Low Inventory State: Identifying Under-Served Queries for Airbnb Search

Toma Gulea
toma.gulea@airbnb.com
Airbnb Inc.
San Francisco, California, USA

Bradley C. Turnbull
bradley.turnbull@airbnb.com
Airbnb Inc.
San Francisco, California, USA

ABSTRACT

Airbnb offers guests the opportunity to book unique as well as more traditional accommodations. Guests primarily explore the available inventory for their trip by executing searches and leveraging a variety of filters to express their preferences. Sometimes a guest will encounter very few results returned for their search which can lead to a poor user experience and eventual abandonment of the platform without booking. From a user experience and business perspective, it is therefore valuable to identify such search queries that yield insufficient inventory and lead to missed conversion opportunities. The longevity and complexity of user search sessions on Airbnb, however, pose an additional challenge to this problem. A user's intent and persistence confounds with our ability to understand the connection between the number of search results returned, and a user's ultimate booking outcome. We overcome this hurdle by employing a causal inference approach paired with predictive modeling. In this paper, we present a causal framework and methodology to identify searches where an insufficient number of results returned is preventing booking conversion. Our ability to identify these searches has applications across analytical insights, experiment analyses, real-time product interventions and supply management. We demonstrate the efficacy of our approach via simulated data experiments and real user search queries.

CCS CONCEPTS

• **Information systems** → *Evaluation of retrieval results*; **Query log analysis**; • **Applied computing** → *E-commerce infrastructure.*

KEYWORDS

Causal inference, Search results, under-served queries, Propensity score matching, Causal graph, E-commerce search

ACM Reference Format:
Toma Gulea and Bradley C. Turnbull . 2023. Low Inventory State: Identifying Under-Served Queries for Airbnb Search. In *Proceedings of 2nd Workshop on End-End Customer Journey Optimization (CJ'23).* , 9 pages.

1 INTRODUCTION

Airbnb is a two-sided rental marketplace where guests can book both traditional and unique accommodations. A guest looking for accommodation generally starts with a search query, likely specifying a location, dates, and guest count. As a guest explores the available accommodation listings they may refine their search, applying filters to express specific preferences, such as price maximum or access to a pool. Once a guest finds a listing that they like, they can checkout and book it.

As guests refine their search, they may encounter very few search results. This may be because they have applied a lot of filters to their search, or there are very few remaining listings for their dates. States of low inventory can be a frustrating experience for the searcher, and can ultimately lead to the searcher abandoning the platform without making a booking. Therefore, an important part of building a great search experience for our guests is identifying when a searcher is facing an insufficient amount of inventory, which is causing a bad user experience and a missed booking opportunity.

The literature on low search results [7, 8], generally focus on methodologies for rewriting such queries. However, we were unable to find any studies addressing the determination of whether the number of results is "too low". Queries returning null search results are generally used as the candidates for rewriting. A singular static threshold can also be used (e.g. 20 results), but this approach has limitations at Airbnb due to the diverse nature of our guests' searches.

For instance, a last minute search for accommodations in a popular coastal city may yield 30 results during the peak season (without any filtering). On the other hand, a search for lodging in a specific neighborhood with a strict price preference may yield 10 results. The 30 results returned for the first broad unfiltered search query is likely insufficient and may prevent the guest from booking, while the 10 results returned for the precise search are typical given the specificity of the search and sufficient to result in a booking.

This paper presents a novel methodology to identify searches where there is an insufficient number of results returned, preventing booking conversion. We use a causal inference approach, as well as predictive modeling to tackle the problem. In summary, the paper makes the following contributions:

(1) Presenting a system describing the causal relationship between booking conversion and number of search results,
(2) An approach to estimate the true effect of number of search results in the presence of latent variables,
(3) An application of combining predictive modeling and causal inference to understand user outcomes in search.

2 PROBLEM DEFINITION

We are interested in identifying search queries for which the amount of inventory returned is preventing booking conversion. We call such queries, a "Low Inventory State" (LIS).

Another way to define such state is a search for which an additional result would bring a significant increase to the likelihood of booking conversion for the guest. This definition implies a binary outcome: a search is either a *"LIS"* or not, but this is only for practical purposes and simplicity of use for consumer of such a metric. Later on, we will briefly discuss how we establish a threshold to satisfy a binary output. In summary, the core of the problem is to determine the incremental effect of an additional result, for a given

search query, on the probability of booking (conversion):

$$BookIncr_{qry} = \frac{P(B = 1|R + 1)}{P(B = 1|R)} \quad (1)$$

where B denotes if the search query eventually resulted in a booking for the user ($Conversion = (B = 1)$ if the query is linked to a booking, 0 otherwise) and R denotes the number of results for the query (i.e. the amount of inventory).

