Simulating Strategic Interaction on Online Marketplaces

[How to Survive Dynamic Pricing Competition]

Marvin Bornstein, Johanna Latt, Jan Lindemann, Nikolai J. Podlesny, Sebastian Serth, Jan Selke

ABSTRACT

Implementing intelligent and efficient pricing strategies for big online marketplaces to make customers buy your product instead of the competitor's is one of the biggest challenges retailers have nowadays. Especially since pricing algorithms can vary heavily from simply rule-based to complex, data-driven strategies. Currently, retailers lack the possibility to test their algorithms appropriately before releasing them into the real world. Additionally, it is hard for researchers to investigate how pricing strategies react and interact based on influences and against each other. We picked up the challenge to create an environment to imitate different market situations in eCommerce. Therefore, we provide the toolkit to simulate and test pricing strategies in various contexts and with different consumer and merchant behaviors on such a marketplace.

1. INTRODUCTION

The field of *dynamic pricing* is concerned with the creation, analysis and application of highly dynamic pricing strategies that change prices of products in reaction to certain variables such as the competitor's actions, demand estimations or certain consumer behaviors. For the last couple of decades those strategies were mainly defined by rule-based actions. Now however, more and more companies start basing their pricing calculations and strategies on technology and data-driven processes. Not only the profit and cost computation is algorithm driven, but also its update and review strategy. One of the most competitive and advanced fields is the algorithmic trading or high-frequency trading on stock exchanges. Also, each one of us has already experienced technology driven price calculations on online marketplaces like Amazon where everything is inherently highly dynamic and price updates can happen within nanoseconds.

Pricing strategies and algorithms currently exist, but handlers of those mechanisms lack the possibility to test them appropriately before releasing them into the real world. This can create huge losses [1, 2, 3, 4]. We picked up the challenge to create an environment to imitate different market situations and test how pricing strategies react and interact when influencing each other.

Contribution: In this work, we briefly elaborate our process of building a distributed and scalable platform to imitate market situations and simulate dynamic pricing algorithms and their effects with potential real world settings. After briefly looking at related work in Section 2, the following Section 3 lists the goals we identified as most important for a good simulation platform and which we tried to address in our solution by using an appropriate architecture as outlined in Section 4. The choreography of different services is described in Section 5 and Section 6 will provide insights in the already implemented algorithms and their behaviors. A user facing interface on top of the RESTful API used for the communication between the services is delineated in Section 7. Furthermore, an evaluation of first simulations and the platform's capabilities and restrictions are elaborated in Section 8 and Section 9. Finally, Section 10 concludes this work.

2. RELATED WORK

The topic of pricing competition has been widely researched and discussed in various research areas. The Bertrand-model from 1883 [5] is a very basic model looking at the situation of two companies offering one exact same product. The result in this case according to the Bertrand-model will always be that the product will be offered at purchase price by both companies. The only differentiating feature for the customers is the price of the product, thus they will always buy the cheaper product which leads to both companies undercutting each other until they both have no profit margin left.

However, the situation in markets with multiple products or products with differentiating features offers a much wider variety of possible outcomes and pricing strategies. Many have researched possible pricing strategies and possible scenarios. Kopalle et al. [6] and Chintagunta and Rao [7] focus on the demand as factor for dynamic pricing models, Martínez-de Albéniz and Talluri [8] investigate the influence of stochastic demand, Gallego and Hu [9] look at perishable products, i.e. products with a finite sales horizon, and Levin et al. [10] consider strategic consumer choices and perishable products as well. Popescu [11] explored models of repricing automation in markets where products have exactly one differentiating attribute such as the seller's reputation.

