Reinforcement Learning for Dynamic Pricing - Q - Learning
Modeling and Simulation GC86120 / I6400
CUNY - The Graduate Center
Nancy Sea and Yaffa Atkins

Abstract:

We use the Airline Revenue Management System (ARMS) as a framework, we can define the parameters and events that will be used as part of the modeling formalism. We will provide the parameters of the airline operation; the demand function, pricing function, arrival process of the customers booking, seat inventory etc, and how each variable interacts and functions with one another to maximize the revenue. This paper will highlight how dynamic pricing, specifically Q-learning can be applied as part of the environment to coordinate and find the optimal pricing and maximizing profits.

Intro:

Modeling and simulation (M&S) is powerful and can be used for learning the complex structure and behaviors of airline revenue management systems (ARMS) so that we can maximize profit. Through the mathematical modeling of the system, airlines would be able to run their dynamic pricing model on the simulation producing various scenarios and generate optimal airline prices accordingly. This paper will discuss modeling the ARMS with a focus on maximizing revenue through dynamic pricing.

Dynamic pricing is used across the market for various revenue management systems. Using dynamic pricing, prices can be optimized in real time to maximize revenue. In traditional pricing, there is a clear relationship between the demand and the price. When that relationship is known, the optimal price can be easily calculated. Algorithm 1 shows the most basic relationship between price and demand. This relationship can be expounded upon to include other various parameters and can be estimated using historical data and regression analysis.

$$p^* = \underset{p}{\operatorname{argmax}} \; p \times d(p)$$

Algorithm 1. Basic relationship between price and demand.

This traditional method works well when the relationship between price and demand is known and does not change often, but in many real life markets, both the environment and the relationship between price and demand are constantly changing and constantly need to be re-evaluated, here traditional methods are not enough, the calculated optimal price and actual optimal price can be quite different resulting in a loss. In scenarios like this, we turn to dynamic pricing algorithms.

Using machine learning and artificial intelligence, dynamic pricing is a method of predicting an item's optimal price, receiving real time feedback on that price prediction, and making any necessary adjustments to the model based on the feedback, this all happens in real time. This allows the system to respond to any changes immediately and optimize revenue while keeping consumers happy. Additionally, we want to use a method that works well with noise and uncertainty. Humans and their choices are not always predictable and it is important to account for that, and any other noise in dynamic pricing.

Using modeling and simulation, we can test our dynamic pricing algorithm on an environment to check its performance. Modeling and simulation help us understand how the revenue management system works so that our model can learn the system and learn the best policies to use when assigning various prices to various states to optimize revenue. Modeling and simulation replicates and acts on the system to learn its behavior, especially the what if scenarios. This is very important because any errors in pricing results in the potential profit not being actualized and possibly even a loss of profit. It is best to experiment with our model in a simulated environment before using it in the real world.

There are multiple models for dynamic pricing such as Bayesian, Decision Tree, and Reinforcement Learning. The Bayesian model uses historical data to predict the probabilities of different demands and the corresponding prices that will optimize revenue for those demands. As more data is entered, the believed optimal price is increased or decreased based on feedback. Bayesian models are great for predicting in real time with uncertain demand. Decision tree models are machine learning models which help businesses understand which parameters have the biggest impact on prices, the various price options, their consequences, and probabilities. Decision tree models build a tree-like model which shows the relationship between price and demand and uses different pricing algorithms for different scenarios. Reinforcement Learning uses trial and error to learn the optimal relationships between states and prices over time based on feedback from the system which rewards good prices. The model learns from experience and adjusts its understanding of this relationship overtime to maximize reward. Reinforcement Learning is great for complex environments which have many factors impacting demand.

For the purposes of this paper we will be focusing on Reinforcement Learning since Reinforcement Learning learns from its own experiences, training on live feedback, and therefore it does not require historical data for training can be difficult to access for the public in many use cases. Additionally this algorithm is simple to code and is continuously learning and improving, considering the whole environment for its specific use case, aiming to find the best price. Reinforcement Learning is also specifically great for modeling in cases of uncertainty since it is learning of the actual environment itself as mentioned, since often in dynamic pricing there is a lot of uncertainty. Figure 1 shows a basic representation of a Reinforcement Learning Model. The “Agent” refers to the Machine Learning algorithm. The “Environment” refers to the simulated or actual environment which contains the relevant factors. The “State” is the state of

the environment, the “Action” is the action the agent makes based on the environment, and the “Result” is the feedback from the environment regarding the chosen action.

