_test(clicks\_A / group\_A\_views, clicks\_B / group\_B\_views) <= 0.05) / 2000



**!!! Прокрасилось меньше 80%**

t\_test(clicks\_A, clicks\_B).shape



# **Проверяем**

def get\_1\_minus\_beta(N, alpha=0.05, std1=std, std2=std, epsilon=0.01):

z\_scores\_sum\_squared = N \* ((std1 \*\* 2 + std2 \*\* 2) / (epsilon \*\* 2))

t\_alpha\_plus\_tbeta = np.sqrt(z\_scores\_sum\_squared)

t\_alpha = norm.ppf(1 - alpha / 2, loc=0, scale=1)

t\_beta = t\_alpha\_plus\_tbeta - t\_alpha

\_1\_minus\_beta = norm.cdf(t\_beta, loc=0, scale=1)

return \_1\_minus\_beta

def get\_1\_minus\_beta\_t\_test(N, effect=1.047):

group\_A\_views = views\_distr.rvs(size=(1000, N)).astype(np.int64)

group\_B\_views = views\_distr.rvs(size=(1000, N)).astype(np.int64)

group\_A\_ctr = np.minimum(get\_ctrs(1000, N) \* effect, 1.0)

group\_B\_ctr = get\_ctrs(1000, N)

clicks\_A = stats.binom.rvs(group\_A\_views, group\_A\_ctr)

clicks\_B = stats.binom.rvs(group\_B\_views, group\_B\_ctr)

return np.sum(t\_test(clicks\_A / group\_A\_views, clicks\_B / group\_B\_views) <= 0.05) / 1000

get\_1\_minus\_beta\_t\_test(1447)



%%time

# Это я тут считаю долю "найденной" разницы

xs = np.arange(700, 2501, 100)

\_1\_minus\_beta\_normal = get\_1\_minus\_beta(xs)

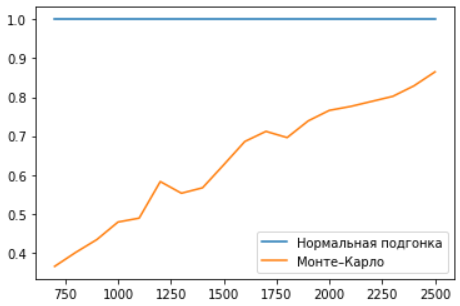
\_1\_minus\_beta\_monte\_carlo = np.array([get\_1\_minus\_beta\_t\_test(i) for i in xs])



\_ = plt.plot(xs, \_1\_minus\_beta\_normal, label='Нормальная подгонка')

\_ = plt.plot(xs, \_1\_minus\_beta\_monte\_carlo, label='Монте–Карло')

plt.legend()



**То есть тут мы не доложились!!!**

# Функция бакетного преобразования

def bucketization(ctrs\_0, weights\_0, ctrs\_1, weights\_1, n\_buckets=100):

"""

Разбиваем на бакеты с весами

:param ctrs\_0: np.array shape (n\_experiments, n\_users), CTRs of every user from control group in every experiment

:param weights\_0: np.array (n\_experiments, n\_users), веса пользователей в контрольной группе

:param ctrs\_1: np.array (n\_experiments, n\_users), CTRs of every user from treatment group in every experiment

:param weights\_1: np.array (n\_experiments, n\_users), веса пользователей в целевой группе

:param n\_buckets: int, кол-во бакетов

:return: np.array shape (n\_experiments), средневзвешенные метрики в каждом бакете

"""

n\_experiments, n\_users = ctrs\_0.shape

values\_0 = np.zeros((n\_experiments, n\_buckets))

values\_1 = np.zeros((n\_experiments, n\_buckets))

for b in np.arange(n\_buckets):

ind = np.arange(b \* n\_users / n\_buckets, b \* n\_users / n\_buckets + n\_users / n\_buckets).astype(int)

values\_0[:, b] = np.sum(ctrs\_0[:, ind] \* weights\_0[:, ind], axis=1) / np.sum(weights\_0[:, ind], axis=1)

values\_1[:, b] = np.sum(ctrs\_1[:, ind] \* weights\_1[:, ind], axis=1) / np.sum(weights\_1[:, ind], axis=1)

return values\_0, values\_1

def mannwhitney(a, b):

"""

Двусторонний тест Манна–Уитни

:param a: np.array вида (n\_experiments, n\_users), значения метрики в контроле

:param b: np.array вида (n\_experiments, n\_users), значения метрики в тесте

:return: np.array вида (n\_experiments), двусторонние p-value методом Манна–Уитни для всех экспериментов

