How Booking.com increases the power of online experiments with CUPED

Simon Jackson Follow
Jan 22, 2018 · 8 min read

Simon Jackson | Data Scientist at Booking.com

Data-supported decisions rule the roost at Booking.com. All product teams are empowered to do controlled experiments (A/B testing) and test any changes they make to the website (Kaufman, Pitchforth, & Vermeer, 2017). Such experiments expose some users to the existing website (base) while others see a new variant, and we statistically test the observed difference.

Booking.com experiment stickers

> *"All Booking.com product teams are empowered to do controlled experiments"*

Small Effects and the Challenge of Statistical Power

As Booking.com continues to optimise its products, new changes produce smaller and smaller effects. Yet we still need to detect these changes via experimentation — even small effects on Booking.com's key metrics can still mean massive revenue differences. **With over 1.5 million room nights reserved on our site each day, even a fraction-of-a-percent increase in conversion can make a big difference to profit**.

> *Detecting small effects can be challenging. Imagine running an e-commerce website with a typical conversion rate of 2%. Using Booking.com's power calculator (open-source code here), you can discover that detecting a relative change of 1% to your conversion rate will require an experiment with over 12 million users.*

This challenge of detecting small effects in experimentation relates to low statistical power. With low power, statistical tests are likely to report a non-significant difference between base and variant, even if the change had a meaningful effect. **An experiment is underpowered when the treatment effect is too small relative to the metric's variance for a given sample size.**

To demonstrate, Figure 1 displays the distribution of properties in base and variant for an important metric in three experiments. Although all three have generated a significant effect, Experiment 2 has the lowest power, and is most likely to return a non-significant result. This is because it has not generated a large enough effect relative to the amount of variance. Experiment 1 has the same variance but has generated a much larger effect, while Experiment 3 has generated the same small effect as experiment 2, but with much smaller variance.

> *Experiment 2 has the lowest power because of the large variance and small effect*

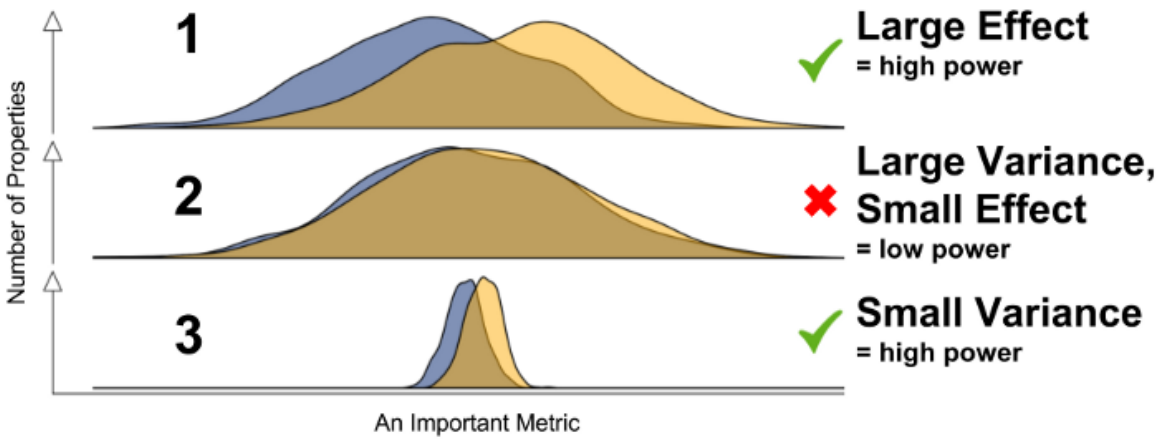

Figure 1. Distributions of properties in base (blue) and variant (yellow) for an important metric in three experiments

CUPED: Controlled-experiment Using Pre-Experiment Data

CUPED is a technique developed by the Experiment Platform team at Microsoft (Deng, Xu, Kohavi, & Walker, 2013) that tries to remove variance in a metric that can be accounted for by pre-experiment information. **Reducing variance helps to increase power to detect small effects.** It's like dealing with the problem in Experiment 2 above, and trying to move to the case of Experiment 3, which has the same effect but with much smaller variance.