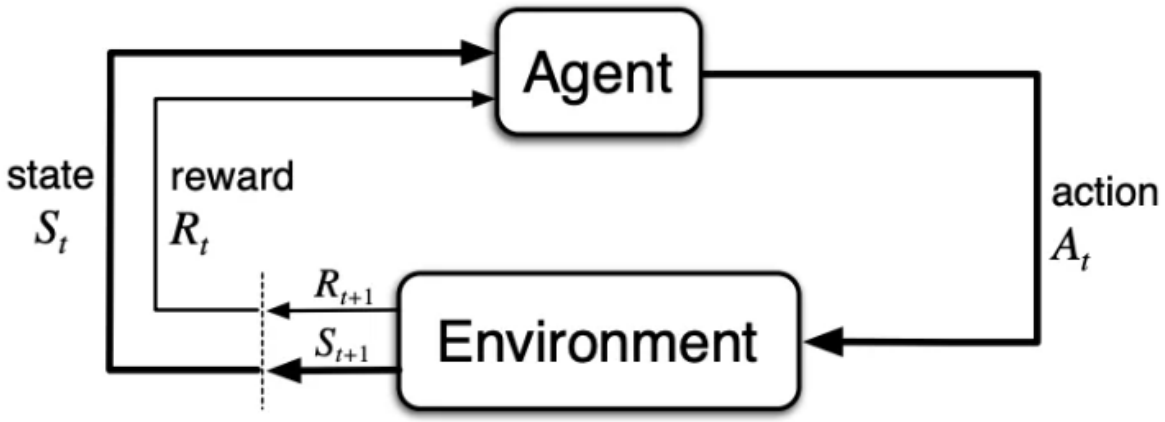

Figure 1. Basis reinforcement learning model.

Related Works:

Shihab [2020] explores the application of deep reinforcement learning to the problem of dynamic pricing and seat inventory control to address the limitations of heuristic-rule-based decision models. The proposed model, DeepARM, is a reinforcement learning model that is capable of directly learning optimal policies from its interaction with the markets. This Q-Learning model is trained in an air travel market simulator built using historical market and booking data.

Pinheiro et. al [2022] adapts the earning while learning (EWL) problem to the airline revenue management environment to generate pricing policies by relaxing the assumption of managing simultaneous flights to a problem of single-leg flights instead. The work focuses on estimating the right demand price sensitivity as an inaccurate estimation will produce suboptimal policies. The EWL method shows that there is value in price experimentation so that we can explore the different amount of revenue that the model would be able to make.

Revenue Management: Models and Methods [Tallur et. al, 2008] introduces different industries that revenue management systems models would benefit. RMs focuses on making better demand decisions through mathematical modeling, processes and systems. Specifically, with the airline industry, this paper highlights the critical issues that RM should focus on, that being demand modeling that takes into account overbooking and cancellations.

Problem Definition (System):

Our goal is to replicate the airline environment and run our model on that system so that the agent can learn the environment and map states to actions which maximize revenue. To simulate different policies the agent learns through applying dynamic pricing Reinforcement Learning problems, we first define the system that our model will be working on. By framing our problem as maximizing revenue for the Revenue Management System, we are able to define some parameters of our system that we are working with. Figure 2 shows an overview of the environment we are working with.

With airfare, the airline is constantly trying to maximize their revenue, whether by charging extra for overweight suitcases, charging for flight changes or cancellation fees, paying for your desired seat, and much more. Flight prices are constantly changing based on demand, depending on the season, day of the week, time of the flight, legal holidays, competing prices, and much more. There are many constantly changing factors impacting the prices airlines charge, so much so that the actual demand function is not actually known, despite that airlines need to maintain a balance of maximizing their revenue without charging too much money so that customers are still buying flights.

If flights are getting canceled and backed up or delayed, the airline loses money, so they may raise prices to compensate for that, but they need to ensure they do not charge too much otherwise nobody will buy tickets and they'll lose more money. In modeling and simulation it is important to take into account all of these what if scenarios so that our model will learn how to deal with them. Some "what if" scenarios we are pointing out in this project are: nobody misses their flight and there's overbooking, delays, bad weather and planes aren't able to fly, something is wrong with the plane and all passengers need to be rebooked or need a refund. It is important to factor in what will happen with our system when bad or unexpected events occur.