"""

result = list(map(lambda x: stats.mannwhitneyu(

x[0], x[1], alternative='two-sided').pvalue, zip(a, b)))

return np.array(result)

# t-тест поверх бакетов

def t\_test\_buckets(ctrs\_0, weights\_0, ctrs\_1, weights\_1, n\_buckets=100):

return t\_test(\*bucketization(ctrs\_0, weights\_0, ctrs\_1, weights\_1, n\_buckets))

# тест Манна–Уитни поверх бакетов

def mannwhitney\_buckets(ctrs\_0, weights\_0, ctrs\_1, weights\_1, n\_buckets=100):

return mannwhitney(\*bucketization(ctrs\_0, weights\_0, ctrs\_1, weights\_1, n\_buckets))

def get\_1\_minus\_beta\_t\_test\_buckets(N, effect=1.047):

group\_A\_views = views\_distr.rvs(size=(2000, N)).astype(np.int64)

group\_B\_views = views\_distr.rvs(size=(2000, N)).astype(np.int64)

group\_A\_ctr = np.minimum(get\_ctrs(2000, N) \* effect, 1.0)

group\_B\_ctr = get\_ctrs(2000, N)

clicks\_A = stats.binom.rvs(group\_A\_views, group\_A\_ctr)

clicks\_B = stats.binom.rvs(group\_B\_views, group\_B\_ctr)

return np.sum(t\_test\_buckets(

clicks\_A / group\_A\_views,

group\_A\_views,

clicks\_B / group\_B\_views,

group\_B\_views

) <= 0.05) / 2000

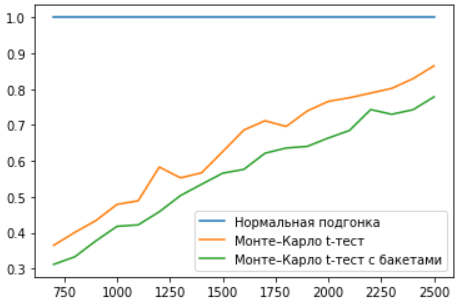
\_1\_minus\_beta\_monte\_carlo\_buckets = np.array([get\_1\_minus\_beta\_t\_test\_buckets(i) for i in xs])

\_ = plt.plot(xs, \_1\_minus\_beta\_normal, label='Нормальная подгонка')

\_ = plt.plot(xs, \_1\_minus\_beta\_monte\_carlo, label='Монте–Карло t-тест')

\_ = plt.plot(xs, \_1\_minus\_beta\_monte\_carlo\_buckets, label='Монте–Карло t-тест с бакетами')

plt.legend()



def get\_1\_minus\_beta\_linearized(N, effect=1.047):

group\_A\_views = views\_distr.rvs(size=(2000, N)).astype(np.int64)

group\_B\_views = views\_distr.rvs(size=(2000, N)).astype(np.int64)

group\_A\_ctr = np.minimum(get\_ctrs(2000, N) \* effect, 1.0)

group\_B\_ctr = get\_ctrs(2000, N)

clicks\_A = stats.binom.rvs(group\_A\_views, group\_A\_ctr)

clicks\_B = stats.binom.rvs(group\_B\_views, group\_B\_ctr)

control\_ctr = stats.binom.rvs(group\_B\_views, group\_B\_ctr).sum(axis=1) / group\_B\_views.sum(axis=1)

linearized\_A = (clicks\_A - (group\_A\_views.T \* control\_ctr).T)

linearized\_B = (clicks\_B - (group\_B\_views.T \* control\_ctr).T)

linearization\_p\_values = t\_test(linearized\_A, linearized\_B)

return np.sum(linearization\_p\_values <= 0.05) / 2000

\_1\_minus\_beta\_monte\_carlo\_linearization = np.array([get\_1\_minus\_beta\_linearized(i) for i in xs])

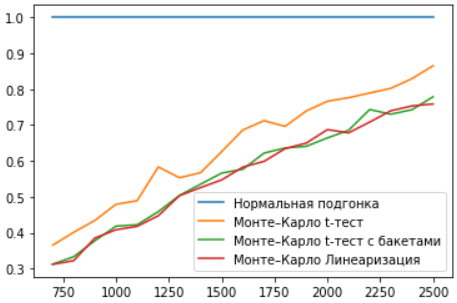
\_ = plt.plot(xs, \_1\_minus\_beta\_normal, label='Нормальная подгонка')