With the increasing use of online marketplaces in the last

decades, the dynamic pricing topic got more and more important due to the possibility to change virtual prices within seconds, whereas physical prices printed on products in stores were much harder and slower to influence. But also, the consumers' behavior changes in this context as Kannan and Kopalle [12] researched by comparing the physical value chain with the virtual-information-based value chain.
This showed that the consumer behavior is just as much of an important factor in dynamic pricing contexts as the pricing behavior is. However, little effort has been made to build simulation platforms that ease the development and evaluation of good pricing algorithms while taking all of these factors into account. Morris [13] developed such a platform in 2001. However, it is limited in its capabilities, especially since it does not allow huge eCommerce simulations, which is necessary today: The simulation program runs on a single machine, offers a limited set of consumer behaviors, simulates solely finite sales horizons and the pricing updates of the sellers happen only in discrete time intervals that are predefined by the system. Thus, reactions to other sellers are very limited. This does not represent the current situation on online marketplaces very well.

3. DESIGN GOALS

Given the current, very limited possibilities to simulate a huge, dynamic online marketplace, we came up with a set of specific design goals and restrictions, that we wanted to fulfill to make the platform and the resulting simulation as realistic, dynamic and reactive as possible:

Reactivity: Allow pricing algorithms to do fully dynamic price updates and look-ups at any time without being refrained by discrete time intervals to enable truly reactive pricing strategies.

Realism: Enforce real-life restrictions present in every big real-life marketplace such as a limited amount of price updates per time interval to avoid advantages by constantly changing prices.

Scalability and Adaptability: The system should be highly scalable and easily expandable to account for all possible demands such as a very high load, the simulation of huge marketplaces or the addition of completely new components and features.

Flexibility: Offer the greatest possible flexibility to make the system adaptable to user needs and new possible design goals. Such needs are to allow an easy entry and exit of firms, to change purchase costs and customer behaviors, and to adjust all settings of the simulation such as update limits. Also, it should allow the simulation of both, rule-based and data-driven pricing strategies, the latter using machine-learning techniques to estimate the customer choices and sales probabilities.

Security: Prevent fraud in simulation setups with multiple participants to allow the simulation to be used for competitions or education purposes. Fraud could be the deliberate influence on the expenses or profits of other pricing algorithms or the usage of data (e.g. for learning purposes) that is normally not accessible.

Measurability: Offer metrics to assess and evaluate the performance of pricing strategies. Such metrics can be the generated long-term and short-term revenue and profit, the market share and adjustment frequencies. Furthermore, the platform should enable an easy access to these metrics, e.g. through graphs and visualizations.

4. ARCHITECTURE

Confronted with the challenge of creating a high-performance and expandable infrastructure for simulating a marketplace with different merchants and consumers, a microservice architecture was created allowing the user to scale, exchange or add single service ad-hoc and on demand. Each service within our architecture implements one business artifact. This architecture pattern comes with the cost of a communication overhead and requires farsighted API design.
Fig. 1 describes the underlying architecture modeling as FMC[1] diagram. We understand a single realization of this architecture as one simulation universe, meaning that key components are unique in this universe.
When initiating a new simulation universe, it comes along with the marketplace component, as well as the producer component and a management UI for controlling each service. While the producer offers products, the marketplace holds the current market situation and handles price updates and purchases of goods. Each transaction handled by the producer and marketplace is logged to a stream database, namely Apache Kafka[2]. Further, those logs are being analyzed and aggregated through a batch data processing component which, in our case, is Apache Flink[3] and written back into a new Kafka topic. Those details can then be accessed selectively through a socket connection or REST interface provided by a kafka-reverse-proxy. To liven the place up, numerous consumers or merchants may join and participate in this market simulation. By default, one consumer and six merchants are deployed with predefined strategies. Those behaviors are described in Section 6 while the choreography of the single services is delineated in Section 5.