For simplicity, we will frame this problem as a single-leg problem, we are only looking to maximize the revenues gained from airfares starting at point A to point B. with flights departing at every timestep T. Maximizing revenue means that we would want to 1) fill up all the number of seats on the flight 2) each customer pays at an optimal price they are willing to pay. The system aims to balance both supply and demand while maximizing revenue one each flight. One thing to note is there are both business and leisure travelers and the airline can offer two types of fare classes with the assumption that the former group of travelers are willing to pay much more than the later. There are two different demand curves for these customer segmentations.

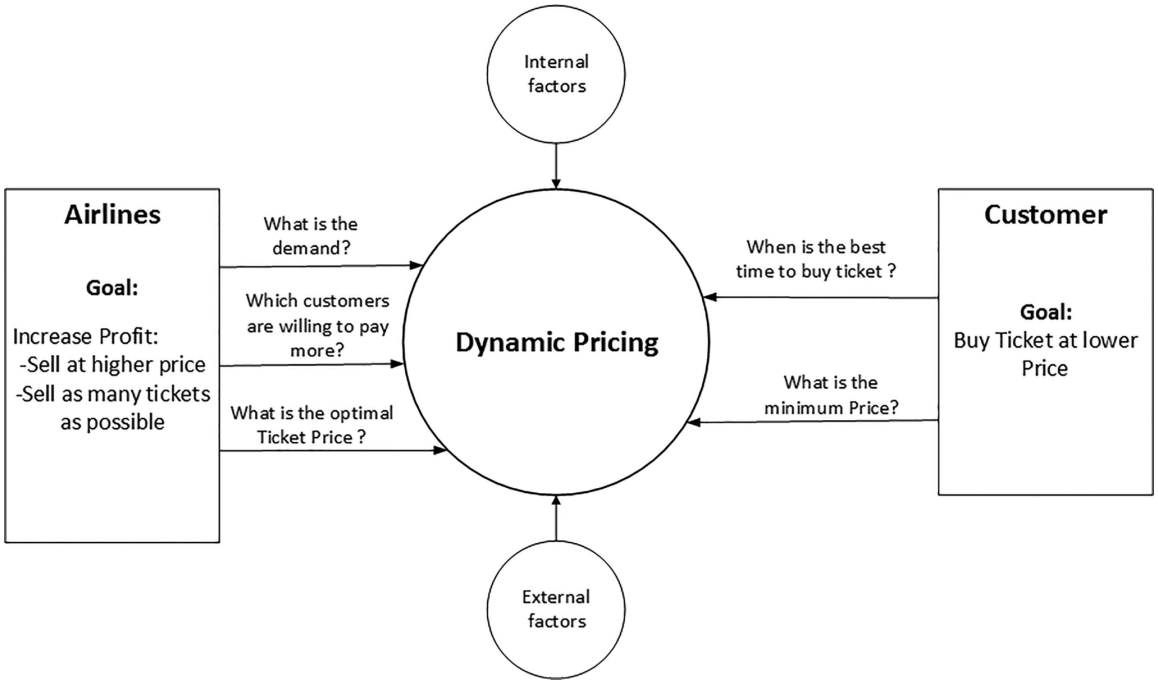

Figure 2. Basic Airline pricing environment.

Model:

When choosing the right model for dynamic pricing, an agent would consider what their main goals are, the complexity of the dynamic environment, scalability, availability of historical data, and the balance between accuracy versus computationally complex and expensive. TensorFLow, PyTorch and scikit-learn are all great libraries that can be integrated with simulation.

Reinforcement Learning learns through trial and error and does not use historical data, additionally it does not have a specific demand function and instead that demand function is like a black box. There are various algorithms and approaches to Reinforcement Learning such as Markov Decision Processes (MDPs), SARSA (State-Action-Reward-State-Action), Q-learning, and Deep Reinforcement Learning. MDPs is a model based algorithm, it models the decision maker, states, actions and rewards. It assumes complete knowledge regarding the system dynamics and transition probabilities between states and their rewards and aims to maximize the accumulated reward. SARSA is a model free algorithm and does not assume prior knowledge of system dynamics, rather it learns through interacting with the system and observing the reward obtained for actions it takes at each state, it follows a policy and improves that policy. Q-learning is similar to SARSA in the sense that it is model free and learns from interacting with the environment, but it is not following a set policy, it's self directed. Deep Reinforcement learning is used to navigate high dimensional and complex scenarios.

We are using Q-learning because it is model free and off policy so it explores more and is often used as a great and simple algorithm for reinforcement learning. Q-learning uses q-values or action values learned overtime. The q value Q(S,A) represents the reward for taking action A at state S. The estimated q-value gets updated each time the action is used with the state.

