# DoorDash

DoorDash · Follow

12 min read · Feb 14, 2018



David Kastelman, Data Scientist & Raghav Ramesh,
Machine Learning Engineer

*To A/B or not to A/B, that is the question*

**Overview**

On the Dispatch team at DoorDash, we use simulation, empirical observation, and experimentation to make progress towards our goals; however, given the systemic nature of many of our products, simple A/B tests are often ineffective due to network effects. To be able to experiment in the face of network effects, we use a technique known as switchback testing, where we switch back and forth between treatment and control in particular regions over time. This approach resembles A/B tests in many ways, but requires certain adjustments to the analysis.

**Dispatch at DoorDash**

The core responsibility of the Dispatch system at DoorDash is to power fulfillment of every delivery in Doordash's three-sided marketplace of Consumers, Dashers and Merchants. To effectively achieve this, we focus on

1. the core assignment problem — deciding which dasher is the best suited to fulfill a delivery, in an efficient, robust and scalable way.
2. machine learning algorithms to predict the numerous timepoints in the life of a delivery — "when can this order reach the consumer", "when will this order order be ready for pickup?", etc.

3. algorithms to decide how and when to group multiple deliveries headed in similar directions at similar times

4. how to leverage various marketplace shaping strategies to balance supply and demand, including pricing changes