3 HIDDEN EFFECT OF THE NUMBER OF RESULTS ON CONVERSION

If we draw the relationship between number of results and conversion, we observe a negative relationship, i.e. the more results the worse the conversion rate. Figure [1] depicts this relationship on a random sample of user queries.

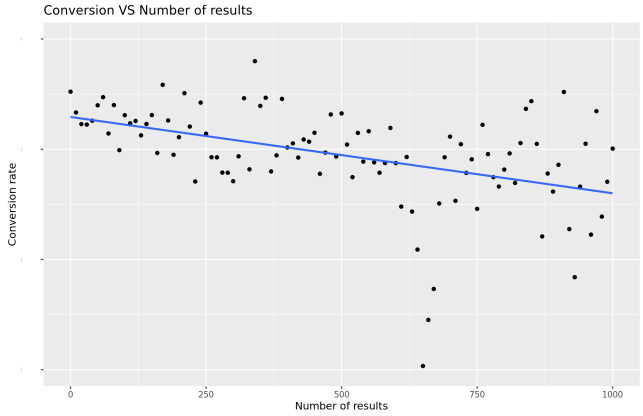

Figure 1: Average conversion rate per number of results

This is counter-intuitive as we would expect that more inventory would lead to higher probability of conversion. What we are actually observing is the effect of a user's intent to book. Users on Airbnb can either be in an early exploration phase, or ready to make the booking for their next trip. We refer to this as the "intent to book" for a search query. The assumption is that users with high intent to book tend to input more criteria and therefore limit the inventory returned. While a user can just be "exploring" with broad searches, a user with higher intent is generally more precise with location, budget and amenities. The result of this phenomenon is that a low inventory is indicative of a high intent and therefore positively correlates with conversion. In order to encode these assumptions on the casual relationships, we will use causal graphs to help us derive an identification [4], [6] of the impact of number of results on conversion. Figure [2] illustrates this problem, where Intent is a latent (unobserved) variable.

Upon initial inspection, one possible way to approach the problem would be to model the likelihood of booking based on the number of search results. For example, we could fit a logistic regression of B on the number of results. However, this approach would pose an identification issue, as the user's intent drives the number of search results down while positively influencing booking conversion. Such an approach would not allow to identify what would happen if we added more results to a query.

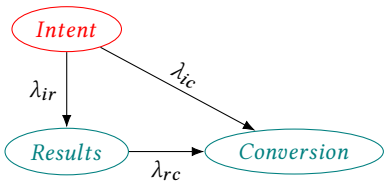

Figure 2: Unidentified model with latent variable

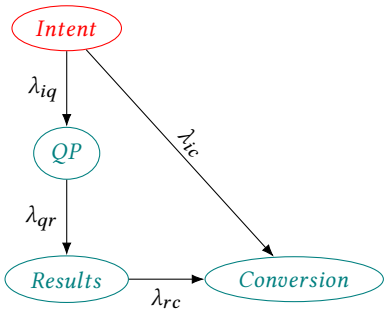

Figure 3: Unidentified model with latent variable 2

4 CLOSING THE BACKDOOR PATH OF INTENT

From Figure [2], we are interested in identifying λ_{rc}: the causal effect of the number of results on conversion. If there is no Intent to book ($I = 0$), the probability to book is null:

- $P(B = 1|I = 0) = 0$
- $P(B = 1|R) = P(B = 1|R, I = 1) \times P(I = 1)$.

Therefore, eq. 1 can be written incorporating Intent I as:

$$BookIncr_{qry} = \frac{P(B = 1|R + 1, I = 1)}{P(B = 1|R, I = 1)} \quad (2)$$

The mechanism by which Intent causes the number of results is indirect; it happens through the parameters of the query QP (e.g. map zoom, price filter, amenities requirements etc.) as depicted in Figure [3].

QP blocks the backdoor path of Intent as the effect of Intent on number of results is unconfounded by the query parameters [5]. QP is observable, we therefore condition Conversion on the query parameters QP and rewrite equation 2 as follows:

$$BookIncr_{qry} = \frac{P(B = 1|R + 1, QP, I = 1)}{P(B = 1|R, QP, I = 1)}. \quad (3)$$

Using Bayes theorem we obtain:

$$BookIncr_{qry} = \frac{P(R + 1|B = 1, QP, I = 1) \times P(R|QP, I = 1)}{P(R|B = 1, QP, I = 1) \times P(R + 1|QP, I = 1)} \quad (4)$$

For additional steps in obtaining the result, please refer to Appendix A.

According to the causal relationships hypothesized in Figure [3]:

- $Intent \perp R|QP$,
- $Intent \perp R|(QP, B)$.

Therefore we can rewrite

$$BookIncr_{qry} = \frac{P(R + 1|B = 1, QP) \times P(R|QP)}{P(R|B = 1, QP) \times P(R + 1|QP)} \quad (5)$$

With this equation, we have closed the backdoor path of Intent.

