Методы повышения чувствительности

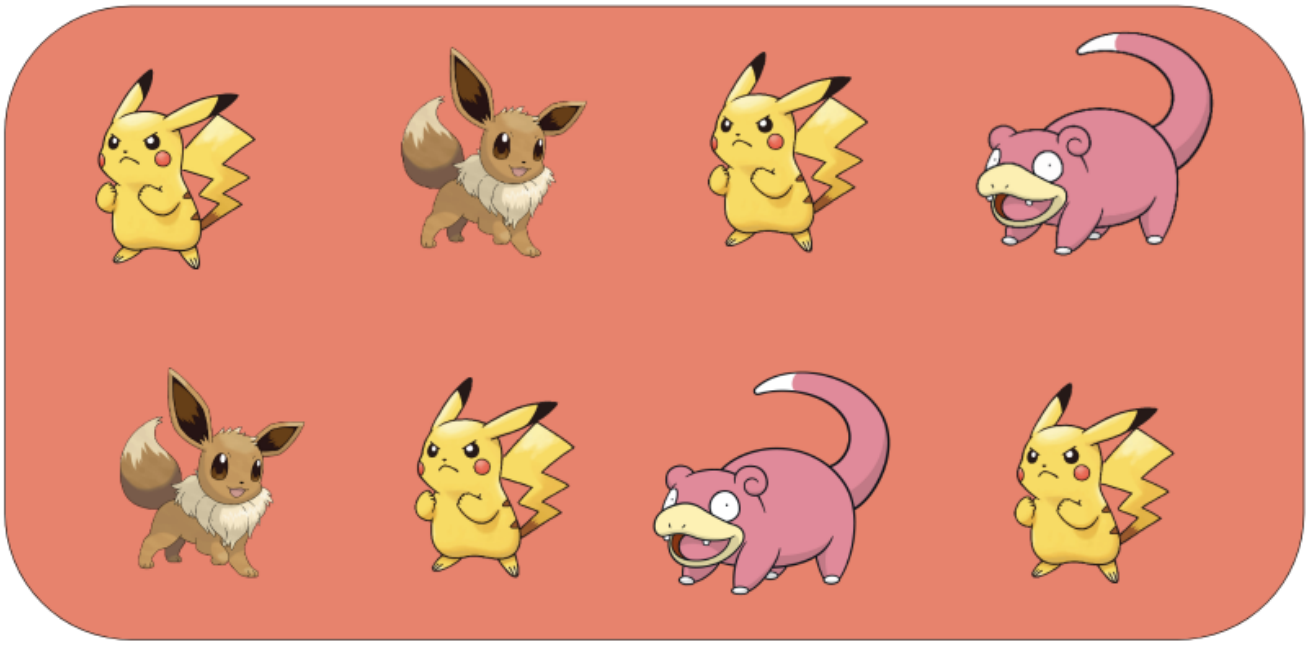
При проведении АБ тестов мы хотим убедиться, что у наших статистических оценщиков достаточно мощности для обнаружения эффекта. Самый простой метод – увеличить количество наблюдений, однако это сложно сделать из-за ограниченности трафика. К тому же, MDE пропорционален $1/\sqrt{N}$, поэтому увеличение мощности таким образом ограничено. Другой способ – увеличение размера эффекта, это может быть не применимо в зависимости от эффекта. Поэтому рассмотрим способ через снижение дисперсии.

Стратификация

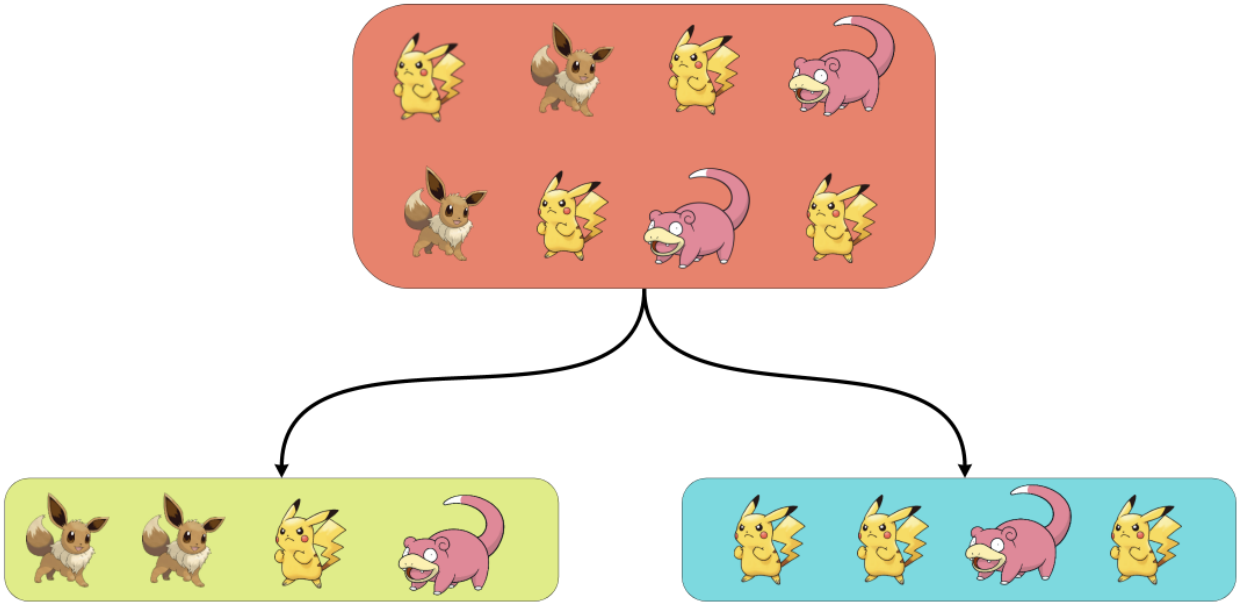
Пре-стратификация

Хорошее описание идеи (отсюда):