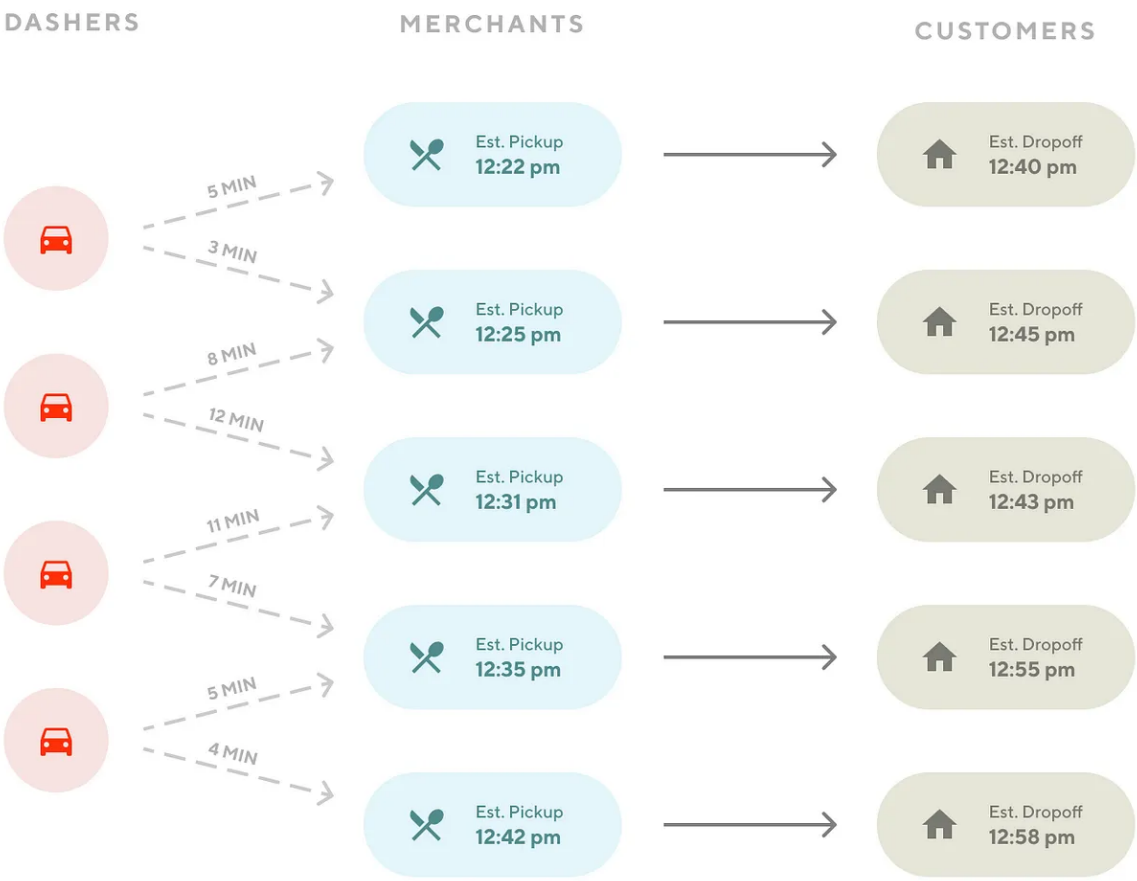



Illustration of the core assignment problem

As we continuously iterate on these problems, we rely heavily on both simulation and experimentation to best serve our audiences on all three sides of the marketplace. Offline simulation is helpful for checking assignment algorithms, and offline evaluation and A/B testing helps us with iteration on prediction algorithms. However, it is no surprise that in dealing with marketplace problems online, *network effects* often make traditional A/B testing ineffectual. We'll explain that with an example of SOS pricing below.

**Marketplace Experimentation**

SOS pricing is a demand shaping strategy used when we have too few dashers compared to the incoming delivery volume. In these scenarios, in order to avoid overwhelming the Dasher fleet with an intractable number of deliveries, thereby increasing Consumer delivery times, we enable SOS pricing, which increases delivery fee. With this increase, demand gets throttled and shifted to later times; meanwhile, dashers are motivated to dash more. When we introduce (or later, modify) the SOS pricing algorithm, we would like to experiment to understand the impact on customer retention and delivery times. If we were to test changes as an A/B experiment on Consumers, 50% of Consumers would see SOS pricing and 50% wouldn't. In this case, the first set of Consumers get only half of the benefit of supply equilibration (by extension, reduced impact on Consumer delivery times), while the other half get the partial benefit without any extra pay, which makes our learning incomplete. The problem here is the network effect — both sets of Consumers share the same Dasher fleet — so adding a Consumer to the treatment group also affects the experience of Consumers in the control group and we can not establish independence between the two groups.

One way to get around network effects is to introduce the change and observe the impact before and after. For instance, we can compare Consumer delivery times before and after the introduction of SOS pricing. The main problem with this approach is the famous maxim that correlation doesn't imply causation. Without a randomized experiment, we cannot be sure if the results we are seeing after our change are really because of that change, or just coincide with other things changing in the system. DoorDash is a dynamic company with a lot changing every day, so relying on a pre-post comparison is a risky proposition. At a minimum, relying on pre-post analyses would cause a dispatch bottleneck where we could only change one big thing at a time. Even this strategy, however, is insufficient for interpreting correlations as causal because occurrences outside dispatch's control like consumer promotions and weather have a big impact on our normal dispatch metrics and might be the true cause of a pre-post observed difference. To take the most-cited line from the oft-cited Airbnb medium post on A/B testing, "the outside world often has a much larger effect on metrics than product changes do."

### Switchback Experimentation

Due to the limitations of A/B tests and the insufficiency of pre-post comparisons, the Dispatch team recently decided to switch to a new analytic framework for much of its experimentation — 'switchback testing.' Fun fact: switchback testing was

originally employed in an agricultural context, specifically for cow lactation experiments. *MOOOOO.*

In switchback testing, the core concept is that we switch back and forth between control and treatment algorithms in a certain region at alternating time periods. For example, in the SOS pricing example, we switch back and forth every 30 minutes between having SOS pricing and not having SOS pricing. We then compare the customer experience and marketplace efficiency between the control time bucket and treatment time bucket metrics corresponding to the decisions made by the algorithm during the two periods.