As a result, computing the incremental impact of number of results only requires us to estimate the law behind $P(R)$ and $P(R|B = 1)$:

- $P(R|QP)$: Number of results given the query parameters as features,
- $P(R|B = 1, QP)$: Number of results for users who made a booking given the query parameters.

The results count variable, R, can reasonably be assumed to follow a Poisson distribution. However, it is important to re-evaluate this assumption based on the specific application. Alternative approaches such as a negative binomial regression or zero-inflated models can be considered as viable options [1]. Here we choose to estimate $P(R|QP)$ and $P(R|B = 1, QP)$ using a Poisson regression, following the Poisson distribution assumption.

One might observe that our solution bears a resemblance to a propensity score matching approach, as demonstrated by matching queries with the same number of expected results but different actual results, then comparing their conversion rates. In Appendix B, we provide additional details on how our solution can be framed using propensity score matching.

5 COMPUTING THE LIS METRIC

In Section 4, we established that to compute our LIS metric we need to:

- train a model estimating the number of results given the query parameters,
- train a model estimating the number of results for users who made a booking given the query parameters,
- derive the incremental value of an additional search result for each query given the two estimations and actual number of results.

5.1 Poisson modeling

We seek to estimate the number of results returned by a search query using a Poisson regression, where the features are the query parameters denoted by QP. To account for potential non-linearities and interactions between the query parameters, we used a boosted regression tree (BRT) model, which combines multiple decision trees via an additive, gradient-boosting algorithm [3].

Specifically, we fit two BRT models using the number of results as the response variable and the query parameters QP as the input variables. One model is trained on all searches and the second model is trained only on searches of guests who made a subsequent booking. The resulting models allow us to estimate the expected number of search results, λ, and the expected number of results when a booking was made, $\lambda_{B=1}$.

For illustrative purposes, we provide below a non exhaustive list of features from QP:

- query type (city, address, point of interest etc.)
- Map radius
- Filters used (e.g. price, amenity etc.)
- Number of guests
- Number of bedrooms
- Number of nights
- Lead time (days before check-in)

5.2 Deriving the LIS metric

Once the number of search results has been estimated, the incremental impact of an additional result on conversion, following equation 4, is derived as follows:

$$\text{IncrBook}(\lambda_{B=1}, \lambda, R) = \frac{P(\lambda_{B=1}, \mu = R+1) \times P(\lambda, \mu = R)}{P(\lambda_{B=1}, \mu = R) \times P(\lambda, \mu = R+1)}, \quad (6)$$

where,

- $P(k, \mu)$ denotes the probability mass function (PMF) of the Poisson distribution of paramter μ,
- λ is the estimated number of results given QP,
- $\lambda_{B=1}$ is the estimated number of results for bookers given QP,
- R is the observed number of results for the search query.

The ratio value $\text{Incr}(\lambda_{B=1}, \lambda, R)$ can be interpreted as the increase in likelihood of conversion for one additional result to the search query. For example, an estimation value of 1.1 implies that one additional result renders a 10% increase in conversion.

In practice this ratio metric can be used directly as a continuous measure or bucketized (e.g. LIS searches and none LIS searches). The bucketization can be done in various manners depending on the application. For the binary case, examples include:

- setting a classification threshold based on share of searches classified as LIS or
- identifying all searches with a specified gain in conversion or more from an additional result.

Python code to compute $\text{Incr}(\lambda_{B=1}, \lambda, R)$ is available in Appendix C.

6 ASSESSMENT ON SIMULATED DATA

We leverage a simulated data experiment to assess the efficacy of our model, ensuring it accurately estimates the true effect, which in practice cannot be directly observed due to the presence of hidden latent variables [10].

6.1 Simulation process

We design a simulation that adheres to the causal relationships presented in Figure [3]. The detailed simulation steps to generate the data are available in the appendix C with a brief description provided in this section. We generate simulated data as follows:

(1) randomly generate a user's intention to book: 0 or 1 (intent),
(2) if intent is 1 then assign a higher probability that filtering (0 or 1) is used,
(3) generate the number of results from a Poisson distribution with the rate parameter influenced by filtering (if filtering is used, we draw less results).
(4) Finally, randomly generate a binary booking outcome:
    (a) 0, if intent is 0,
    (b) $binom(p)$, with p scaling with the number of results if intent is 1.
(5) The true incremental impact (not observable in the real world context) $\frac{P(B=1|R+1)}{P(B=1|R)}$ is also computed for each simulated data point.

6.2 Benchmark

We benchmark the performance of our methodology against a non-causal approach of directly modeling $P(B|R, QP)$ and extracting the marginal impact of R on $P(B)$, with QP serving as a control. More specifically, we fit a logistic regression of B on R and QP. We refer to this comparative approach as *"direct booking method"*, and the causal inference methodology presented in this paper as *"LIS"*. In practice and in the literature it is common to evaluate the impact of the number of search results using a fixed threshold, e.g. "does the search have more than 10 results returned?". We, however, do not benchmark against this method as it is possible to generate simulated data such that a fixed threshold would perform worse than random.