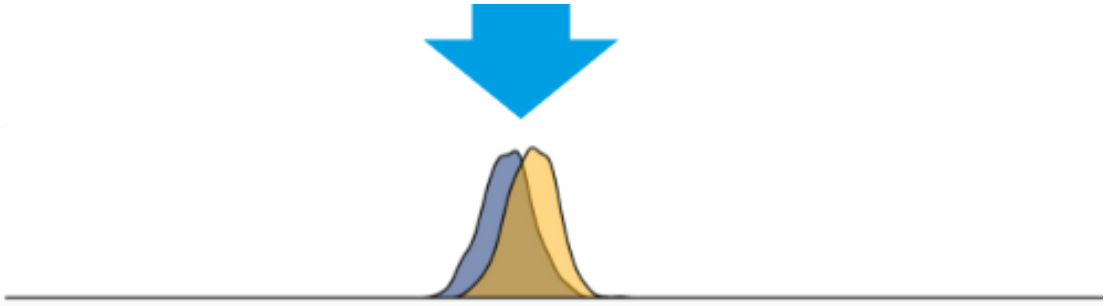

Figure 2. Example of how distribution can change when applying CUPED

> *"CUPED tries to remove variance in a metric that can be accounted for by pre-experiment information"*

The variance that pre-experiment data can explain in a metric is unrelated to any effects of the experiment and can, therefore, be removed. To demonstrate, imagine running an experiment with properties hoping to increase the number of daily bookings they receive. The number of bookings per property per day can range from zero to thousands, so variance in this metric can be enormous. But we often know the average bookings-per-day for each property before the experiment; instead, we can use this knowledge to test whether properties start to receive more, less, or about the same number of bookings-per-day after the experiment compared to before it. Variance in the metric after controlling for pre-experiment information would be considerably smaller than it was originally.

CUPED is a linear-based model for using pre-experiment data to remove explainable variance in a straightforward and scalable way. Microsoft's original paper and a validation paper published by the Netflix experiment team (Xie, & Aurisset, 2016) both provide nice examples of the power gained by using CUPED for online experiments using massive data.

CUPED's Covariate Method

There are stratification and covariate-based methods for implementing CUPED. Stratification involves the use of categorical pre-experiment data like country or browser type. We will focus on the "covariate" method, which uses any continuous metric (bookings per day, number of clicks, time of day, etc.) calculated for each unit (users, cookies, etc.) before they are exposed to your experiment. When possible, our continuous covariate is the same metric as the one we're interested in testing. This is because the achieved variance reduction (and accompanying increase in power) is based on the strength of the correlation between the covariate and the metric.

> *"CUPED's covariate method uses a continuous metric calculated for each unit before they are exposed to your experiment to adjust each unit's score"*

At Booking.com, we implement CUPED to adjust each unit's metric score with the covariate as follows (which slightly differs from how Microsoft and Netflix use CUPED by including `mean(covariate)` to shift the sample mean for reporting, but this has no impact on group differences):

```
CUPED-adjusted metric =
  metric - (covariate - mean(covariate)) x theta
```

Where `mean(covariate)` refers to the covariate mean and `theta` is a constant applied to all units such that:

```
theta = covariance(metric, covariate) / variance(covariate)
```

Consider an experiment testing whether a change in discounts offered to regular users is posited to increase the average bookings-per-week. We have the condition each user is exposed to (base or variant), as well as their bookings-per-week before and after being exposed to the experiment. Let's see what this data might look like, including the CUPED-adjusted metric:

user_id	group	bookings-per-week (metric)	bookings-per-week (pre-experiment covariate)	CUPED-adjusted metric
1	base	2	2	2 - (2 - covariate mean) x theta
2	base	3	2	3 - (2 - covariate mean) x theta
3	variant	5	4	5 - (4 - covariate mean) x theta
4	variant	5	6	5 - (6 - covariate mean) x theta
...	...	...	...	...

Table 1. Example demonstrating calculation of a CUPED-adjusted metric using a pre-experiment covariate

When the metrics are mean-centered, another way to look at what is going on is a scatter plot of our metric (y-axis) against our pre-experiment covariate (x-axis). The slope of the line that best fits these points is defined by `theta` , and the simplified CUPED-adjusted score is the vertical distance from each point to the line.