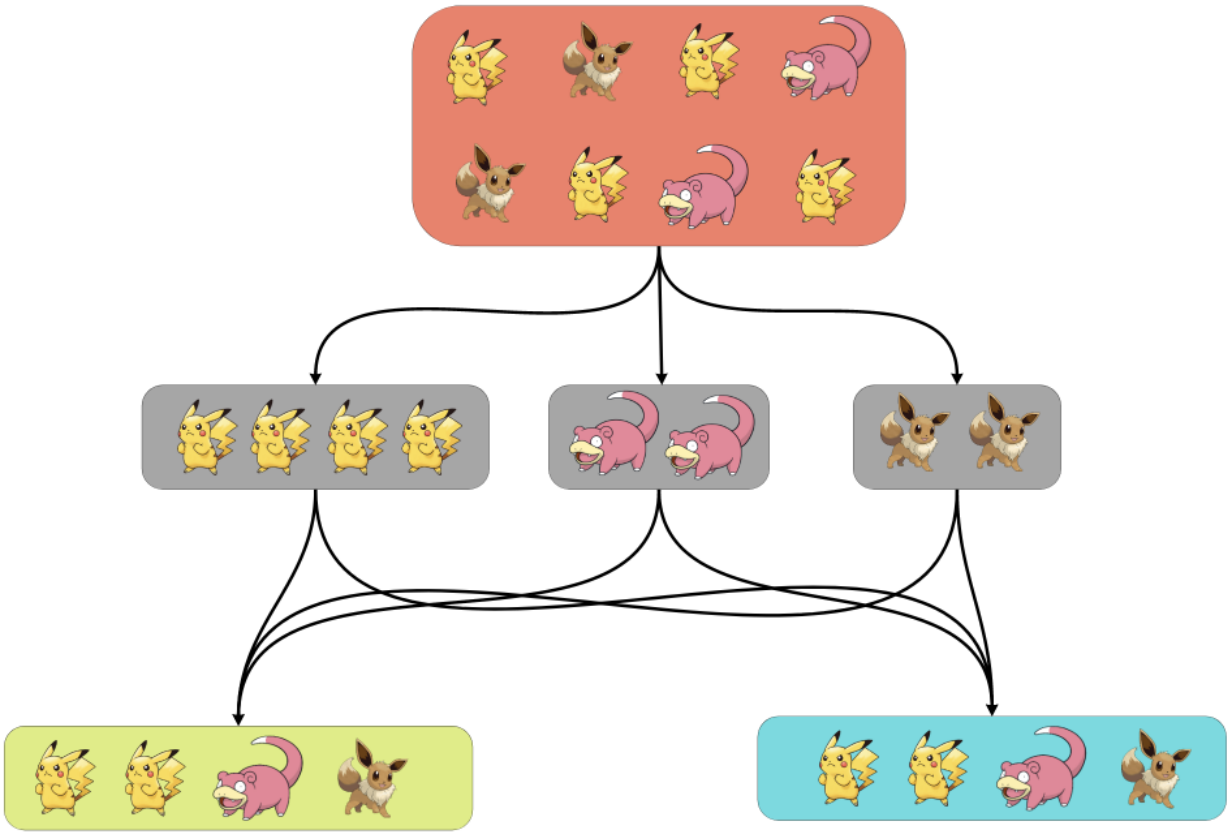
Рассмотрим самый простой пример для визуализации: пусть у нас есть генеральная совокупность пользователей-покемонов:



Мы захотели провести на них A/B-тест. К примеру, раздать скидки клиентам СберМаркета. При обычном A/B-тестировании мы случайно разбиваем всю выборку на тест и контроль, к примеру так:



Дальше к этому мы применяем t-test, бутстрап, CUPED и т.д. и считаем результаты. Но вопрос: а что, если Пикачу (жёлтенькие) реагируют на тритмент не так, как Слоупоки (розовенькие)? Это вносит дополнительный шум в данные, так как в одной выборке три Пикачу, а в другой — один. Поэтому давайте добавим вспомогательный шаг при делении выборки на тест и контроль. Сначала сгруппируем всех покемонов по виду, а потом будем сэмплировать из каждой группы (или страты) половину покемонов в тест, а другую — в контроль. Этот метод и называется стратификацией.



Теоретическое обоснование понижения дисперсии:

Дисперсия стратифицированной выборки состоит из взвешенных дисперсий внутри страт. А дисперсия при случайном (обычном) разбиении состоит из дисперсии стратифицированной выборки и взвешенной «дисперсии между стратами». Таким образом, уменьшение дисперсии происходит за счёт “выкидывания” дисперсии между стратами

$$\begin{aligned}\operatorname{var}(\overline{Y}) &= \sum_{k=1}^{K} \frac{w_k}{n}\sigma_k^2 + \sum_{k=1}^{K} \frac{w_k}{n}(\mu_k - \mu)^2 \\ &\geq \sum_{k=1}^{K} \frac{w_k}{n}\sigma_k^2 = \operatorname{var}(\widehat{Y}_{strat})\end{aligned}$$

Более детальное доказательство в источнике.

Как реализовывать (идея):

1. Собираем в один dataframe метрики, по которым будем стратифицировать. Это могут быть любые метрики, как категориальные (например, город, пол (если речь про юзеров), ретейлер (если речь про магазины)), так и вещественные (GMV (/per smth), num orders, AOV, etc).
2. Делим на бины (переводим в категориальные) вещественные метрики

3. Затем выделяем страты (по страте на каждый уникальный вектор бинаризованных метрик + категориальных)
4. Внутри каждой страты рандомно распределяем юзеров на тест и контроль. Таким образом каждая страта представлена в тесте и контроле одинаково и пропорционально генеральной совокупности.

Пример кода (на примере теста по магазинам):

```
  def bucketization(df, num_metrics, n_bins):
    data = df.copy()
    bin_metrics = []
    for metric in num_metrics:
        data[metric + "_bin"] = pd.cut(data[metric].values, bins=n_bins)
        bin_metrics.append(metric + "_bin")
    return data, bin_metrics

def clusterization(df, bin_metrics, cat_metrics):
    data = df.copy()
    cluster_metrics = bin_metrics + cat_metrics
    all_clusters = data[cluster_metrics].drop_duplicates().copy()
    all_clusters["num_cluster"] = range(all_clusters.shape[0])
    data = data.merge(all_clusters, on=cluster_metrics)
    return data

def get_stratified_splits(df, id_column, cluster_column="num_cluster", n_splits=50, group_size=100):

    final_split = {i:{'control' : {},'test' : {}} for i in range(n_splits)}
    for split in tqdm(range(n_splits)):
        data = df.sample(n=2*group_size, replace=False).copy()
        cluster_names = data[cluster_column].unique()
        test_group_samples = []
        control_group_samples = []
        for cluster in cluster_names:
            cluster_samples = data[data[cluster_column]==cluster]
            size = cluster_samples.shape[0]//2
            if size == 0:
                size = np.random.choice([0,1])

            test_samples = np.random.choice(cluster_samples[id_column].values, size=size, replace=False)
            test_group_samples.extend(test_samples)
            control_group_samples.extend(cluster_samples[~cluster_samples[id_column].isin(test_samples)]
[id_column].tolist())
        final_split[split]['control'][id_column] = np.array(control_group_samples)
        final_split[split]['test'][id_column] = np.array(test_group_samples)
    return final_split

num_metrics = ['not_found', 'conversion', 'check', 'gmv_net_of_promo',]
cat_metrics = ["retailer_id",]
ID_COLUMN = "store_id" #   of randomization unit
N_BINS=4
N_SPLITS = 1
group_size = 500

bin_df, bin_metrics = bucketization(df=stores_df, num_metrics=num_metrics, n_bins=N_BINS)
clustered_df = clusterization(df=bin_df, bin_metrics=bin_metrics, cat_metrics=cat_metrics)
final_split = get_stratified_splits(df=clustered_df, id_column=ID_COLUMN, n_splits=N_SPLITS,
group_size=group_size)
```