### Implementing Switchback Experiments

In implementing switchback experiments, we include two extra pieces of logic:

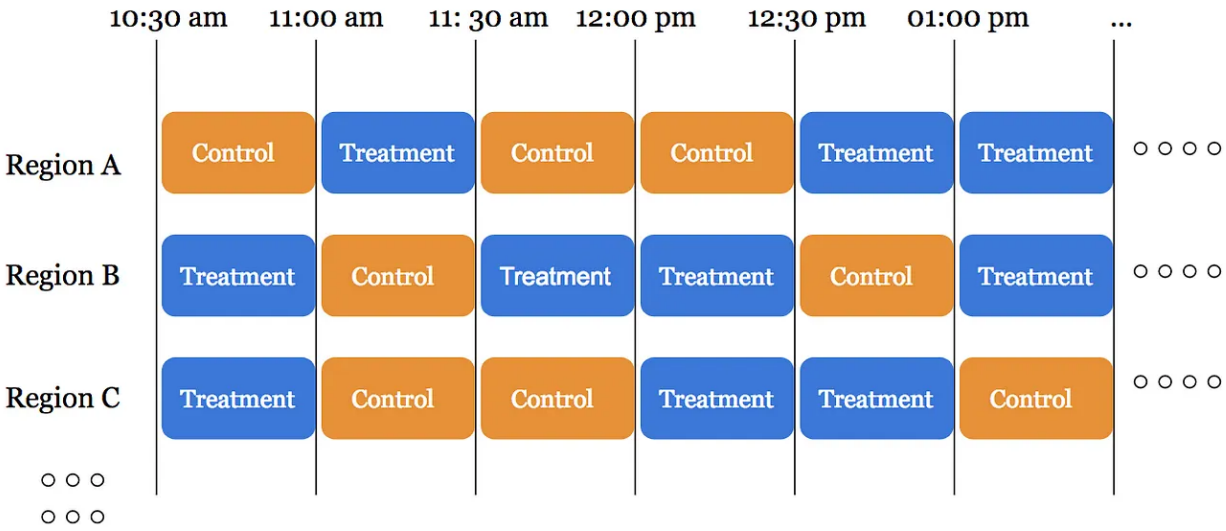



Illustration of time-market unit randomization

1. We randomize the variant used for each time window, rather than simply randomizing the initial variant and flipping back and forth deterministically. With this approach, each time unit is a randomized experimental unit.

2. We further split across different geographical units (regions) and randomize them independently, so region A could use the current algorithm for one window and the new algorithm for the next, while region B could use the new algorithm for the first window and the old algorithm for the next. This in a sense actually creates a series of 'time-region units.'

In many ways, switchback testing is exactly like A/B testing, but instead of randomizing based on something like deliveries, we randomize based on time-

region units.

The rest of the section explains the design of the switchback service we use at DoorDash.

The switchback service is responsible for three functions

1. Storing metadata on all experiments (Metadata)
2. Choosing the algorithm variant to use at certain point of time for any experiment (Bucketing)
3. Handling tracking to enable metrics calculation and analysis (Tracking)

The most important metadata of an experiment is the “switchback window”, i.e., the duration of time for which we persist a variant before switching. Other metadata include names and relative sizes of the bucket variants. Internally, the switchback service stores the following state — 1. The start of the current time window, 2. A map with region id as the key and the current bucket name as its value and 3. A unique identifier for the experiment unit (time + region)