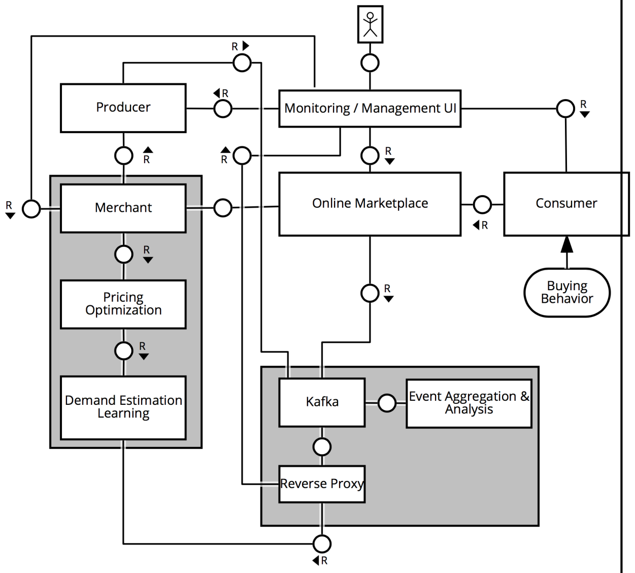

Figure 1: FMC diagram of the Price Wars architecture

[1]http://www.fmc-modeling.org/
[2]https://kafka.apache.org/
[3]https://flink.apache.org/

5. SERVICE CHOREOGRAPHY

As previously discussed, the architecture consists of several services communicating with each other. This communication is done via well-defined RESTful APIs using JSON objects[4]. This helps to fulfill the design goal of reactivity, since each service is totally autonomous and can contact every other service via their exposed RESTful API at any time. There is no need for explicit simulation ticks or action requests.

Due to the micro-service architecture and the design goal to allow real competition of different merchants without cheating, all important routes are secured by authorization tokens. For authenticating all participants in the simulation without the need of a centralized authentication server, a hash-based token and identification system was introduced. It enables the ID-based logging of event messages corresponding to one merchant or consumer. One of the causes to implement this decentralized authentication system was to reduce the number of requests during the simulation.

In the following, we will briefly outline some of the main challenges which were solved.

5.1 Decentralized Authorization

One of the most important aspects of the simulation is the analysis of revenue and thus the performance of the pricing algorithm. In order to generate basic statistics without the merchant (to prevent any manipulation), each relevant transaction requires authentication and is done by another service (see Section 5.2). The only service that always has an up to date list of active and registered merchants is the marketplace. Due to the high density of requests being handled by the marketplace during a simulation, the goal was to prevent additional communication for authorization purposes.

Therefore, all merchants are required to register at the marketplace and to get an authorization token (which is a secret and not shared with any other merchant). This token is enforced when setting a new price or when buying new products from the producer. Each service is capable of transferring a token to an ID by using the following hash function: `base64(sha256(token))`. The resulting ID only depends on the given token. Due to the nature of the used hash function, there is no fast computational way to get the token to a known ID. Thus, this successfully prevents any merchant to pretend being a competitor because the producer requires the token and calculates the ID before logging the expense and returning a new product.

Similar to the real world, the simulation should also prevent unauthorized transactions by merchants to the marketplace. Merchants should offer their products in the same condition as they bought them from the producer. Therefore, each new item posted to the marketplace must contain a valid signature, which is created by the producer. This signature is encrypted and contains product details such as the quality or amount. In addition, the merchant ID is included by the producer and must match the ID of the merchant, that tries to use the product in an offer. As described in the previous paragraph, the producer calculates the ID based on the given token he also uses to log the purchase. With this technique, no merchant is able to get a product without paying for it and without changing its quality.

[4]https://hpi-epic.github.io/masterproject-pricewars/

5.2 Event Log Analysis with Kafka and Flink

All transactions during the simulation are persisted using a Kafka stream database. The log is sourced by the marketplace and the producer with JSON formatted messages and stored in different, so-called *topics*. Each topic can be consumed independently with the preserved order of events. For example, the Flink jobs read all messages within a predefined time range (such as one minute or one hour). Each Flink job aggregates different events from multiple topics and logs the result in an own Kafka topic. The jobs are executed every ten seconds or every minute, depending on the aggregation.