\_ = plt.plot(xs, \_1\_minus\_beta\_monte\_carlo, label='Монте–Карло t-тест')

\_ = plt.plot(xs, \_1\_minus\_beta\_monte\_carlo\_buckets, label='Монте–Карло t-тест с бакетами')

\_ = plt.plot(xs, \_1\_minus\_beta\_monte\_carlo\_linearization, label='Монте–Карло Линеаризация')

plt.legend()



**С НЕНОРМАЛЬНЫМ РАСПРЕДЕЛЕНИЕМ**

import pandahouse

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

import numpy as np

import hashlib

import swifter

from scipy import stats

from scipy.stats import norm, ttest\_ind

%matplotlib inline

connection = {

'host': 'https://clickhouse.lab.karpov.courses',

'password': 'dpo\_python\_2020',

'user': 'student',

'database': 'simulator'

}

q1 = '''

select floor(ctr, 2) as ctr, count() as users

from (

select

toDate(time) as dt,

exp\_group,

user\_id,

sum(if(action = 'like', 1, 0)) / sum(if(action = 'view', 1, 0)) as ctr

from simulator\_20250520.feed\_actions

where dt between '2025-04-25' and '2025-05-01'

and exp\_group in (1, 2, 3, 4, 5)

group by dt, exp\_group, user\_id

)

group by ctr

'''

ctr\_distribution = pandahouse.read\_clickhouse (q1, connection=connection)

ctr\_distribution['p'] = ctr\_distribution['users'] / ctr\_distribution.users.sum()

# Делаем распределение

probabilities = ctr\_distribution['p']

values = ctr\_distribution['ctr']

distrib = stats.rv\_discrete(values=(range(len(probabilities)), probabilities))

def get\_ctrs(x, y):

idx = distrib.rvs(size=x\*y)

result = values[idx]

return np.array(result).reshape(x,y)

mu = 1

sigma2 = 2

N = 1447

experiments = 1000

views\_lognormal = np.absolute(np.exp(norm(mu, sigma2).rvs(

experiments \* N))).astype(np.int64).reshape(experiments, N) + 1

fig = plt.figure()

ax\_views = fig.add\_subplot(1, 1, 1)

views\_distr = sns.histplot(views\_lognormal.ravel(), # Теперь это массив длины experiments \* N

bins=range(0, 30),

kde=False,

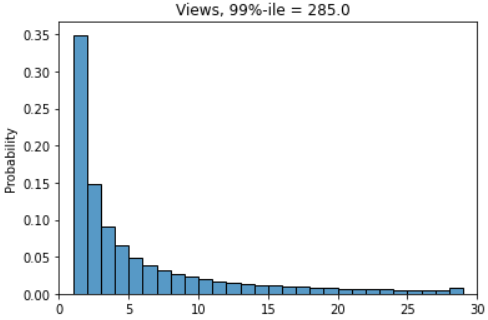
ax=ax\_views,

stat='probability')

ax\_views.set\_xlim((0, 30))

views\_99\_percentile = np.percentile(views\_lognormal.ravel(), 99)

ax\_views.set\_title(f'Views, 99%-ile = {views\_99\_percentile:<7.1f}')



def get\_views(experiments, N, mu=1, sigma=2):

views\_lognormal = np.absolute(

np.exp(

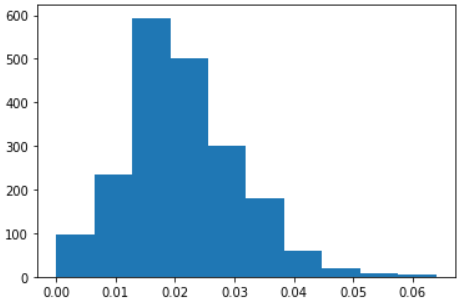
norm(mu, sigma).rvs(experiments \* N)

).astype(np.int64).reshape(experiments, N) + 1

)

return views\_lognormal

plt.hist(get\_ctrs(2000, 1)/10)



group\_A\_views = get\_views(2000, 10000).astype(np.int64)

group\_B\_views = get\_views(2000, 10000).astype(np.int64)

group\_A\_ctr = np.minimum(get\_ctrs(2000, 10000) \* 1.2, 1.0)

group\_B\_ctr = get\_ctrs(2000, 10000)

clicks\_A = stats.binom.rvs(group\_A\_views, group\_A\_ctr)

clicks\_B = stats.binom.rvs(group\_B\_views, group\_B\_ctr)

group\_B\_ctr.std()