6.3 Simulation results

6.3.1 Simulation for formula validation. Our first simulation serves as a validation of the formula. Simulation (1) respects the strict assumptions that results are drawn from a Poisson distribution and conversion is linearly derived from the number of results. We can observe in Figure [4] that the estimation of the odds ratio produced by the "LIS" method (conditional on user intent) is exactly what is observed in the data. Conversely, the estimation from the "direct booking method" falls short, underestimating the decline in the odd ratio as the number of results increases.

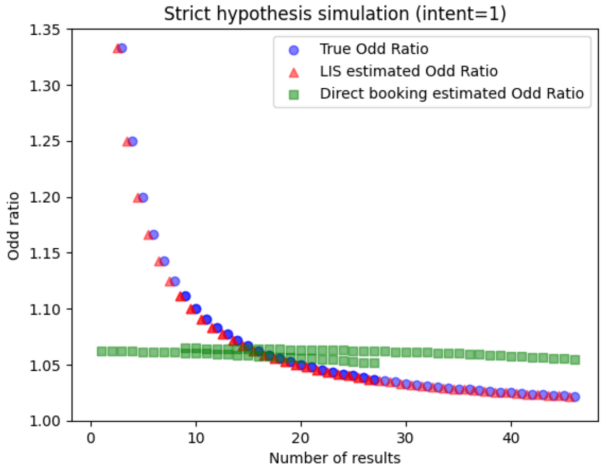

Figure 4: simulation (1): True Odd ratio VS LIS estimated

True odd ratio and LIS estimation exhibit complete overlap. The direct booking estimation yields two lines (depending on filtering status).

6.3.2 Simulation against benchmark. Simulation (2) is designed to compare the performance of our method vs the "direct booking method" under the scenario when data is generated in a manner that favors the benchmark. Data is generated by drawing results from over-dispersed data (violating the Poisson distribution assumption), and the impact of results on conversion is computed using a linear logistic transform, the same specification and assumptions as the direct booking method.

We focus on comparing each method's ability to estimate the ground truth odds ratio for two key classes of queries: (1) user queries with filtering applied, and (2) queries with no filtering. Each class of queries should have varying levels of impact from a lack of search results, specifically we should observe:

(1) For queries with filtering, the impact (odds ratio) of fewer search results should be larger. Users who filter are more likely to be high intent, therefore the number of search results will presumably impact their booking outcome more.
(2) For queries without filtering, the impact (odds ratio) should be less, as users without filtering are lower intent and therefore their chances of booking are low regardless of the number of search results.
(3) For both classes of queries, the impact on conversion should exponentially decay with the number of search results.

It is most important, and the main evaluation criteria, that each method can properly capture the above trends. The final application of any method to identify low inventory searches will require us to set some classification threshold. Therefore, as long as the estimations follow the proper trends with respect to query types and number of results, the method will be valuable for identifying low inventory states.

Figures [5] and [6] present the estimation results for each method for filtered queries and unfiltered queries respectively. For readability purposes we choose to display the z-score of the estimations. Despite the intentional misspecification of the generated data, the LIS method more closely estimates the ground truth odds ratio. Most importantly, it captures all the desired trends with respect to the classes of queries and number of results. The decay shape of the LIS method is only weakly aligned with the ground truth. This is due to the Poisson distribution assumption, a low variance estimate is connected to a low probability value from the pmf; therefore, pulling the estimates down for the low number of search results scenarios.

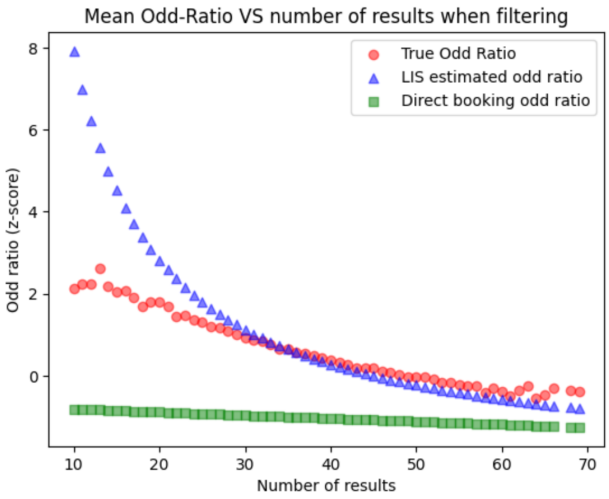

Figure 5: Observed and estimated odd ratios when filtering

In contrast, the direct booking method does not capture well the differences in impact between the two classes of queries, nor does it follow the exponential decay shape of the ground truth. The method