For our model, the agent, which is the algorithm, learns the environment overtime through interacting with it. Each different situation the agent encounters is known as a state and at each state the agent can pick from a selection of actions. Choosing each action at each different state returns a reward which the agent attempts to maximize. The goal of the agent is to pick the best action for any given state. At each state, the agent chooses an action, observes the reward feedback from the environment, then takes another action based on that reward. Once an agent reaches a state where there are no other actions to take, it's the completion of an episode.

At every action, the q value for that state and action gets updated with the Temporal Difference Update rule which is shown in algorithm 2. S is the current state. A is the current action picked by the algorithm for the current state, S' is the resulting next state, A' is the next best action using the current q-values, R is the current reward observed from the current action choice at the current state, y() is the discounting factor for future rewards since according to this method, rewards get less value each action, and alpha is the learning rate or step size of updating the q-values.

$$Q(S,A) = Q(S,A) + \alpha(R + \gamma \max_{a} Q(S', a) - Q(S,A))$$

Algorithm 2. TD update used to update q values.

The agent chooses an action using the epsilon greedy policy. The policy says that with probability epsilon, choose a random action, and with probability 1-epsilon, choose the action with the highest q-value. Q learning balances the dynamic between exploration and exploitation. Exploration is exploring other state action combinations we have not tired yet and do not know much about, this can be risky since its possible the state and action are a bad combination and will return poor revenue, but simultaneously, we can learn new and exciting state action combinations that return great revenue which we would not have known about otherwise. Exploitation is the act of constantly sticking with state action combinations we know to be good, this is playing it safe since we know we will get a good reward, but we might miss out on exploring better potential we haven't yet discovered. It is important to balance the two.

Another way we can explore and model this problem is through the multi armed bandit, this method also focuses on exploration vs exploitation. Balancing exploiting the option we know so far to be the best, or exploring other and newer options to see if they could possibly produce better results. A multi armed bandit is an agent with various arms and each arm represents another method or algorithm that can be used to obtain the same goal, the bandit or agent then goes with the arm who's method returns the best results. Multi armed bandits can also use epsilon greedy to balance exploring and exploiting the different arms. For our use case, each arm can be a different type of dynamic pricing algorithm or different type of reinforcement

learning algorithm. This is expensive and difficult to program, but it can have great benefits since it shows the full picture and truly returns optimal results.

Simulation:

One way to model our simulation is the following, our problem is one of a single-leg, single origin-destination with two fare classes each with different prices. We only look at one flight that departs once every t timestep, the variables of the system are updated after every flight departure to ensure we have captured the changes of each state. The flight capacity is set to a fixed amount but the allocations of how many seats belong to class fares 1 and 2 can be determined dynamically.

The first input to the model is the arrival of the first customer to book the ticket of a flight. This arrival and those after are randomly generated according to a Poisson distribution process. For each customer, the model checks the availability of the tickets and fares, the predicted demand for this flight based on historical data, and the price function that would optimize this revenue. If the customer decides to purchase the ticket (determined by some probability distribution or price elasticity of demand), the model updates its inventory, the demand, records this price in its database and updates its revenue. If the customer decides not to purchase the ticket, the updates will not happen but we could record the parameters that had caused them to not complete the purchase as this could help in learning the buying behavior of customers. Customers continue to arrive until there are no seats left or it is departure time. However, the model will have to take into account the probability that a subgroup of passengers will cancel their flights or modify their travel dates thus the inventory/seat allocation needs to take this into account. Though we note that sometimes overbooking does happen, in which case we would offer the passenger compensation or to move their flight to the next available one, thus reducing the capacity of the next flight as well as the revenue that we could get if that particular fare is higher than the one previous. The model generates the maximum revenue when all seats are filled at the optimal price that the pricing optimizer has determined taking into account all these factors. A visual of this environment can be seen in Figure 3.