# Новое std на нормализованных данных

my\_new\_std = (clicks\_B / group\_B\_views).std()

my\_new\_std



# Функция формулы калькулятора

def compute\_sample\_size\_abs(epsilon, std1, std2, alpha=0.05, beta=0.2):

t\_alpha = norm.ppf(1 - alpha / 2, loc=0, scale=1)

t\_beta = norm.ppf(1 - beta, loc=0, scale=1)

z\_scores\_sum\_squared = (t\_alpha + t\_beta) \*\* 2

sample\_size = int(

np.ceil(

z\_scores\_sum\_squared \* (std1 \*\* 2 + std2 \*\* 2) / (epsilon \*\* 2)

)

)

return sample\_size

compute\_sample\_size\_abs(0.005, my\_new\_std, my\_new\_std)



def t\_test(a, b):

"""

Считает p-value для t-теста с двусторонней альтернативой

:param a: np.array вида (n\_experiments, n\_users), значения метрик в контрольных группах

:param b: np.array вида (n\_experiments, n\_users), значения метрик в тестовых группах

:return: np.array вида (n\_experiments), посчитанные p-value t-теста для всего списка экспериментов

"""

result = list(map(lambda x: stats.ttest\_ind(x[0], x[1]).pvalue, zip(a, b)))

return np.array(result)

# T-test

np.sum(t\_test(clicks\_A / group\_A\_views, clicks\_B / group\_B\_views) <= 0.05) / 2000



def mannwhitney(a, b):

"""

Двусторонний тест Манна–Уитни

:param a: np.array вида (n\_experiments, n\_users), значения метрики в контроле

:param b: np.array вида (n\_experiments, n\_users), значения метрики в тесте

:return: np.array вида (n\_experiments), двусторонние p-value методом Манна–Уитни для всех экспериментов

"""

result = list(map(lambda x: stats.mannwhitneyu(

x[0], x[1], alternative='two-sided').pvalue, zip(a, b)))

return np.array(result)

# t-тест поверх бакетов

def t\_test\_buckets(ctrs\_0, weights\_0, ctrs\_1, weights\_1, n\_buckets=100):

return t\_test(\*bucketization(ctrs\_0, weights\_0, ctrs\_1, weights\_1, n\_buckets))

# тест Манна–Уитни поверх бакетов

def mannwhitney\_buckets(ctrs\_0, weights\_0, ctrs\_1, weights\_1, n\_buckets=100):

return mannwhitney(\*bucketization(ctrs\_0, weights\_0, ctrs\_1, weights\_1, n\_buckets))

def bucketization(ctrs\_0, weights\_0, ctrs\_1, weights\_1, n\_buckets=100):

"""

Разбиваем на бакеты с весами

:param ctrs\_0: np.array shape (n\_experiments, n\_users), CTRs of every user from control group in every experiment

:param weights\_0: np.array (n\_experiments, n\_users), веса пользователей в контрольной группе

:param ctrs\_1: np.array (n\_experiments, n\_users), CTRs of every user from treatment group in every experiment

:param weights\_1: np.array (n\_experiments, n\_users), веса пользователей в целевой группе

:param n\_buckets: int, кол-во бакетов

:return: np.array shape (n\_experiments), средневзвешенные метрики в каждом бакете

"""

n\_experiments, n\_users = ctrs\_0.shape

values\_0 = np.zeros((n\_experiments, n\_buckets))

values\_1 = np.zeros((n\_experiments, n\_buckets))

for b in np.arange(n\_buckets):

ind = np.arange(b \* n\_users / n\_buckets, b \* n\_users / n\_buckets + n\_users / n\_buckets).astype(int)

values\_0[:, b] = np.sum(ctrs\_0[:, ind] \* weights\_0[:, ind], axis=1) / np.sum(weights\_0[:, ind], axis=1)

values\_1[:, b] = np.sum(ctrs\_1[:, ind] \* weights\_1[:, ind], axis=1) / np.sum(weights\_1[:, ind], axis=1)

return values\_0, values\_1

# Бакетный Т-тест

np.sum(t\_test\_buckets(

clicks\_A / group\_A\_views,

group\_A\_views,

clicks\_B / group\_B\_views,

group\_B\_views

) <= 0.05) / 2000



def get\_1\_minus\_beta\_t\_test\_lognormal(N, effect=1.2, divide\_by=10):

group\_A\_views = get\_views(2000, N).astype(np.int64)

group\_B\_views = get\_views(2000, N).astype(np.int64)