Пост-стратификация

Мотивация: как мы увидели, стратификация – это хороший способ понижения дисперсии. Но есть проблема – в случае online экспериментов (когда мы не знаем тестовую и контрольную выборку до начала эксперимента) мы не можем делать пре-стратификацию. Точнее, в теории, можем, но это дорого и сложно. Обойти эту проблему нам частично помогает техника пост-стратификации. Она описана тут, тут, и частично тут.

Механика:

1. Присвоим группы рандомно
2. Добавим в линейную регрессию категориальную переменную для страты, то есть перевесим оценку

Куда идти за деталями (источники):

1. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3140631
2. https://www.kdd.org/kdd2016/subtopic/view/improving-the-sensitivity-of-online-controlled-experiments-case-studies-at-
3. https://towardsdatascience.com/online-experiments-tricks-variance-reduction-291b6032dcd7
4. https://exp-platform.com/Documents/2013-02-CUPED-ImprovingSensitivityOfControlledExperiments.pdf
5. https://habr.com/ru/company/avito/blog/571096/

CUPED

CUPED (Controlled-experiment Using Pre-Experiment Data) – техника, позволяющая увеличить чувствительность метрик за счет использования данных, полученных до начала эксперимента.

Идея довольно проста: мы берем какую-то метрику, на которую мы не повлияли нашим экспериментом, и с помощью этой метрики "объясняем" часть нашей экспериментальной метрики, понижая таким образом дисперсию наших оценок экспериментального эффекта. Таким образом, у нас два требования к искомой метрике – то, что на нее не повлиял эксперимент, и то, что она может объяснить экспериментальную метрику. Самым логичным примером является наш Y, взятый за период, предшествующий эксперименту.

В русскоязычном интернете существует довольно много описаний реализации этой техники, однако не все из них являются корректными. При неправильной реализации мы рискуем получить смещение (угасание) коэффициентом, а также недостаточное понижение дисперсии. Ниже я привожу корректный пример. Симуляции показывают, что оценки эффекта, полученные с применением CUPED'а в такой реализации, являются несмещенными.

Корректная техника:

1. Собираем датасет следующего содержания: (Метрика (Y), Метрика (или метрики) за период, предшествующий эксперименту (Cov), индикатор попадания в тестовую группу (treatment), константа (const))
2. Выделяем контрольную группу и оцениваем коэффициенты регрессии Y_control ~ Const + Cov, затем с помощью полученных коэффициентов предсказываем всю нашу выборку, вектор предсказаний обозначаем за Y_hat.
3. Преобразовываем исходный Y: Y_cuped = Y - Y_hat
4. Оцениваем эффект его статистическую значимость на Y_cuped (с помощью регрессии / t-test'а). Полученные оценки будут состоятельными и несмещенными, а дисперсия этих оценок (мощность теста) будет существенно ниже (выше).

Пример кода:

```
import statsmodels.api as sm
import pandas as pd

y = df["y"]
X_cov = df.drop(["y"], axis=1)

y_control = df.query("treatment==0")['y']
X_cov_control = df.query("treatment==0").drop(["y","treatment"],axis=1)

y_hat = sm.OLS(y_control, X_cov_control).fit().predict(X_cov.drop(["treatment"], axis=1))
sample_df['y_cuped'] = y.values - y_hat
model = sm.OLS(sample_df["y_cuped"], X_cov[["const","treatment"]]).fit()

print(model.summary())
```

Куда идти за деталями (источники):

1. https://exp-platform.com/Documents/2013-02-CUPED-ImprovingSensitivityOfControlledExperiments.pdf– оригинальная статья
2. https://habr.com/ru/company/avito/blog/571096/ – блог от Авито. Тут есть немного про некорректные имплементации.
3. https://www.youtube.com/watch?v=jPysoXa3udU – доклад от X5 (и статья в описании). Тут есть про *biased имплементацию CUPED'а.* А еще в описание есть статья от того же господина, там детальные выводы bias'а.
4. https://bytepawn.com/tag/ab-testing.html – блог про A/B тесты, симуляции и много умного про CUPED
5. Тык – Бабушкин рассказывает, как накосячил в своем докладе о CUPED

Пример реализации поюзерного теста с CUPED

Актуальная версия здесь.

Симуляции на синтетике

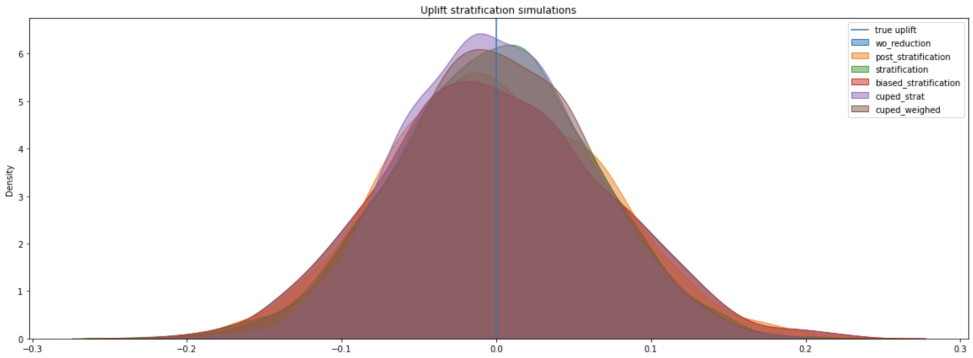
TLDR: из протестированных методов лучший в соотношении variance-bias CUPED с пред-стратификацией и без перевзвешивания.