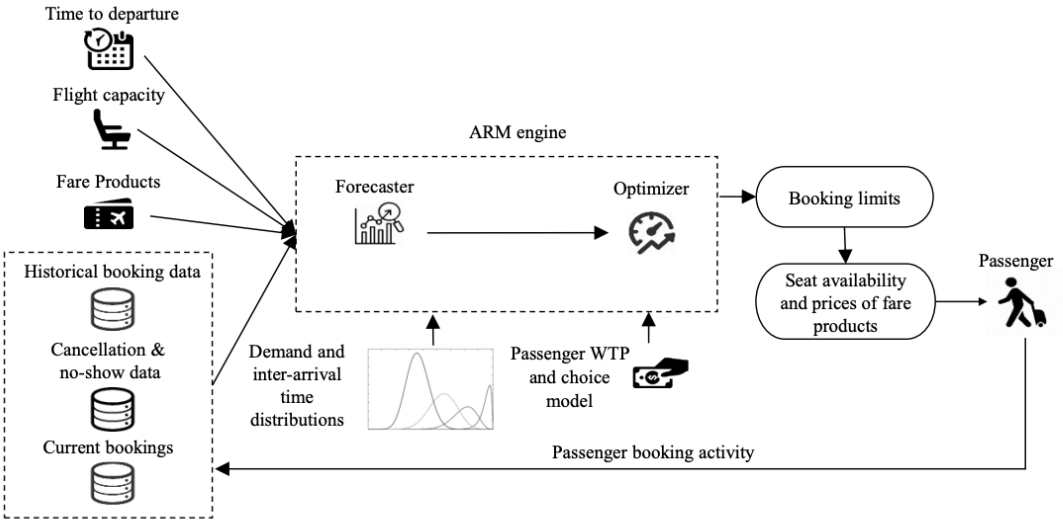

Figure 3: Simplified model of ARMs to optimize seat inventory allocation and fare pricing Shihab [2020]

The above model describes a Discrete Event System model. Though for our purpose of applying Q-learning, we will further simplify this model. Learning from historical data requires actually having the data, takes time, and requires large computational power Q-learning is a often a better alternative for our use case since it learns from its own experience in the actual environment and does not require a demand function, this is helpful since the demand function in this type of scenario would be quite complex with many parameters and would have a lot of noise. Although Q-learning in itself is not a discrete event simulation framework, it can be used within one by combining the simulation environment with the Q-learning algorithm which is what we aim to do in our project.

We choose discrete event simulation since it fits our use case scenario. Using DES, discrete event simulator software, we can model dynamic pricing simulations. DES is able to model complex systems with discrete events like the constant change of prices due to change in demand and other external conditions. Simulation allows us to plan for the actual system, including all the various what if scenarios. This way we can optimize pricing with all the correct knowledge. Using Reinforcement Learning, the model learns policies and gains insights into the mechanisms of the system, we can evaluate our model by looking at revenue and customer satisfaction.

ARMS with the focus on revenue maximization is suited for a DES approach since reinforcement learning is looking at the state of the environment which takes into account different variables such as the behavior of airline operations, arrival of customer bookings, availability of seats/fare classes, and much more, DES can focus on the change of these variables. The simulation will help the airline anticipate demand for the next flight which is in the same state.

First we need to define the DES framework by defining the environment. Simulate the environment with all parameters that would be relevant to the environment state and impact the demand or price such as customer rate of purchasing plane tickets, the competitions prices, seasons, day of the week, and more. Include as many parameters you are able to, or as many as you wish. Define all the possible states and actions, this includes specifying which parameters are relevant in determining a state and all available prices. Run q-learning in this environment until a stopping position is reached, and at that point you can gradually decrease the learning rate to favor exploitation as the learned policy gets stronger and solidified. Once done, assess the results.

In our code, we did not simulate the environment, rather we focused on the q-learning algorithm. In algorithm 3 you can see pseudo code for this algorithm. We have a set learning rate and exploration rate, a given cost, and basic demand function which states that the more expensive the item, the lower the demand and vice versa, as explained earlier, the actual demand function is unknown. For a given state, an action is chosen using epsilon greedy, so either at random, or the best action for the given state, which is the action with the highest q-value. Using TD update, the q-value is updated based on positive or negative feedback from the system each time a state action combination is chosen to learn the best policies. The agent or model in continuously learning and improving.

initialize arbitrarily $Q(s,a) \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$ and $Q(\textit{terminal-state},\cdot) = 0$
repeat
  initialize S
  repeat
    choose A from S using policy derived from Q
    take action A and observe R, S'
    $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_a Q(S',a) - Q(S,A)]$
    $S \leftarrow S'$
  until S is a terminal state
until all episodes are observed

Algorithm 3. Pseudo code for Q-learning.

In our example, the environment would be the airline market, the state would be the current price and demand which is based on season, holidays, availability, competing prices etc, the agent would be our dynamic pricing algorithm, action would be to increase or decrease the flight price, as well as any other ideas we might come up with such as discounts and bonuses. Reward would be the end profit resulting in the agent's decision.