The aggregated messages contain the calculated market share per merchant per time slice and are used by the management UI to display corresponding statistics. Because Kafka only offers stream-based access, the Kafka reverse proxy is responsible for fetching a preset amount of events on request to serve these via a RESTful API. This additional layer is required because available libraries for the Kafka access use a timeout to wait for new events during read or require a fix amount of messages to fetch. In addition, the reverse proxy also implements socket.io, a web technology to push new events directly to the management UI in browsers for live statistic updates.

In addition to being the main statistic source for the management UI, the reverse proxy is also responsible for serving the event log to merchants and to export topics to CSV files. These exports are mainly used for data-driven merchants. As described in Section 5.1, the messages contain a specific merchant ID and are filtered for that ID when exporting data. Therefore, the corresponding merchant token is required and used to filter out non-related events.

6. BEHAVIORS

The addressed solution provides in its default setting already several behaviors for merchants as well as the consumers. In this section, we will outline and describe the underlying behaviors which are currently available.

6.1 Merchant

The merchant-component has one main task: The calculation of a price for a given product. This can either be a new product, purchased from the producer, or an existing product, that is already on the marketplace as an offer. This calculation has to take the trade-off between maximizing the probability to sell a product and maximizing the own profit into account.

To allow an easy start, the platform already offers a set of five rule-based strategies and one data-driven approach, that implements a pricing strategy based on logistic regression [14].

6.1.1 Rule-based merchants

The simple, rule-based behaviors include "Be the n^{th}-cheapest", "Fixed price", "Randomly be the 1st, 2nd or 3rd cheapest ('Random Three')" and the "Gas Station strategy" (also called "Two Bound" in our system). The "Two Bound" strategy sets a minimum and maximum profit margin that can be added to the product's purchase price. The strategy keeps lowering the price to be the cheapest until the minimum bound is reached - then the algorithm sets the price for this product to the maximum profit margin (see Fig. 2).

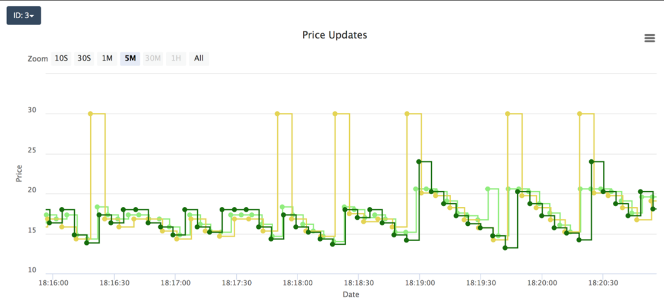

Figure 2: Price Graphs

6.1.2 *Data-driven merchant*

The data-driven behavior follows a pricing strategy aimed at maximizing the expected profit. To do so, a logistic regression model is trained, based on features such as the distance to the cheapest competitor, the price- and quality-rank, the number of competitors or the average price of the product on the marketplace. The data used to learn this model are past market situations in which the merchant sold his own product, i.e. the dependent variable representing whether a product was sold or not is true. The merchant can only use his own sales data to train the model - the access to sales data of other merchants is restricted by the system, just as it is the case in real-life marketplaces. As a result, the logistic regression model outputs the probability with which a product is sold, given a certain market situation and price for that product. To maximize the expected profit, this merchant comes up with a set of possible prices for a product, calculates the selling probability in the current situation, multiplies this probability with the corresponding price and as a result gets the expected profit for each possible price.
All of these behaviors have certain base settings that further determine the behaviors and that can be adjusted during run time. These are the number of starting products the merchant buys from the producer or a minimum profit margin that this merchant will never undercut.
Additionally, an SDK in Python was implemented to facilitate an easy on-boarding and extension of the platform by adding custom merchants.