Тестируемые методы:

- **wo_reduction** – разница средних в тесте и контроле
- **post_stratification** – рандомное присвоение тритмента, регрессия с категориальными признаками-индикаторами страты

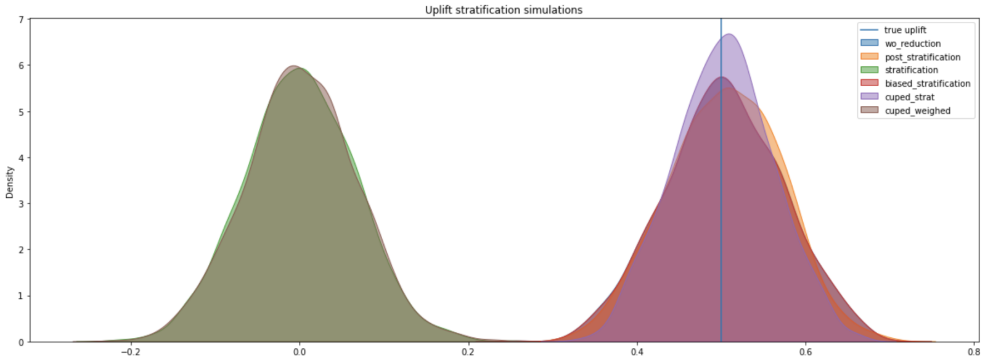
- **stratification** – стратифицированная рандомизация, регрессия с категориальными признаками-индикаторами страты
- **biased_stratification** – стратифицированная рандомизация, оценка эффекта через регрессию без категориальных признаков-индикаторов страт
- **cuped_strat** – стратифицированная рандомизация, CUPED fit control only без учета страт
- **cuped_weighted** – стратифицированная рандомизация, CUPED fit control only с учетом страт

Генерируем выборку из 1000 наблюдений, 5 ковариат и не добавляем эффект.



	wo_reduction	post_stratification	stratification	biased_stratification	cuped_strat	cuped_weighed
variance_reduction	1.000000	0.939546	0.795757	0.999437	0.677909	0.776436
bias	0.001015	0.000672	0.001443	0.001074	0.000846	0.001237
std	0.071386	0.069195	0.063680	0.071366	0.058776	0.062902
mean	-0.001015	-0.000672	-0.001443	-0.001074	-0.000846	-0.001237
lvl_05	0.039000	0.042000	0.051000	0.039000	0.151000	0.086000
lvl_01	0.009000	0.005000	0.011000	0.009000	0.060000	0.023000
CI_len	0.286689	0.279109	0.248950	0.286696	0.167950	0.213911

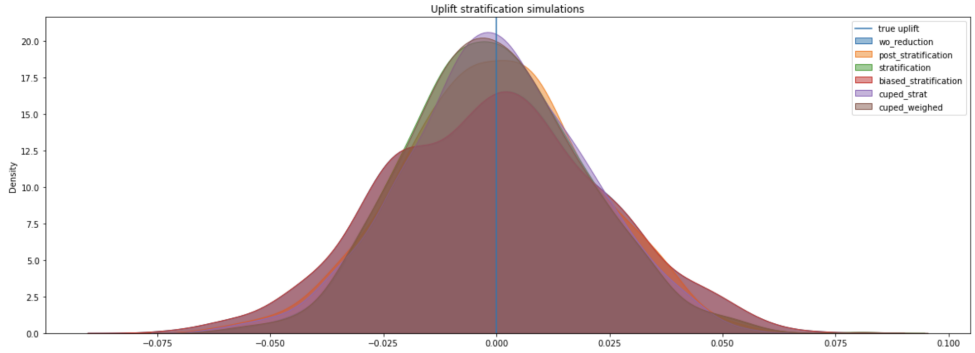
Генерируем выборку из 1000 наблюдений, 5 ковариат, добавим эффект от тритмента в тестовой группе.



	wo_reduction	post_stratification	stratification	biased_stratification	cuped_strat	cuped_weighed
variance_reduction	1.000000	0.932539	0.896587	0.997671	0.710740	0.898564
bias	-0.003255	-0.004245	0.499792	-0.003242	-0.001878	0.498815
std	0.069381	0.067000	0.065695	0.069300	0.058492	0.065768
mean	0.503255	0.504245	0.000208	0.503242	0.501878	0.001185
lvl_05	1.000000	1.000000	0.053000	1.000000	1.000000	0.098000
lvl_01	1.000000	1.000000	0.011000	1.000000	1.000000	0.027000
CI_len	0.286610	0.279148	0.256996	0.286614	0.167893	0.218649

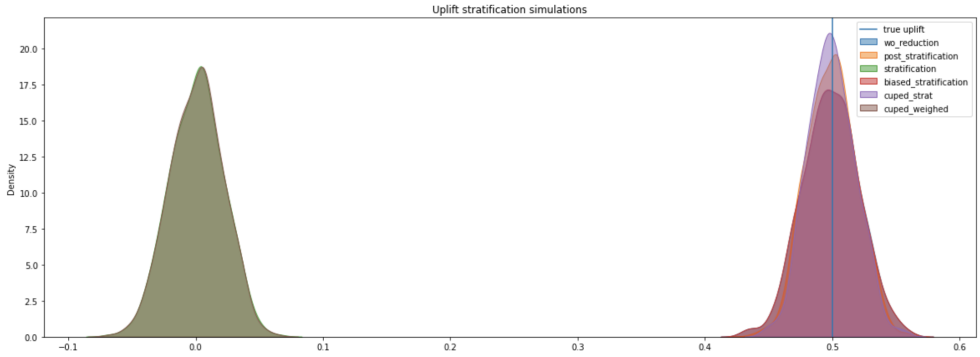
- CUPED со стратифицированной рандомизацией снижает дисперсию на 30% и не генерирует смещение. Доверительный интервал оценки наименьший.
- CUPED со стратифицированной рандомизацией и дальнейшим перевешиванием хуже сокращает дисперсию и генерирует смещение.