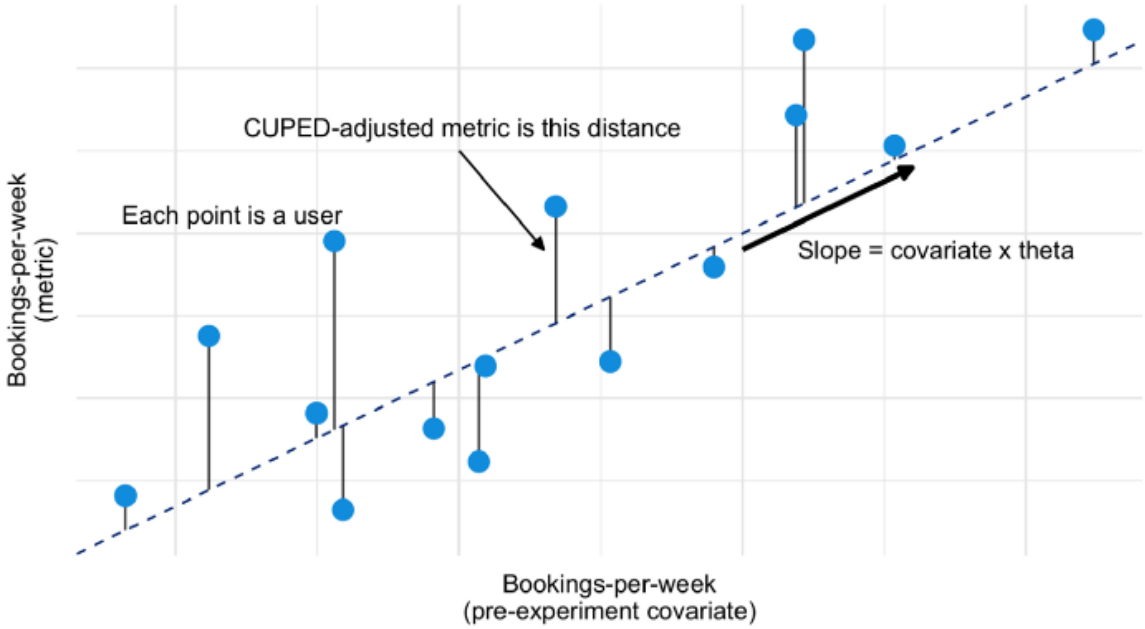

Figure 3. A visual example of how to compute a CUPED-adjusted metric for mean-centered metrics

Statisticians will recognize that `theta` is equivalent to the unstandardized coefficient from an ordinary-least-squares regression of the metric on the pre-experiment covariate.

The practical difference is in the computation of this coefficient and the lack of an intercept term, which is not required for variance reduction.

From this point, you can apply an appropriate statistical test — like a *t*-test — to the CUPED-adjusted metric values, just as you would have tested the original metric values.

Handling Missing Data and Pseudo Code

You might have units with missing data for the pre-experiment covariate, which presents a challenge. For example, we don't have historical data for users who were not logged in, or perhaps for properties who join up with Booking.com while the experiment is running.

We handle missing data in a simple way: **leave metric scores unadjusted for units with missing pre-experiment data**. The reason is quite straightforward. In general, the first best estimate of a missing value is the sample mean. Look at what happens when we substitute missing pre-experiment covariates with the sample mean:

```
if covariate is missing:

 # Substitute covariate with mean
 CUPED–metric = metric – (mean(covariate) – mean(covariate)) x theta

 # Which reduces to...
 CUPED–metric = metric – (0) x theta

 # Which reduces to...
 CUPED–metric = metric
```

Using a best-estimate, the CUPED adjustment is actually no adjustment at all.

Here's the pseudo code explaining how Booking.com uses CUPED:

```
covariate_mean = mean(covariate)
theta = covariance(metric, covariate) / variance(covariate)

for each unit:

  if covariate is missing:
    cuped_metric = metric

  else:
    cuped_metric = metric – (covariate – covariate_mean) x theta
```

Real Example

Here you can examine the results of a Booking.com experiment demonstrating the value of CUPED. In this experiment, we investigated whether a small number of partners would benefit from being able to add room-nights via a new calendar interface. The experiment was ultimately run for 6 weeks to yield a clearly significant result. The daily *p*-value comparing base and variant on a key business metric are shown in Figure 4.