Discussions and future works:

Through a thorough modeling of an environment that we wish to analyze, modeling and simulation allows us to gain more insights for our goal of revenue maximization. We are able to properly prepare different outcomes for different scenarios with our “what-if” section. We are able to understand the dynamics of operation and use that to make more informed operational decisions. With the DES, we can see how one parameter affects another, how seating inventory affects demand, how demand affects revenue and so forth. On the other hand, dynamic pricing’s automated workflow allows quicker learning and turn-around for the purpose of maximizing profits through a black-box approach. Q-learning is great for long-term outcomes as it learns from its previous iteration and improves the next.

The reinforcement learning algorithm that we used could be optimized further by hyper-parameter tuning as well as factoring in more parameters. A powerful approach would be to combine elements of both models. Utilizing the proactive approach of the DES ARMS model, extracting the scenario-specific insights to refine the Q-learning algorithm for efficient calculation in a dynamic and uncertain market environment.

Conclusion:

In this paper, we used the Airline Revenue Maximization System to explore the different simulation methods to optimize the airfare tickets and maximize the firm’s profit. We explored two different approaches: mathematical modeling and simulation of the system and a black-box machine learning approach.

Modeling and simulation of this environment requires specific domain knowledge of the operation itself. With the use of historical data and state updates, we achieve valuable insights into the function of the system and how one state and action affects one another. Q-learning is a much cheaper alternative as it learns through trials and errors whilst still looking for the optimal price so that we obtain the maximum revenue.

References:

Abdella, Juhar Ahmed et al. “Airline Ticket Price and Demand Prediction: A Survey.” *Journal of King Saud University - Computer and Information Sciences* 1 May 2021: 375–391. *Journal of King Saud University - Computer and Information Sciences*. Web.

Dasani, Div. “Single-Agent Dynamic Pricing with Reinforcement Learning.” *Northwestern University*.

Dilmegani, Cem. “Reinforcement Learning: Benefits & Applications in 2023.” *AIMultiple*, research.aimultiple.com/reinforcement-learning/. Accessed 22 May 2023.

Ferrara, M.. "A reinforcement learning approach to dynamic pricing." 2018

Induraj. “Implementing Dynamic Pricing Strategy in Python - Part 3 - Reinforcement Learning.” *Medium*, 2 Mar. 2023, induraj2020.medium.com/implementing-dynamic-pricing-strategy-in-python-part-3-reinforcement-learning-1aaeeb0c1591.

K. T. Talluri, G. J. van Ryzin, I. Z. Karaesmen and G. J. Vulcano, "Revenue management: Models and methods," 2008 Winter Simulation Conference, Miami, FL, USA, 2008, pp. 145-156, doi: 10.1109/WSC.2008.4736064.

Katsov, Ilya. “A Guide to Dynamic Pricing Algorithms Using Reinforcement Learning.” *Grid Dynamics Blog*, 12 May 2022, blog.griddynamics.com/dynamic-pricing-algorithms/.

Pinheiro, Giovanni Gatti, Michael Defoin-Platel and Jean-Charles Régin. “Optimizing Revenue Maximization and Demand Learning in Airline Revenue Management.” ArXiv abs/2203.11065, 2022.

Shihab, Syed Arbab Mohd. “DeepARM: An Airline Revenue Management System for Dynamic Pricing and Seat Inventory Control Using Deep Reinforcement Learning.” *Iowa State University*, 2020.

“Deep Reinforcement Learning with Rllib and Tensorflow for Price Optimization.” *Mawazo*, 10 June 2020, pkghosh.wordpress.com/2020/06/08/deep-reinforcement-learning-with-rllib-and-tensorflow-for-price-optimization/.

“Dynamic Pricing Models: Types, Algorithms & Best Practices.” *Aporia*, 19 Mar. 2023, www.aporia.com/blog/learn/machine-learning-for-business/dynamic-pricing-models-types-algorithms-and-best-practices/.

“ML: Reinforcement Learning Algorithm : Python Implementation Using Q-Learning.” *GeeksforGeeks*, 7 June 2019,

www.geeksforgeeks.org/ml-reinforcement-learning-algorithm-python-implementation-using-q-learning/.

“Q-Learning in Python.” *GeeksforGeeks*, 22 Dec. 2022, www.geeksforgeeks.org/q-learning-in-python/.

“Reinforcement Learning.” *GeeksforGeeks*, 18 Apr. 2023, www.geeksforgeeks.org/what-is-reinforcement-learning/