To achieve bucketing and tracking, the switchback service exposes two endpoints

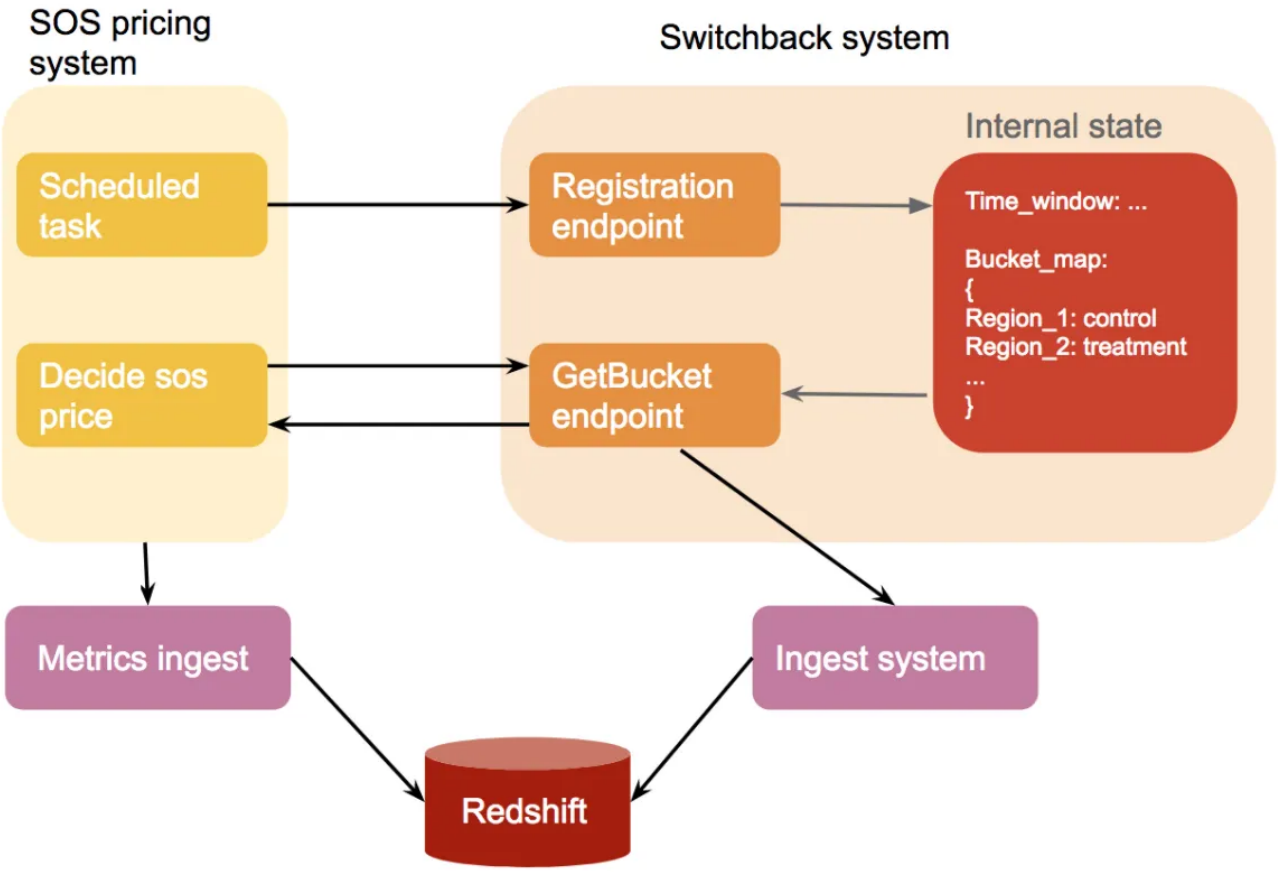
Registration

- Based on the time passed by the registration endpoint, the switchback service checks and updates its internal state e.g. check if it is time to update the buckets and if so, flip a (multi-sided) coin for every region and determine the bucket for the new time window

GetBucket

- Used by the experimenter system to ask for the current bucket (variant) for an experiment
- the switchback service fetches the bucket value from its state and sends a tracking event (which includes experiment metadata and unit identifier) to the ingest pipeline.

A set of ETL jobs that combine the above tracking events and our metrics tables enable analysis of the experiment results.