> *For this low-traffic experiment, CUPED-adjustment helped detect a clearly significant result sooner and with a smaller sample than with no adjustment*

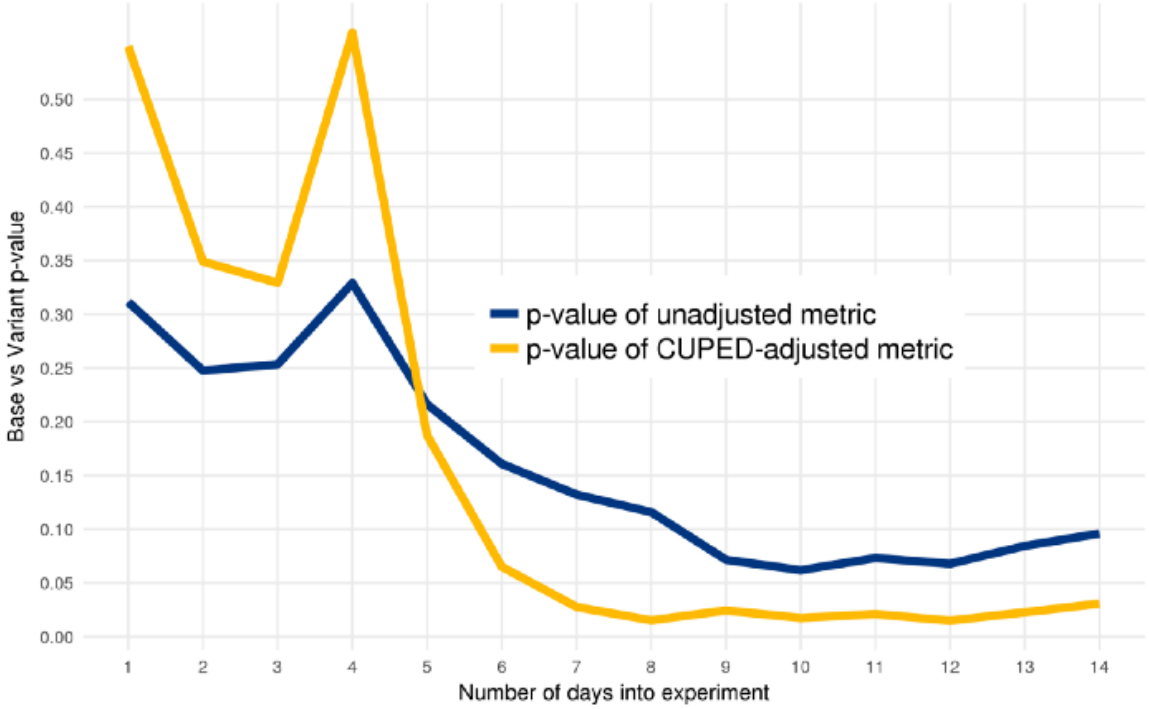

Figure 4. Comparison of experiment results with and without CUPED adjustment for low-traffic experiment

Big-data Implementation

CUPED in Hive

Now we've discussed the method and its value, let's see how to implement CUPED at scale for multiple experiments with Hive. Say you have a table called `exp_data` that looks like this:

exp_id	unit_id	grp	metric	covariate
1	1	base	7	6
1	2	variant	12	NULL
1	3	variant	11	7
2	1	base	8	10
2	2	base	6	5
2	3	variant	9	NULL
...	...	...	...	...

Table 2. Example data for multiple experiments to which CUPED-adjustment can be applied. From left to right, the columns represent an ID for each experiment, ID for each unit (user, property, etc) in each experiment, the condition group (base or variant) that the unit was in, the metric value obtained during the experiment, and the pre-experiment covariate.

For each experiment, we need two constants: the mean of the pre-experiment covariate and the adjustment value, `theta`. Create these in a table as follows:

```sql
CREATE TABLE cuped_constants AS

SELECT        exp_id
            , AVG(covariate) AS covariate_mean
            , COVAR_SAMP(metric, covariate) / VAR_SAMP(covariate) AS theta

FROM        exp_data

GROUP BY    exp_id
```

how_booking_use_cuped-cuped_constants.sql hosted with ❤ by GitHub view raw

Example Hive query for computing constants needed by CUPED adjustment

Join these constants to the unit-level data to do the CUPED adjustment:

```sql
CREATE  TABLE cuped_adjusted AS

SELECT  -- All unit-level, experiment variables
          E.*

        -- CUPED-adjusted metric
        , COALESCE(metric - (covariate - covariate_mean) * theta,
                   metric) AS cuped_metric

FROM    exp_data E

JOIN    cuped_constants C

ON      E.exp_id = C.exp_id
```

how_booking_use_cuped-cuped_adjustment.sql hosted with ❤ by GitHub view raw

Example Hive query for joining CUPED constants and completing CUPED adjustment

We keep the original metric data (via `E.*`) for various reasons (for example, we can compare the original metric to its CUPED-adjusted version later on).