6.2 Consumer

A consumer-component is included in the deployment providing several buying behaviors which can be chosen, weighted, and reconfigured on-the-fly. Those behaviors range from very subtle approaches like buying the n^{th}-cheapest, first, most expensive or simply random offers on the marketplace up to more sophisticated methods trying to imitate more complex consumer situations. A sigmoid distribution with twice of the producer price as mean is available through the default settings as well as a data-driven "logit"-behavior.
The logit-behavior implements a logistic regression with feature scaling and calculates for each offer the buying probability based on the feature coefficients provided in the behavior settings. Based on this buying probability for each offer, the consumer will actually choose an offer to buy. In this way, potential consumer behavior can be learned on real world data and imitated in the simulation solution.
The default settings for the logit-behavior holds coefficients which were extracted from a real-world use case provided by a big book retail company. For deeper insights in the concepts of logistic regression, the reader is kindly referred to Hosmer Jr et al. [14]'s elaboration.

7. USER INTERFACE

The "Management User Interface" (UI) enables the user to configure, operate and orchestrate the different microservices all in one place.
Consuming the RESTful APIs exposed by each individual service, we use websockets to realize a a real-time streaming connection to the kafka-reverse-proxy component that provides the UI with all necessary data.
The UI allows a user of the simulation to start and stop a simulation in the sense that all components having a state can be started, stopped, accessed and configured here, including their state. These components are the merchants and the consumer. Furthermore, all exposed settings for the behavior of each merchant and the consumer can be viewed, edited and updated.
The stateless components of the simulation, such as the producer and the marketplace, cannot be stopped or started, but configured here as well. E.g., products can be added or deleted, or the marketplace can be emptied with the necessary login-credentials for the underlying database.
If a user wants to register a new merchant and send it into the simulation, they can use the UI to register a new endpoint under which the merchant is running, and in return receive a secret token that is used for authorization (rf. Section 5.1).
Furthermore, the UI also offers easy access to visualizations of the real-time pricing interaction of merchants and to key performance indicators of each merchant, simplifying the process of comparing different pricing strategies.

8. EVALUATION

We used the platform built to evaluate the performance of the already provided merchant behaviors. Thus, we wanted to examine the quality of and the possibilities offered by the simulation. We also evaluated the amount of data generated by the simulation and tried to identify bottlenecks for the system's performance.

8.1 Merchant Performance

We assumed that the exemplary data-driven approach based on a logistic regression pricing model promises higher profits than common rule-based behaviors because of its better adjustment to the consumer behavior in pricing. To test this hypothesize, we performed a one-on-one evaluation of the 6 provided merchant behaviors from Section 6.1. We ran each evaluation with the following settings:

- Duration: 20 minutes
- Consumer:
  - Max. 20 purchases per minute
  - 100% logit-behavior
- Producer: 1 product with 4 qualities
- Merchant:
  - 3 products in stock per merchant
  - 2 updates per second

The results can be found in Table 1.
We also ran an evaluation with all merchants running at once to see whether the data-driven merchant would beat

Table 1: Performance results of the one-on-one merchant behavior evaluation. A: Cheapest, B: Second Cheapest, C: Random Thee, D: Two-Bound, E: Fixed Price, F: Machine-Learning

	A	**B**	**C**	**D**	**E**	**F**
A	▨				E +90%	
B		▨			E +120%	
C			▨			
D				▨		
E					▨	
F						▨

Table 2: My caption

	A	B	C	D	E	F
A	▨					F +150%
B		▨				
C			▨			
D				▨		
E					▨	
F						▨

the other, rule-based merchants. The same settings as above were used except for the consumer who bought a maximum of 60 items per minute to account for the increased number of merchants. The results can be seen in Table 3.

Table 3: Performance results of the all merchant evaluation. The numbers are the profit gained by each merchant after 20 minutes of simulation, i.e. the higher the better. A: Cheapest, B: Second Cheapest, C: Random Thee, D: Two-Bound, E: Fixed Price, F: Machine-Learning