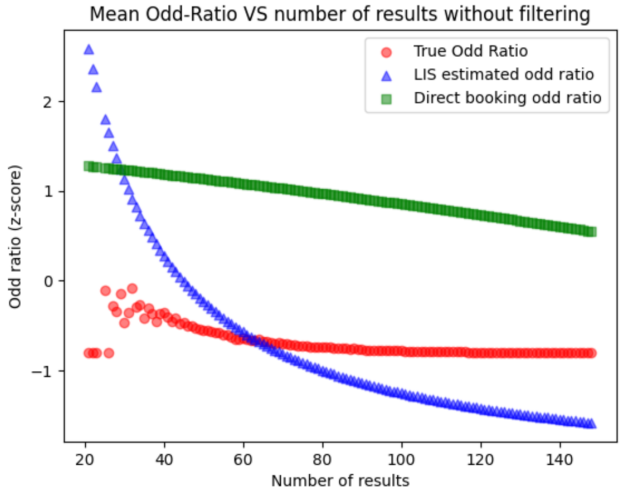

Figure 6: Observed and estimated odd ratios without filtering

suffers from an identification issue as it is not able to account for the confoundedness of intent (through filtering) on the relationship between the number of results and conversion. More specifically, users with high intent are more likely to obtain fewer results but still have a high chance to book, whereas users with less intent observe more results but have low probability of booking. This results in the trend that conversion is negatively correlated with the number of results, as shown in Figure [7]. Including filtering as a feature in the logistic regression helps mitigate this phenomenon, but as shown in the estimation results it is not enough to capture the true relationship between conversion and the number of search results.

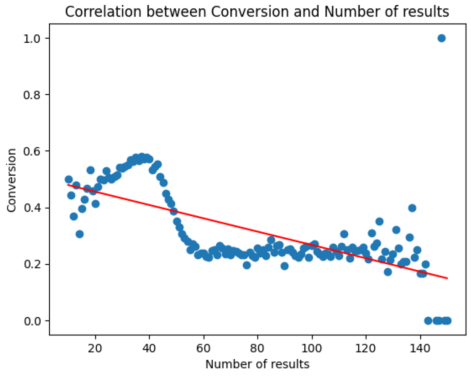

Figure 7: Correlation between number of results and conversion (simulation)

7 ASSESSMENT ON USER QUERIES

7.1 Validity of LIS on Airbnb data

When dealing with real user queries, we don't have access to any ground truth. We, therefore, cannot make any precise assessments of the model performance, but we can bring a body of evidence which shows that the model is useful for our main use case: identifying search queries for which user conversion suffers from a low amount of inventory.

Given a low number of results is correlated with intent and therefore conversion, we do not want to directly predict conversion using *LIS*. However, we can condition on the number of search results, and evaluate if LIS detects lower conversion. In Figure [8] we show that within each bucket of number of results, low inventory state correlates with a worse rate of conversion.

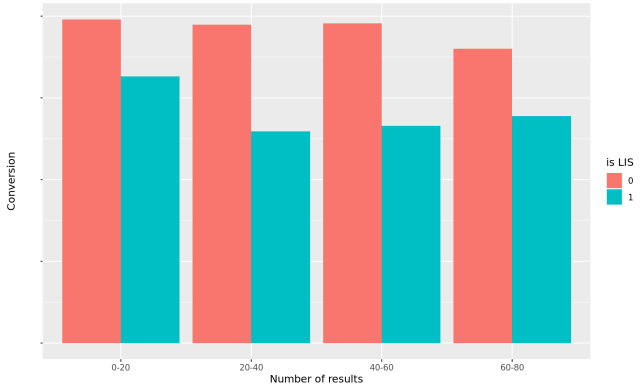

Figure 8: Conversion rate for LIS and non-LIS identified searches across varying number of search results

More directly, we can assess the validity of our causal inference methodology by analyzing its observable implications [9]. We achieve this by comparing conversion rates across three key inputs into our LIS metric: (1) the expected number of results for a search, (2) the expected number of results for only bookers, and (3) the observed number of results. Note that the values for expected number of results are produced by our *BRT Poisson model*. If the mechanics behind our causal formula are valid, we should expect the following trends:

- The larger the positive difference between the expectation of the number of results for bookers and the observed number of results → the greater the incremental impact of LIS, and
- the larger the positive difference between the expectation of the number of results for all searchers and the observed number of results → the greater the incremental impact.

We can see in Figure [9] that the conversion rate is worse (i.e. greater incremental impact) when the positive difference between the expectation of the numbers of results (for all searches or bookers) and the number of observed results is large. Therefore validating our causal inference methodology.