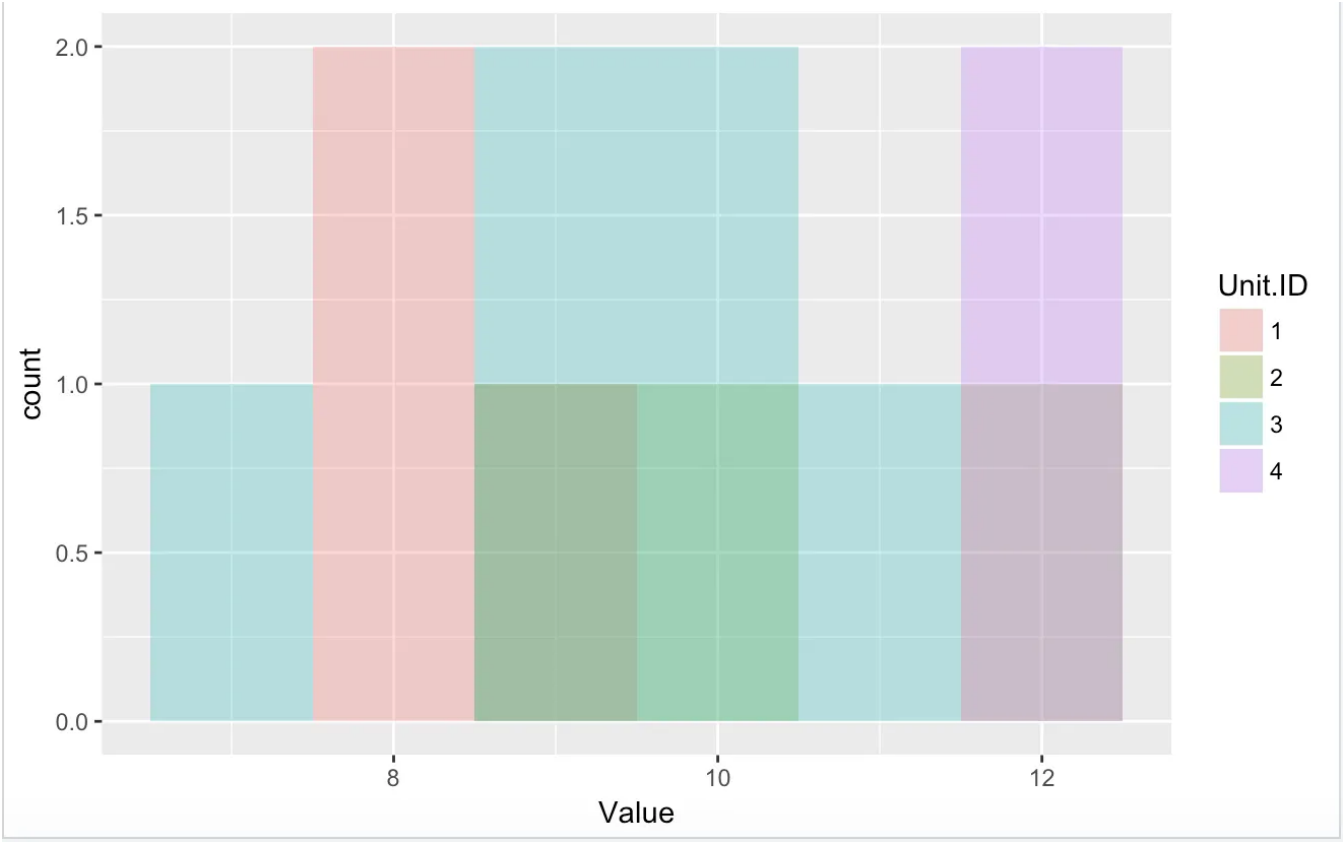



*Schematic of the switchback system*

### Analyzing Switchback Experiments

In order to use something like a t-test (which is often used to analyze the results of A/B tests) for our switchback experiments, we analyze based on the same units we're randomizing on: namely time-region units. This means we're conducting our statistical tests on the **average values** of control and treatment **time-region units,** rather than the **individual values** of control and treatment **deliveries.** Therefore, the average value for the treatment and control groups is a simple average of all time-region units rather than a weighted average of those values (weighted by deliveries). This is illustrated with a simple example of 14 deliveries below. While the simple and weighted average of values usually converge over time, if they do not, it is an indication that your intervention has a different impact on units with a lot of deliveries than it does on units with fewer deliveries. In the future, we plan to handle these divergent situations by using multi-level modeling (MLM, also known as hierarchical modeling). MLM also has the advantage of likely being able to reduce the margin of error of our statistical tests.