A	**B**	**C**	**D**	**E**	**F**

The results clearly show that the data-driven merchant outperforms all other, rule-based merchants in a one-on-one setup. This is in line with the hypothesize we formulated above. However, the simulation with six merchants at once disproves this hypothesize - the data-driven merchant only had the second best result and was beaten by the two-bound, rule-based strategy, that clearly lost against the data-driven merchant in the one-on-one simulation. This result shows very well how important and unpredictable the influence of interaction effects between multiple merchants can be on the results and that simply testing and evaluating strategies on a one-on-one basis is not sufficient to assess its performance. Thus, these results demonstrate the usefulness and possible applications of the platform built. It allows to account for these unforeseeable interaction effects by enabling the user to run quick and complex simulations of possible real-world setups with an almost arbitrarily high variety of merchants and strategies. At the same time it offers comprehensive evaluation metrics, giving the user immediate feedback on the performance of each merchant, fulfilling many of the design goals we postulated at the beginning in Section 3.
Of course, further tests have to be run with other data-driven strategies as well, especially to investigate how data-driven strategies perform when they play against each other and not only against rule-based strategies.

8.2 Produced Data

We did an evaluation of the data size produced by a simulation using the following benchmarks:

- Duration: 1 hour
- Products: 4000 with 4 different qualities each, i.e. 16,000 products in total
- Merchants: 5 merchants
  - 750 products in stock per merchant
  - 17 requests per second per merchant
- Consumer: Max. 1000 purchases per minute

The amount of data generated and persisted per service can be seen in Table 4.

Table 4: Data size generated by a 1-hour simulation with x merchants, 4000 products and max. 2000 actions per minute

Service	**Data Size**
Kafka	2.3 GB
Merchants (total)	25 MB
Marketplace	24 MB
Producer	4 MB
UI	2 MB
Consumer	60 KB

is this really data generated in one hour? I think it also includes data that is just always present, eg the 4 MB at the producer won't change anymore, they are just due to the high number of products

we started with 7 merchants, one stopped at the beginning and two died later - do we say we tested it with 4, 5, 6 or 7 merchants...?

8.3 System Load

The developed system is able to handle very high loads in terms of the number of merchants, offers or products. The only potentially problematic bottlenecks are the user interface (see also Section 9) and the amount of produced data as stated in the previous section, depending on the duration of the simulation and the underlying machine's capacities.
As shown during the evaluation of the data sizes, a very high number of products is no problem for the system - only the creation of those 4000 products took some time, but the overall performance of the simulation itself was not affected negatively.
The load on the marketplace is determined by the number of requests per minute that a merchant and the consumer can make. This number is adjustable to allow the simulation of times of higher or lower demand on the consumer side (such as Christmas) or to simply speed up the simulation if all parameters are increased equally. As implied above, we ran simulations with more than 100 requests per second without any problems. Also, simulations with 6 merchants at once,

doing 2 updates per second each, and a consumer buying 60 products per minute, were absolutely no problem for the system.

9. FUTURE WORK

While providing an easy and comprehensive way to simulate different market situations based on a variety of consumer behaviors and competing merchants, the current solution still has a lot of potential for extension and improvement.
In the current setup, the producer only offers goods without any expiration date. But when thinking about plane or festival tickets, perishables or any other short-life products, pricing strategies might perform very differently and have to adapt to completely new features. Thus, it would be interesting to add the possibility to offer such perishable products. In consequence, a more comprehensive notion of time would have to be introduced throughout the whole system, and the marketplace would have to check and verify a product's expiration date with the producer. Additionally, the behaviors of the consumer as well as within the merchants need to react on those additional attributes.
Another very interesting case to cover in the near future could be consumer ratings and how they influence pricing strategies. Having now such an environment to simulate different market situations and consumer demands, different consumer ratings may also influence pricing strategies significantly.
Lastly, it would be interesting to extend the simulation to support buying strategies within merchants in addition to pricing strategies, meaning that merchants no longer receive a fixed number of random products, but can decide on what, when and how much they want to buy themselves. This would add a lot of additional complexity to the design of merchant strategies, but would be an even closer simulation of the real world.
We are also aware that the current solution is far from being perfect. Components of the current solution that need further revision in the future are, in particular, the UI and also the Kafka-related components.
The user interfaces turned out to be a bottleneck in the current setup when the simulation is under a high load, i.e. there are many price updates and sales. Especially the real-time price interaction graphs start to lag heavily or even crash the browser under high load, making parts of the UI almost unusable. To solve this problem, a complete refactoring of this UI component or a switch of the underlying third party charts library might be necessary.
The Kafka-components might need to be refactored in terms of the size of the produced data. As shown in Section 8, the amount of data produced in just one hour of simulation is above 2 GB. While that amount should be rather easy to handle on most systems, potential long term simulation - as might be needed for more sophisticated data-driven pricing strategies or long-term algorithm evaluations - might produce more problematic data amounts. A more considerate handling of outdated data as well as a stronger compression of current data might be needed.