7.2 Product Applications

In practice, our LIS metric can be leveraged both offline and online to improve the user search experience. Offline, we can compute the metric for experiments and analyze how new innovations are driving guests to more or less low inventory states. Additionally, we can utilize the metric for analytics to identify opportunities to improve the search experience, e.g. flagging product paths where

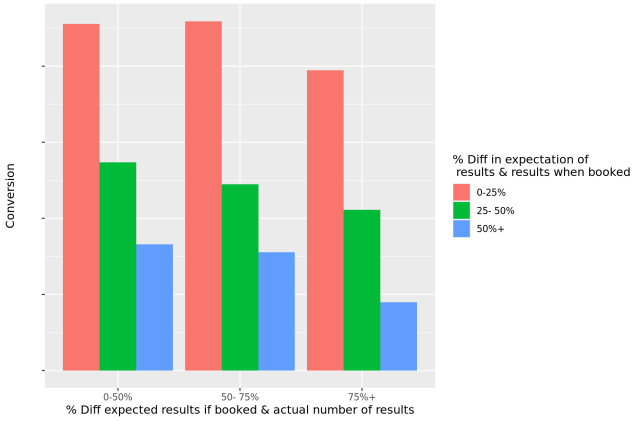

Figure 9: Conversion versus expectations and outcome

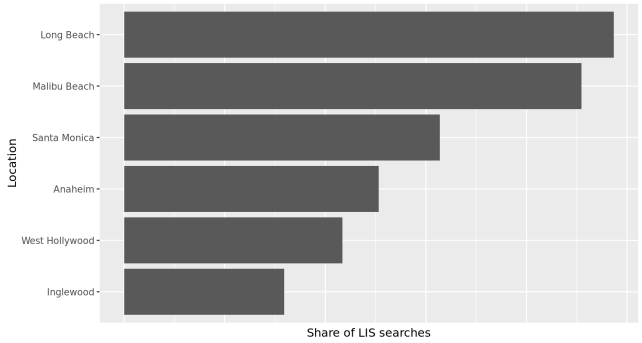

Figure 10: Share of LIS searches per location in Los Angeles

guests often find themselves in a low inventory state. Finally, our metric can be scored online and used to power or as an input to product features, e.g. as the triggering criteria for product interventions to help guests who find themselves with limited available inventory for their search.

It is worth noting that we currently focus on the incremental impact of one additional result. However, depending on the application it could be more appropriate to assess the impact of a percentage increase in results, e.g. 10% more results. Given the non-linear shape of the odds ratio of incremental impact the distance of the comparative point can influence which search queries are flagged as having the highest potential improvements to conversion.

7.3 Application to Inventory management

Another area where our LIS estimator can be applied is inventory management and supply acquisition. The LIS metric directly identifies if a searcher would benefit from more supply for their search, therefore guiding us on what locations and types of guests are most in need of additional supply. A big advantage is that the metric works at the search level which comes with useful dimensions such as number of guests, accommodation type, price point, etc. This granularity allows us to provide actionable insights e.g. "users searching for 5 guests accommodations in Long Beach are facing a high share of LIS states".

For example, if we take the large Los Angeles market, we can tell that searches in the following areas have a high rate of LIS; *Long Beach, Santa Monica, Malibu Beach*, while the following areas have a relatively low share of LIS states; *Anaheim, West Hollywood, Inglewood* (see figure [10]). These insights directly inform us as to where we should, and should not, focus our supply management.

8 CONCLUSION & FUTURE WORK

In this paper, we presented a causal framework on how the number of results returned for a search query can impact a user's booking conversion for Airbnb Search. From this framework, we derived a methodology to identify search queries for which the amount of inventory returned is preventing booking conversion. We demonstrated via simulated data experiments and an assessment on real user queries the validity of our causal model, and its superior performance compared to alternative non-causal methods. Finally, we showcased some product and business applications of our new metric.

In the future we can explore alternative modeling approaches for estimating the expected number of search results for bookers and all searchers. We currently utilize Poisson regression, but a negative binomial approach may perform better given the high dispersion of the number of search results. Ultimately, any improvements in our ability to estimate the expected number of results will lead to an improvement in our ability to identify search queries with an insufficient number of results returned.

ACKNOWLEDGMENTS

We would like to thank Alex Deng, Totte Harinen, Ju Tian and the Airbnb Search Relevance Team for their feedback and suggestions throughout the development of this work.

REFERENCES

[1] A. Colin Cameron and Pravin K. Trivedi. 2013. *Regression Analysis of Count Data*. (2nd ed.). *Econometric Society Monographs*. Cambridge University Press. DOI: 10.1017/CBO9781139013567.

[2] Melissa Garrido, Amy Kelley, Julia Paris, Katherine Roza, Diane Meier, R Sean Morrison, and Melissa Aldridge. 2014. Methods for constructing and assessing propensity scores. *Health Services Research*, 49, (Apr. 2014). DOI: 10.1111/1475-6773.12182.

[3] 2009. *Additive models, trees, and related methods. The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer New York, New York, NY, 295–336. ISBN: 978-0-387-84858-7. DOI: 10.1007/978-0-387-84858-7_9.