Генерируем выборку из 10000 наблюдений, 5 ковариат, не добавляем эффект.



	wo_reduction	post_stratification	stratification	biased_stratification	cuped_strat	cuped_weighed
variance_reduction	1.000000	0.744168	0.671970	1.002155	0.688064	0.675651
bias	0.000652	0.000103	0.000272	0.000674	0.000290	0.000277
std	0.023948	0.020658	0.019631	0.023973	0.019864	0.019684
mean	-0.000652	-0.000103	-0.000272	-0.000674	-0.000290	-0.000277
lvl_05	0.059000	0.056000	0.045000	0.059000	0.181000	0.098000
lvl_01	0.011000	0.012000	0.014000	0.011000	0.082000	0.033000
CI_len	0.090543	0.077891	0.076956	0.090543	0.053084	0.064344

Генерируем выборку из 10000 наблюдений, 5 ковариат, добавляем эффект.



	wo_reduction	post_stratification	stratification	biased_stratification	cuped_strat	cuped_weighed
variance_reduction	1.000000	0.774267	0.905577	0.999719	0.708470	0.893260
bias	0.001353	0.001039	0.499300	0.001426	0.000860	0.499335
std	0.021962	0.019325	0.020899	0.021958	0.018485	0.020756
mean	0.498647	0.498961	0.000700	0.498574	0.499140	0.000665
lvl_05	1.000000	1.000000	0.048000	1.000000	1.000000	0.122000
lvl_01	1.000000	1.000000	0.012000	1.000000	1.000000	0.030000
CI_len	0.090565	0.077911	0.079417	0.090564	0.053084	0.065685

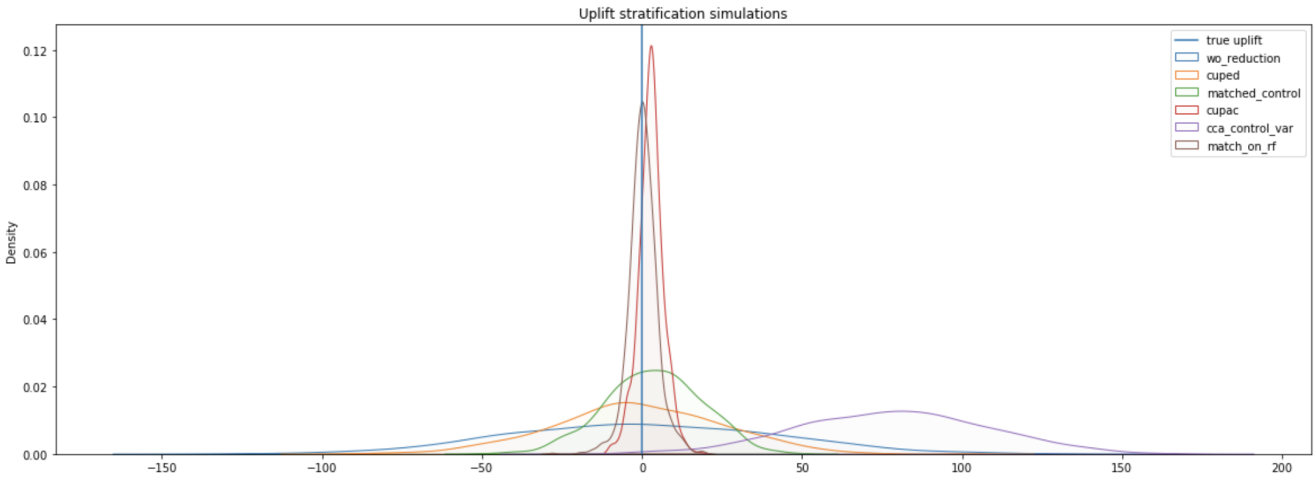
Ноутбук:

variance_reduction.ipynb

Мэтчинги и CUPED с ковариатами

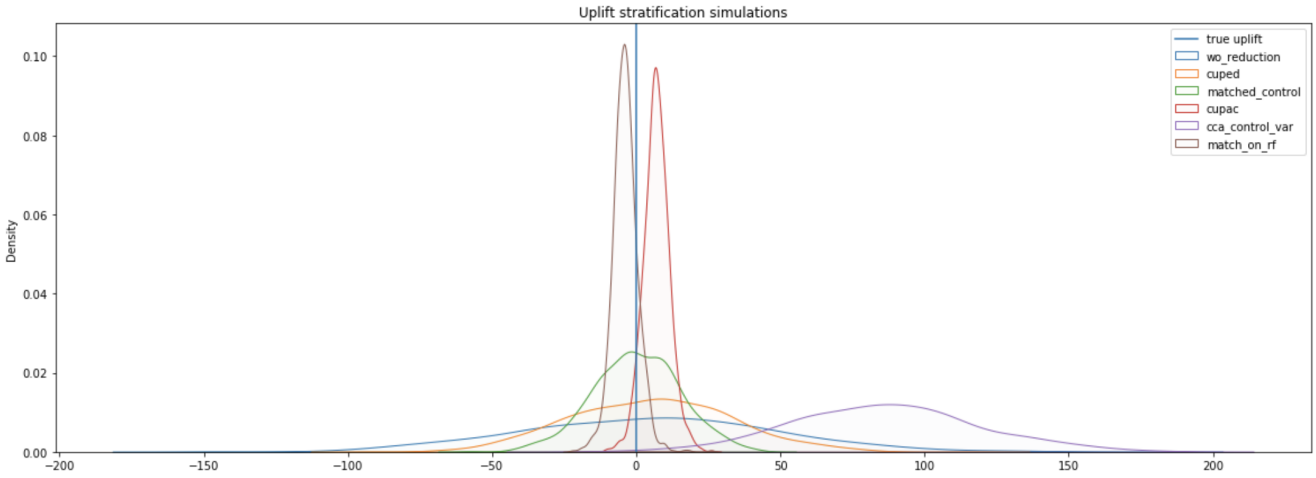
Целевая метрика на синтетических данных задавалась нелинейно, с использованием страт.

Без эффекта



	wo_reduction	cuped	matched_control	cupac	cca_control_var	match_on_rf
variance_reduction	1.000000	0.406707	0.128339	0.008500	0.503601	0.011623
bias	0.000000	1.680690	-3.254759	-2.484847	-78.000338	0.005979
std	42.724160	27.246731	15.305679	3.938943	30.319137	4.606189
mean	0.168638	-1.512052	3.423396	2.653485	78.168976	0.162659
lvl_05	0.052000	0.166000	0.076000	0.439000	0.723000	0.100000
lvl_01	0.012000	0.069000	0.020000	0.269000	0.504000	0.015000
CI_len	168.445498	75.483513	NaN	8.706057	119.105272	NaN

С эффектом



	wo_reduction	cuped	matched_control	cupac	cca_control_var	match_on_rf
variance_reduction	1.000000	0.414268	0.131861	0.009575	0.510027	0.012471
bias	0.000000	-1.472933	-5.015807	-4.034590	-80.796145	-1.424525
std	42.325836	27.242457	15.369635	4.141756	30.227490	4.726759
mean	-1.312738	0.160195	3.703069	2.721853	79.483407	0.111787
lvl_05	0.048000	0.169000	0.088000	0.425000	0.749000	0.093000
lvl_01	0.005000	0.073000	0.021000	0.279000	0.526000	0.020000
CI_len	168.516490	75.510751	NaN	9.045894	119.157762	NaN

Ноутбук

new_variance (2).ipynb

Симуляции на поюзерных данных

TLDR: CUPED с несколькими ковариатами больше снижает дисперсию и имеет меньшее смещение, чем CUPED с одной ковариатой.