10. CONCLUSION

In this work, we presented a distributed and scalable platform to imitate market situations and simulate dynamic pricing algorithms and their effects with potential real world settings. With this toolkit, handler may be enabled to evaluate their pricing strategies appropriately before releasing them into the real world where they can create huge loses. Simultaneously, students and researchers may now implement and evaluate how pricing strategies react and interact based on influences and against each other.
The source code and its technical documentation will be publicly available at github.com[5] while a screencast is accessible on YouTube[6].

Acknowledgments

As part of this elaboration, special thanks goes to Dr. Matthias Uflacker, Dr. Rainer Schlosser, and Martin Boissier for their continuous support and supervision. Also, we thank all attendant members of the EPIC research group for their fruitful discussions.

References

[1] Matthias Uflacker, Rainer Schlosser, and Christoph Meinel. Ertragsmanagement im wandel–potentiale der in-memorytechnologie. *Handel 4.0: Die Digitalisierung des Handels-Strategien, Technologien, Transformation*, page 177, 2016.

[2] Rainer Schlosser and Martin Boissier. Optimal price reaction strategies in the presence of active and passive competitors. In *Proceedings of the 6th International Conference on Operations Research and Enterprise Systems (ICORES)*, pages 47–56, 2 2017.

[3] Rainer Schlosser. Stochastic dynamic multi-product pricing with dynamic advertising and adoption effects. *Journal of Revenue and Pricing Management*, 15(2): 153–169, 2016.

[4] Rainer Schlosser, Martin Boissier, Andre Schober, and Matthias Uflacker. How to survive dynamic pricing competition in e-commerce. In *RecSys Posters*, 2016.

[5] Joseph Bertrand. Théorie mathématique de la richesse sociale. *Journal des Savants*, pages 499–508, 1883.

[6] Praveen K. Kopalle, Ambar G. Rao, and João L. Assunção. Asymmetric reference price effects and dynamic pricing policies. *Marketing Science*, 15(1):60–85, 1996.

[7] P. K. Chintagunta and V. R. Rao. Pricing strategies in a dynamic duopoly: A differential game model. *Management Science*, (11):1501–1514, 1996.

[8] Victor Martínez-de Albéniz and Kalyan Talluri. Dynamic price competition with fixed capacities. *Management Science*, 57(6):1078–1093, 2011.

[9] G. Gallego and M. Hu. Dynamic pricing of perishable assets under competition. Technical report, Rotman School of Management, University of Toronto, 2008.

[10] Yuri Levin, Jeff McGill, and Mikhail Nediak. Dynamic pricing in the presence of strategic consumers and oligopolistic competition. *Management Science*, 55 (1):32–46, 2009.

[5]https://github.com/hpi-epic/masterproject-pricewars
[6]https://www.youtube.com/watch?v=bqXSi5cv8cE

[11] Dana Popescu. Repricing algorithms in e-commerce. *INSEAD Working Paper Series*, 2015.

[12] P.K. Kannan and Praveen K. Kopalle. Dynamic pricing on the internet: Importance and implications for consumer behavior. *International Journal of Electronic Commerce*, 5(3):63–83, 2001.

[13] Joan Morris. A simulation-based approach to dynamic pricing. Master's thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, 2001.

[14] David W Hosmer Jr, Stanley Lemeshow, and Rodney X Sturdivant. *Applied logistic regression*, volume 398. John Wiley & Sons, 2013.