**Sample of 14 deliveries (numeric table in appendix) with illustration of weighted vs. simple average**




**Weighted Average**

| Group | Average | Count of Delivs |
|---|---|---|
| control | 9 | 9 |
| treatment | 11 | 5 |

**Simple Average**

| Unit ID | Average Value | Count of Delivs | Group |
|---|---|---|---|
| 1 | 8.333333333 | 3 | Control |
| 2 | 10.33333333 | 3 | Treatment |
| 3 | 9.333333333 | 6 | Control |
| 4 | 12 | 2 | Treatment |

| Group | Average Value | Count of Units |
|---|---|---|
| Control | 8.833333333 | 2 |
| Treatment | 11.16666667 | 2 |

Moreover, traditional A/B testing has an assumption of independent units, which means that if we know something about the behavior of one unit, it does not tell us anything about the behavior of the second. This assumption is clearly violated in the case of time-region units as the performance of one time-region is highly correlated and can influence the performance of the next. To give a more concrete example, the average time it takes to complete deliveries in one area in one 10-minute chunk of time is related and highly correlated to the average time it takes to complete deliveries in the same area in the next 10-minute chunk of time — much more so than in the case of a single delivery and the next delivery assigned. To account for this lack of independence in our statistical tests, we use an approach robust to a lack of independence called a sandwich estimator of variance (R-snippet of the code we use included in appendix). While strictly speaking this approach (or something similar) is required to avoid violating core assumptions, thus far when running A/A tests (a dummy experiment where there's no actual difference between "treatment" and "control") we've found the sandwich estimator's effect on our variance calculations has been <10%.

### Setting up Experiments and Doing Power Calculations

In creating our time-region units and determining how granular the geographic units should be and how quickly to switch back and forth, we really have to consider two factors: bias and margin of error.

Bias occurs when your randomization of time-region units is compromised, and the sorts of deliveries that are in treatment and control group are not the same on average. For instance, we might test a certain algorithm which is reluctant to assign deliveries with longer distances between stores and customers. As a result, this treatment algorithm may be observed to have a faster average time to completion even if it is not a better algorithm simply because it is cherry-picking short deliveries and leaving the control group to complete longer deliveries. Bias is more likely to occur if your regions get too small or you switch back too frequently; when we switchback less frequently, it forces the treatment group to complete many more of the longer deliveries and rely less on the control group to clean up its mess. More on health checks to guard against bias and check successful randomization in the footnote.

Margin of error is how much uncertainty exists in our estimate of the impact of an intervention. There are two main factors that influence the uncertainty in our estimate: the natural variation in the data we are observing and the number of units in our dataset. Without getting too technical (refer to sampling distributions for more information), the margin of error in our estimates is proportional to the natural variation in our sample divided by the square root of the number of samples. As time-region buckets get more granular, natural variation in a metric tends to go up, however more granular time-region buckets provide more units in our dataset which drives down uncertainty.

To understand the interplay between natural variation and number of samples, we ran a series of long-term A/A tests. These tests allowed us to look at how margin of error differed for the average value of a certain metric in a time-region unit as we switched back and forth more quickly or more slowly. The results for one metric are summarized in the table below (note, given the small impact of the sandwich estimator in most situations, it is ignored in this table):

| Time window | Sample stddev | % of time units (indexed to 60min window) | Total margin of error (indexed to 60min window) |
|---|---|---|---|
| 20 Minutes | 7.94 | 300% | 0.68 |
| 30 Minutes | 7.47 | 200% | 0.79 |
| 60 Minutes | 6.72 | 100% | 1 |
| 1 day | 3.56 | 4.17% | 2.59 |
| 1 week | 3.12 | 0.60% | 6.02 |

We were able to do a similar sort of analysis using different levels of geographic granularity.