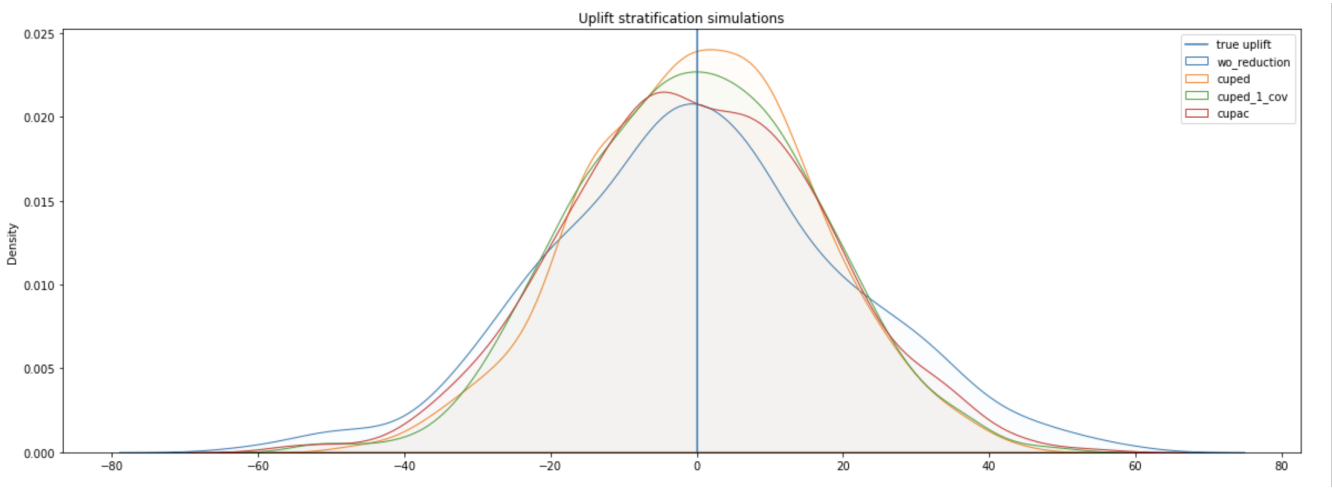
Целевая метрика: gmv per user

Признаки: gmv per user, кол-во заказов, "активность" (кол-во событий) пользователя 2/4 недели назад

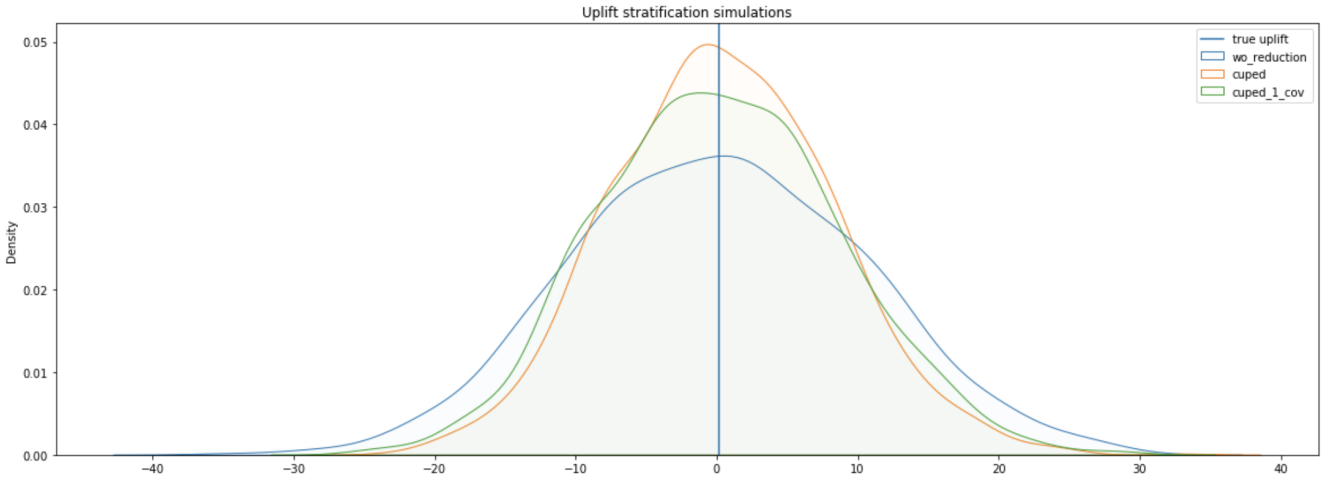
Формирование групп: рандомно присваивалось 25% траффика для каждой группы