The CUPED adjustment makes use of `COALESCE` (an alternative is `nvl`) to revert to the original metric value whenever the CUPED adjustment produces a `NULL`. This handles units for which the covariate is missing as well as other cases like missing covariate means or thetas, which can occur where your entire sample has no pre-experiment data.

You can now treat `cuped_metric` as a continuous variable for testing. In a typical A/B scenario, for example, we use Hive to compute summary statistics suitable for a two-sample *t*-test: mean, standard deviation, and sample size.

CUPED in Spark

I'd also like to share a code snippet demonstrating how to do CUPED adjustment with Spark, another popular big-data engine. Spark offers an API in numerous languages, and here I use R, thanks to the sparklyr package.

```r
# Packages
library(sparklyr)
library(tidyverse)
```

```r
# Simulate some data
n_experiments <- 3
n_users_per_exp <- 10000
set.seed(171104)

d <- cross_df(list(exp_id = seq(n_experiments),
                   user_id = seq(n_users_per_exp))) %>%
  mutate(group = sample(c("base", "variant"), n(), replace = TRUE),
         covariate = rnorm(n()),
         metric = covariate + rnorm(n(), sd = 0.3) + as.numeric(group == "variant"),
         covariate = if_else(runif(n()) > 0.2, covariate, NA_real_))

print(d, n = 5)
#> # A tibble: 30,000 x 5
#>   exp_id user_id   group  covariate      metric
#>    <int>   <int>   <chr>      <dbl>       <dbl>
#> 1      1       1 variant -0.9815665 -0.07581224
#> 2      2       1 variant         NA -0.13754373
#> 3      3       1 variant -1.6105574 -0.79136708
#> 4      1       2 variant -1.2811401 -0.72244066
#> 5      2       2 variant  0.1324251  1.37553241
#> # ... with 3e+04 more rows

# Connect to Spark
sc <- spark_connect(master = "local")
#> * Using Spark: 2.1.0

# Copy data to spark (and keep connection to it)
d_tbl <- copy_to(sc, d)

# Do CUPED adjustment per experiment in Spark
cuped_results <- d_tbl %>%
  group_by(exp_id) %>%
  mutate(cuped_metric = metric - (covariate - mean(covariate)) * cov(metric, covariate)/var(covariate)) %>
  mutate(cuped_metric = coalesce(cuped_metric, metric))

# Collect results into local memory
d <- collect(cuped_results)

d
#> # A tibble: 30,000 x 6
#> # Groups:   exp_id [3]
#>    exp_id user_id   group   covariate     metric cuped_metric
#>     <int>   <int>   <chr>       <dbl>      <dbl>        <dbl>
#>  1      2       1 variant         NaN -0.1375437  -0.13754373
#>  2      2       2 variant  0.13242509  1.3755324   1.24621238
#>  3      2       3    base  1.96319126  1.6714552  -0.28382946
#>  4      2       4    base  0.44404116  0.3664763  -0.07364255
#>  5      2       5 variant  0.02134668  0.9870520   0.96851903
#>  6      2       6 variant         NaN  1.2720197   1.27201966
#>  7      2       7    base -0.35044885 -0.2419659   0.11032162
#>  8      2       8 variant         NaN -0.8750044  -0.87500437
#>  9      2       9 variant  0.81343645  1.3115542   0.50300887
#> 10      2      10 variant -0.18234841  0.6818743   0.86650225
#> # ... with 29,990 more rows

# Disconnect form Spark
spark_disconnect(sc)
```

how_booking_use_cuped-sparklyr_cuped.md hosted with ❤ by GitHub view raw

Reproducible example for doing CUPED adjustment in Spark via R

The CUPED adjustment is done where `cuped_results` is being created.

Want to learn more?

To learn more, keep in touch with @drsimonj on Twitter, connect on LinkedIn, and be sure to visit Planet Booking if you're interested in joining our growing team.

Finally, I'd like to extend my thanks to Nishikant Dhanuka, Steven Baguley, Kristofer Barber, Pavel Levin, and Themis Mavridis for their invaluable comments on this post.

Thanks to Nishikant Dhanuka, Pavel Levin, Steven Baguley, Kristofer Barber, and Booking.com Data Science.