[4] Leland Gerson Neuberg. 2003. Causality: models, reasoning, and inference, by judea pearl, cambridge university press, 2000. *Econometric Theory*, 19, 4, 675–685. DOI: 10.1017/S0266466603004109.

[5] Judea Pearl. 1995. Causal diagrams for empirical research. *Biometrika*, 82, 4, 669–688.

[6] Ilya Shpitser and Judea Pearl. 2008. Dormant independence. In (AAAI'08). AAAI Press, Chicago, Illinois, 1081–1087. ISBN: 9781577353683.

[7] Gyanit Singh, Nish Parikh, and Neel Sundaresan. 2012. Rewriting null e-commerce queries to recommend products. In *Proceedings of the 21st International Conference on World Wide Web* (WWW '12 Companion). Association for Computing Machinery, Lyon, France, 73–82. ISBN: 9781450312301. DOI: 10.1145/2187980.2187989.

[8] Zehong Tan, Canran Xu, Mengjie Jiang, Hua Yang, and Xiaoyuan Wu. 2017. Query rewrite for null and low search results in ecommerce. In *eCOM@SIGIR*.

[9] Jin Tian and Judea Pearl. 2012. On the testable implications of causal models with hidden variables. (2012). arXiv: 1301.0608 [cs.AI].

[10] Anita Natalia Varga, Alejandra Elizabeth Guevara Morel, Joran Lokkerbol, Johanna Maria van Dongen, Maurits Willem van Tulder, and Judith Ekkina Bosmans. 2023. Dealing with confounding in observational studies: a scoping

review of methods evaluated in simulation studies with single-point exposure. *Statistics in Medicine*, 42, 4, 487–516. DOI: https://doi.org/10.1002/sim.9628.

Appendices

A SUPPLEMENTARY STEPS TO OBTAIN THE RESULTS ON INCREMENTALITY

$$BookIncr_{qry} = \frac{P(B=1|R+1,QP,I=1)}{P(B=1|R,QP,I=1)}$$

Using Bayes theorem, we can rewrite the numerator and denominator as follows:

$$P(B=1|R+1,QP,I=1) = \frac{P(R+1|B=1,QP,I=1)\times P(B=1|QP,I=1)}{P(R+1|QP,I=1)}$$

$$P(B=1|R,QP,I=1) = \frac{P(R|B=1,QP,I=1)\times P(B=1|QP,I=1)}{P(R|QP,I=1)}$$

and obtain

$$BookIncr_{qry} = \frac{P(R+1|B=1,QP,I=1)\times P(R|QP,I=1)}{P(R|B=1,QP,I=1)\times P(R+1|QP,I=1)}$$

B RELATIONSHIP WITH PROPENSITY SCORE MATCHING

We can also frame the same solution with a propensity score approach. We can view the treatment (TE) as adding one additional result and therefore write the treatment effect (TE) as follows:

$$\begin{aligned}(T=1) &= R+1\\ (T=0) &= R\\ TE &= P(B=1|R+1,QP)P(B=1|R,QP)\\ TE &= P(B=1|T=1,QP)P(B=1|T=0,QP)\end{aligned}$$

Here QP (query parameters) is a list of covariates such as

$$QP \perp T|b(QP)$$

where b(x) is a balancing score i.e. a function of the observed covariates x such that the conditional distribution of x given $b(x)$ is the same for treated (T = 1) and control (T = 0) [2]

A fitting propensity score (that is a balancing score) in that context is:

$$b(QP) = Pr(R|QP) \tag{7}$$

Searches that have the same balancing score, in this case the same expectation of number of results can serve as unbiased subjects to estimate the treatment effect.

The above shows that the same methodology of first estimating the expected number of results given the search parameter to derive the effect on an additional result can be framed as a propensity score matching method.