Методы: CUPED с одной ковариатой – таргетом 2 недели назад; CUPED; CUPAC

Без эффекта

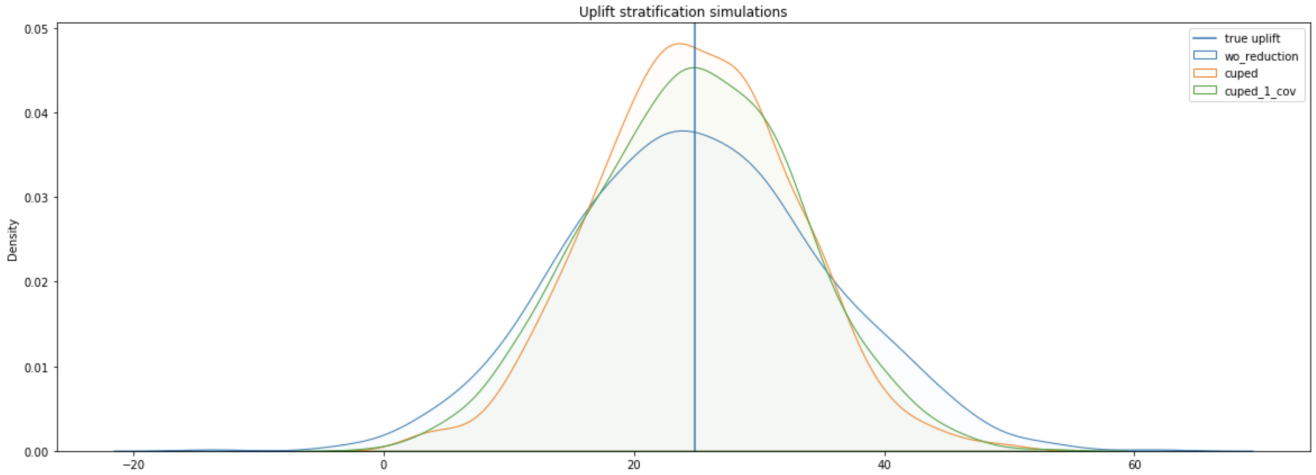


Добавление новых признаков, без эффекта



	wo_reduction	cuped	cuped_1_cov
variance_reduction	1.000000	0.573973	0.678328
bias	0.000000	-0.244869	-0.023258
std	10.322766	7.820630	8.501900
mean	0.176837	0.421706	0.200094
lvl_05	0.044000	0.053000	0.051000
lvl_01	0.008000	0.009000	0.011000
CI_len	40.970807	30.517857	33.140982

Добавление новых признаков, 2% эффект



	wo_reduction	cuped	cuped_1_cov
variance_reduction	1.000000	0.598420	0.678184
bias	0.000000	0.080537	0.140793
std	10.188809	7.881819	8.390684
mean	24.846337	24.765800	24.705544
lvl_05	0.662000	0.887000	0.819000
lvl_01	0.415000	0.724000	0.636000
CI_len	40.956606	30.495591	33.129989

Ноутбук:

user_data_new_te...niques (1).ipynb

Симуляции на данных магазинов

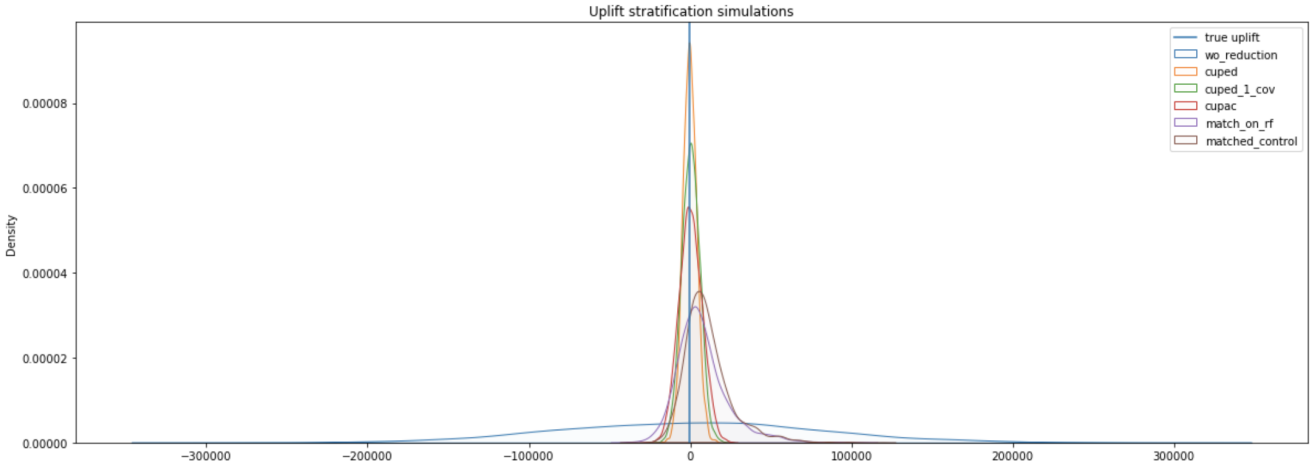
TLDR: CUPED с несколькими ковариатами показывает большее снижение дисперсии, но не меньшее смещение. Предиктивный CUPED без ковариат на симуляциях с добавлением эффекта показывает такое же снижение дисперсии эффекта, что и CUPED с ковариатами.

Целевая метрика: gmv магазина

Признаки: gmv магазина, кол-во заказов, gross profit, доля ручных перестроений, oph, конверсия в заказ в приложении и на сайте, кол-во айтемов в заказе, время сборки, доли опозданий, доля отмен 2/4 недели назад

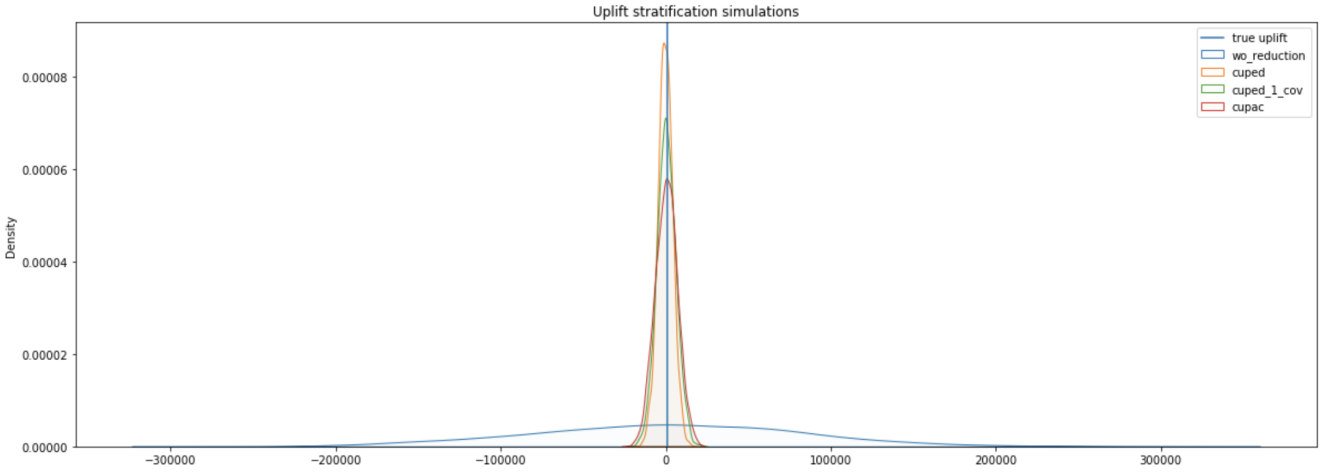
Методы: CUPED с 1 ковариатой, CUPED, CUPAC, мэтчинг контроля к тесту по предсказаниям (match_on_rf), мэтчинг контроля к тесту по ковариатам (matched_control)