#group\_A\_ctr = np.minimum(get\_ctrs(2000, N) \* effect, 1.0)

group\_A\_ctr = np.minimum(get\_ctrs(2000, N) / divide\_by \* effect, 1.0)

group\_B\_ctr = get\_ctrs(2000, N) / divide\_by

clicks\_A = stats.binom.rvs(group\_A\_views, group\_A\_ctr)

clicks\_B = stats.binom.rvs(group\_B\_views, group\_B\_ctr)

return np.sum(t\_test(clicks\_A / group\_A\_views, clicks\_B / group\_B\_views) <= 0.05) / 2000

def get\_1\_minus\_beta\_t\_test\_buckets\_lognormal(N, effect=1.2, divide\_by=10):

group\_A\_views = get\_views(2000, N).astype(np.int64)

group\_B\_views = get\_views(2000, N).astype(np.int64)

group\_A\_ctr = np.minimum(get\_ctrs(2000, N) / divide\_by \* effect, 1.0)

group\_B\_ctr = get\_ctrs(2000, N) / divide\_by

clicks\_A = stats.binom.rvs(group\_A\_views, group\_A\_ctr)

clicks\_B = stats.binom.rvs(group\_B\_views, group\_B\_ctr)

return np.sum(t\_test\_buckets(

clicks\_A / group\_A\_views,

group\_A\_views,

clicks\_B / group\_B\_views,

group\_B\_views

) <= 0.05) / 2000

def get\_1\_minus\_beta\_mw\_buckets\_lognormal(N, effect=1.2, divide\_by=10):

group\_A\_views = get\_views(2000, N).astype(np.int64)

group\_B\_views = get\_views(2000, N).astype(np.int64)

group\_A\_ctr = np.minimum(get\_ctrs(2000, N) / divide\_by \* effect, 1.0)

group\_B\_ctr = get\_ctrs(2000, N) / divide\_by

clicks\_A = stats.binom.rvs(group\_A\_views, group\_A\_ctr)

clicks\_B = stats.binom.rvs(group\_B\_views, group\_B\_ctr)

return np.sum(mannwhitney\_buckets(

clicks\_A / group\_A\_views,

group\_A\_views,

clicks\_B / group\_B\_views,

group\_B\_views

) <= 0.05) / 2000

%%time

xs = np.arange(1000, 12001, 1000)

\_1\_minus\_beta\_t\_test = np.array([get\_1\_minus\_beta\_t\_test\_lognormal(i, divide\_by=10) for i in xs])

\_1\_minus\_beta\_t\_test\_buckets = np.array([get\_1\_minus\_beta\_t\_test\_buckets\_lognormal(i, divide\_by=10) for i in xs])

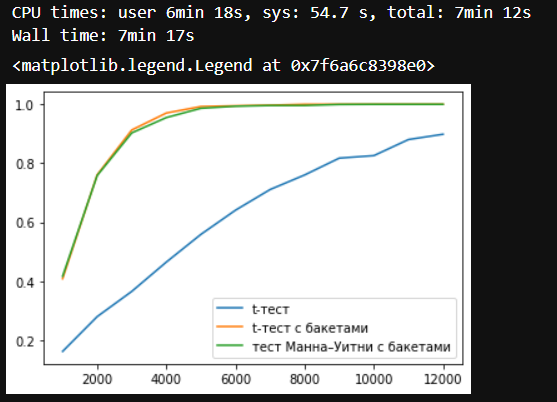
\_1\_minus\_beta\_mw\_buckets = np.array([get\_1\_minus\_beta\_mw\_buckets\_lognormal(i, divide\_by=10) for i in xs])

\_ = plt.plot(xs, \_1\_minus\_beta\_t\_test, label='t-тест')

\_ = plt.plot(xs, \_1\_minus\_beta\_t\_test\_buckets, label='t-тест с бакетами')

\_ = plt.plot(xs, \_1\_minus\_beta\_mw\_buckets, label='тест Манна–Уитни с бакетами')

plt.legend()



Поменяйте соответствующую строчку на это:

group\_B\_views = group\_B\_views + (rng.binomial(n=1, p=0.9, size=num\_group) \* (1 + rng.binomial(n=1, p=0.5, size=num\_group)) \* (group\_B\_views >= 50))