C SIMULATION CODE

```
# for data validation
def simulate_dataset(num_samples):
    dataset = []
    for _ in range(num_samples):
        intent = random.choice([0, 1])
        filtering = simulate_filtering(intent)
        num_results = simulate_num_results(
            filtering)
        has_booked = simulate_has_booked(intent,
            num_results)
        odd_ratio_results = odd_ratio(intent,
            num_results)

        data_point = {
            'intent': intent,
            'filtering': filtering,
            'num_results': num_results,
            'has_booked': has_booked,
            'odd_ratio_results': odd_ratio_results
        }
        dataset.append(data_point)

    return dataset

def simulate_filtering(intent):
    if intent == 0:
        return random.choices([0, 1], weights
            =[0.8, 0.2])[0]
    else:
        return random.choices([0, 1], weights
            =[0.2, 0.8])[0]


def simulate_num_results(filtering):

    base_rate = 10  # Base rate of num_results
    rate_multiplier = 1

    if filtering == 0:
        rate_multiplier *= 2.5  # Increase rate if
             filter is 0

    num_results = int(poisson.rvs(base_rate *
        rate_multiplier)) + 1
    return num_results

def logistic(r):
    return r / 100

def simulate_has_booked(intent, num_results):
    w = logistic(num_results)
    if intent == 0:
        return 0
    else:
```

```
        return random.choices([0, 1], weights=[1-w, w])[0]

def odd_ratio(intent, num_results):
    if intent == 0:
        return 1
    else:
        return  logistic(num_results + 1) / logistic(num_results)

def incr_poisson(lbda_booked, lbda, num_res):
    numerator = poisson.logpmf(num_res + 1, mu = lbda_booked + num_res + 1) + poisson.logpmf(num_res, mu=lbda + num_res, loc = -num_res)
    denominator = poisson.logpmf(num_res, mu = lbda_booked + num_res, loc = -num_res) + poisson.logpmf(num_res + 1, mu=lbda + num_res + 1, loc = -num_res -1)
    # scipy does not allow non for non int values,
    # we use a trick to account for the exact differnce to 3 dec points
    return max(1, np.exp(numerator - denominator))

# second simulation with more realistic data
def simulate_dataset(
    num_samples,
    p_filter_intent=0.9,
    p_filter_no_intent=0.2,
    intercept = -1,
    coef = 0.03):
    dataset = []
    for _ in range(num_samples):
        intent = random.choice([0, 1])
        filtering = simulate_filter(intent, p_filter_intent, p_filter_no_intent)
        num_results = simulate_num_results(filtering)
        has_booked = simulate_has_booked(intent, num_results, intercept, coef)
        odd_ratio_results = odd_ratio(intent, num_results, intercept, coef)

        data_point = {
            'intent': intent,
            'filtering': filtering,
            'num_results': num_results,
            'has_booked': has_booked,
            'odd_ratio_results': odd_ratio_results,
        }
        dataset.append(data_point)

    return dataset

def simulate_filter(intent, p_filter_intent, p_filter_no_intent):
    if intent == 0:
        return random.choices([0, 1], weights=[1-p_filter_no_intent, p_filter_no_intent])[0]
    else:
        return random.choices([0, 1], weights=[1-p_filter_intent, p_filter_intent])[0]


def simulate_num_results(filtering):
    base_rate = 35  # Base rate of num_results

    rate_multiplier = 1

    if filtering == 0:
        rate_multiplier *= random.choices([1.5, 2, 3])[0]
        # Increase rate of results randomly if filter is 0
    # different option allows more dispersion

    num_results = max(10, min(150,
    int(poisson.rvs(mu = rate_multiplier * base_rate + 20 , loc=-20))))
    # loc allows to increase the dispersion
    return num_results

def logistic(r, intercept, coef):
    results_spec =  coef*r - intercept
    return 1 / (1 + math.exp( -1*(results_spec ) ) )

def simulate_has_booked(intent, num_results, intercept, coef):
    if intent == 0:
        return 0
    else:
        w = logistic(num_results, intercept, coef)
        return random.choices([0, 1], weights=[1-w, w])[0]

def odd_ratio(intent, num_results, intercept, coef):
    if intent == 0:
        return 1
    else:
        return logistic(num_results + 1, intercept, coef) / logistic(num_results, intercept, coef)

# Usage
dataset = pd.DataFrame(
    simulate_dataset(
        100000,
         p_filter_intent=0.75,
         p_filter_no_intent=0.25,
         intercept=0.5, coef=0.05
```

```
    )
)

# Model training

## logistic regression
features = ['filtering','num_results']
target = 'has_booked'
# Create the logistic regression model
model = LogisticRegression()

# Fit the model
model.fit(dataset[features], dataset[target])

# Make predictions
predictions = model.predict_proba(dataset[features])[:,1]

dta_plus1 = dataset.copy()
dta_plus1['num_results'] = dataset['num_results'] + 1
predictions_plus1 = model.predict_proba(dta_plus1[features])[:,1]

# Apply the predictions to the original dataset
dataset['non_causal_odd_ratio'] = predictions_plus1 / predictions


## Poisson regression and LIS estimationm
booked_dataset = dataset[dataset['has_booked'] == 1].copy()
features = ['filtering']
target = 'num_results'

model_p = PoissonRegressor(alpha = 0)
model_p_booked = PoissonRegressor(alpha = 0)

# Fit
model_p.fit(dataset[features], dataset[target])
model_p_booked.fit(booked_dataset[features], booked_dataset[target])


# Make predictions on the original dataset
dataset['predicted_num_results'] = model_p.predict(dataset[features])
dataset['predicted_num_results_booked'] = model_p_booked.predict(dataset[features])
dataset['predicted_odd_ratio'] = dataset.apply(lambda row: incr_poisson(row['predicted_num_results_booked'], row['predicted_num_results'], row['num_results']), axis=1)
```