Рандомизация без стратификации



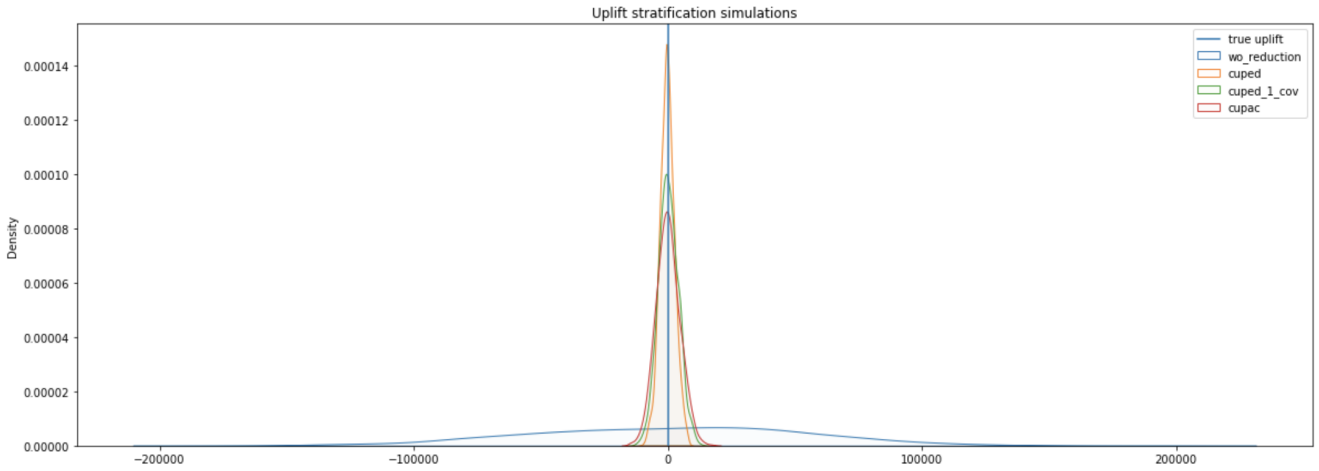
	wo_reduction	cuped	cuped_1_cov	cupac	match_on_rf	matched_control
variance_reduction	1.000000	0.002777	0.004674	0.007597	0.039409	0.033664
bias	0.000000	-44.859616	-591.293338	-469.534147	-7726.965403	-11037.971214
std	82231.354746	4333.729407	5622.036547	7167.194356	16324.357708	15087.660416
mean	-503.237617	-458.378000	88.055722	-33.703470	7223.727787	10534.733597
lvl_05	0.049000	0.081000	0.047000	0.045000	0.118000	0.168000
lvl_01	0.007000	0.021000	0.009000	0.008000	0.023000	0.041000
CI_len	323868.656780	14562.647134	21743.680396	28231.865885	NaN	NaN

Рандомизация без стратификации с использованием части признаков



	wo_reduction	cuped	cuped_1_cov	cupac
variance_reduction	1.000000	0.002886	0.004813	0.006479
bias	0.000000	1091.946707	518.112217	527.969449
std	83466.457064	4483.670887	5790.677935	6718.323332
mean	596.205438	-495.741269	78.093221	68.235989
lvl_05	0.048000	0.091000	0.056000	0.049000
lvl_01	0.009000	0.026000	0.010000	0.007000
CI_len	323494.092716	14583.819298	21726.735379	26351.990820

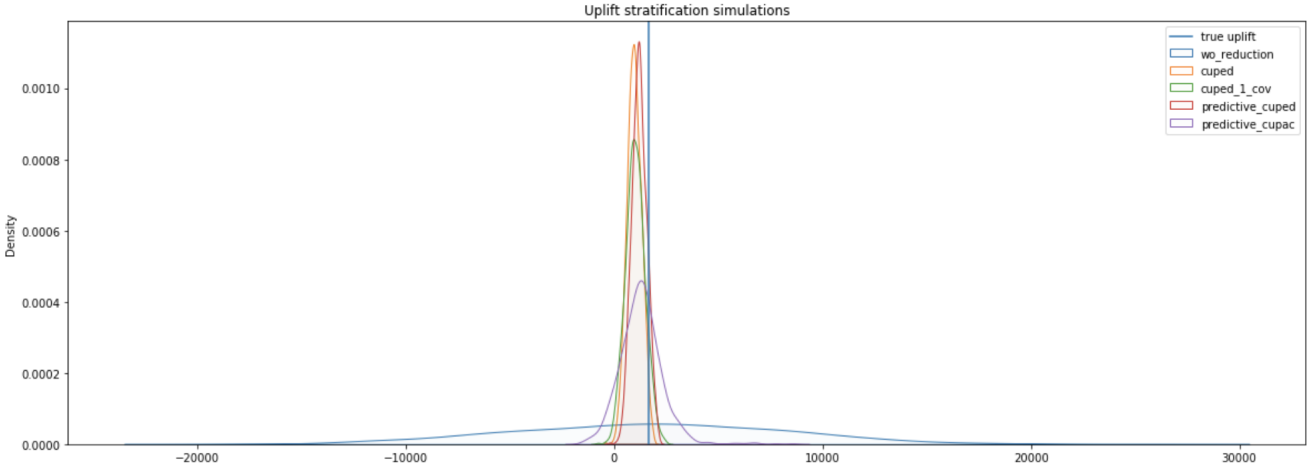
Рандомизация со стратификацией



	wo_reduction	cuped	cuped_1_cov	cupac
variance_reduction	1.000000	0.002697	0.005199	0.007430
bias	0.000000	356.966253	-23.909197	136.453679
std	54705.354223	2841.081482	3944.639328	4715.333645
mean	-33.800256	-390.766509	-9.891059	-170.253935
lvl_05	0.039000	0.072000	0.055000	0.049000
lvl_01	0.003000	0.023000	0.008000	0.010000
CI_len	229944.632863	10218.799012	15392.682927	18731.573186

Рандомизация со стратификацией, с добавлением эффекта. Группировка магазин-день

2% эффект, среднее = 964.8



	wo_reduction	cuped	cuped_1_cov	predictive_cuped	predictive_cupac
variance_reduction	1.000000	0.002652	0.004764	0.002843	0.024666
bias	0.000000	729.287252	685.989724	493.592211	389.703994
std	6707.300123	345.426789	462.952717	357.634556	1053.408657
mean	1669.602056	940.314805	983.612332	1176.009845	1279.898062
lvl_05	0.607000	0.729000	0.561000	0.968000	0.792000
lvl_01	0.494000	0.470000	0.307000	0.915000	0.725000
CI_len	7396.883369	1465.412208	1511.013855	NaN	NaN

Ноутбук:

Страница поддерживается Егор Афанасьев