Summarizing the following above considerations about bias and margin of error: we do not want time-region units that are too small or we risk introducing bias and failing one of our randomization health checks; on the other hand, we do not want time-region units that are too large or we risk having large margin of errors and taking too long to get our learnings. Based on these analyses, we typically run our tests with 30 minute switchback periods using certain geographic divisions roughly at the city-level.

### Validation of our Switchback System

An extra verification step we are interested in is “are the results from a switchback experiment reflective of reality?”. Concretely, if we see an X% increase in a metric in an experiment and we completely move over to the new algorithm, will we actually observe the gains in the metric in the real world? This is particularly important from a business perspective and for making reliable product decisions based on experimental results.

To do this, we observe the changes in metrics prior to and after the launch of the new algorithm. As described earlier, observing changes in a pre/post scenario is challenging for multiple reasons. However, over large time windows after product launches, we can reliably observe the impact on the metrics time series. So, what we look for here is just directional confirmation of the results we observe in the experiment.

### Conclusion

As DoorDash tackled this problem, we were able to find some helpful resources like Lyft's 3-part discussion of experimentation in a ridesharing marketplace. We felt there were still some questions left to answer and believe we've been able to make some additional observations particularly on the analysis and implementation details, and we hope our observations may help you if you're considering similar issues. That being said, we fully admit we're just getting started on unpacking exciting challenges like the ones discussed above. *Special thanks to Yixin Tang and Viraj Bindra for their help in publishing this post.*

**See something we did wrong or could do better, please let us know! And if you find these problems interesting, come work with us!**

### Appendix

**R-Snippet for Analysis with Sandwich Variance Estimator**

```
library(sandwich)
sb_data <- read.csv(file.choose())
sb <- sb_data

fit <- lm(dat~result, data=sb_data)
sandwich_se <- diag(vcovHC(fit, type = "HC"))^0.5
sandwich_se
coef(fit)-1.96*sandwich_se
coef(fit)+1.96*sandwich_se
z_stat <- coef(fit)/sandwich_se
p_values <- pchisq(z_stat^2, 1, lower.tail=FALSE)
P_values
```

**Sample Data**

| Delivery ID | Value | Unit ID | Experiment Group |
|---|---|---|---|
| 1 | 8 | 1 | control |
| 2 | 9 | 1 | control |
| 3 | 8 | 1 | control |
| 4 | 10 | 2 | treatment |
| 5 | 9 | 2 | treatment |
| 6 | 12 | 2 | treatment |
| 7 | 9 | 3 | control |
| 8 | 9 | 3 | control |
| 9 | 10 | 3 | control |
| 10 | 11 | 3 | control |
| 11 | 7 | 3 | control |
| 12 | 10 | 3 | control |
| 13 | 12 | 4 | treatment |
| 14 | 12 | 4 | treatment |

Numeric Data for Sample of 14 deliveries used in example in article above

[1] To guard against bias and ensure 'successful randomization', you can check to make sure that factors that are unaffected by your intervention have equal values between the treatment and control group: in our example, a good value to check would be the distance between restaurant and customer. At DoorDash, however, we often find that simply checking to make sure the expected proportion of deliveries are considered by treatment runs and control runs sufficiently guards against most instances of bias. To return to our example of an algorithm that is hesitant to assign long-distance deliveries, this would result in unequal proportion of deliveries being assigned the control group rather than the treatment group and this deviation would indicate the existence of bias.

[2] For example, average delivery times in a specific 5-minute window at a city level varies more widely than can average delivery times across all of DoorDash's regions in an hour.

[3] However, if we notice any bias, we shut down the experiment and start over with coarser time and/or geographic units.

Data Science | Engineering | Logistics | Ab Testing | Statistics