Simulating Concert Ticket Purchases For Specified Venue Capacities, Layouts, and Group Purchase Dynamics

Group 275
Alex Shropshire
Lyle Sweet

July 22, 2024

Abstract

This paper details a simulation model built in Python's SimPy library to estimate data points regarding online ticket sales for a concert. We discuss how the simulation works, and compare SimPy to Arena simulation software. Additionally, we highlight the flexibility of SimPy by coding in specific nuance to the process of online ticket sales, including accommodating different seating supplies and pricing tiers, whether or not customers wish to purchase only seats next to each other or are willing to attend with a stratified group, and cart abandonment rates. The model can easily be modified to accommodate different-sized venues, seating arrangements, and customer preferences. Based on the model output, we can recommend the number of days in advance tickets need to go on sale to reach 95% sellout for a venue. We can also suggest how to tier your seating supply between three ticket tiers: general admission, regular seating, and premium seating to maximize sales and decrease the time to sell out.

1 Background & Description of the Problem

Imagine you're responsible for ticket sales and operations for a live venue. You aim to maximize ticket revenue for events across three tiers: general admission, regular seating, and premium seating. Your team's success is defined as selling 95 percent of tickets within each tier between the event announcement date and the event date. A challenge, besides standard cart abandonment, is that some customers won't buy unless they can sit with their entire group, which can delay the venue reaching the 95 percent target.

This report aims to: (1) guide your imagined ticketing operations team through how the SimPy Python Library works, (2) walk through a discrete event simulation tutorial for 3 different venue layouts, and (3) discuss the benefits and weaknesses of applying SimPy to the discrete event simulation problem space in general and in direct comparison to the Arena simulation software.

2 Main Findings

2.1 Summarizing How The Python Library SimPy For Simulation Works

SimPy is a process-based discrete-event simulation framework in Python. It allows you to model and simulate real-world processes where events occur discretely. Here's a basic overview of how it works:

1. **Processes**: SimPy uses generator functions to define processes, which simulate the behavior of entities over time. In this example, we focus on ticket sales processes like customer purchase behavior.

2. **Environment**: The Environment class manages simulation time and orchestrates event scheduling and execution. In this example, we focus on scheduling events like customer website arrival and ticket purchase completion.

3. **Events**: Events are actions that change the state of the simulation, such as resource requests or delays. In this example, we model customers selecting a section and a group size, while our assignment

system handles group seating requests. Additionally, whether or not customers exit the system with the ticket they want is in focus.

4. **Resources**: Resources (like servers or machines) can be shared among processes, managing competition and access. In this example, we want to optimize the resource of available seating to maximize the number of customers that get the seats they want, actually attend the event, and exit the event satisfied.

2.2 Tutorial / User Guide

2.2.1 Problem Formulation

To build a successful simulation, you need to know what problem you are solving before you start building. The problem should be addressable and measurable by repeated simulations. In the case of this paper, we are solving a problem for concert venues that want to experiment with seating arrangements and ticketing tiers quickly and cheaply by simulating ticket sales to maximize ticket purchase completion in the smallest amount of time.

2.2.2 Objectives & Planning

It is good to set planning and objectives early, before you build your simulation to avoid having to redo work. Our goal is to help venue owners or their ticket operations departments plan their available ticket supply per price tier to make sure they are maximizing their sellout rate in a short amount of time by starting sales far enough in advance and adjusting the arrangement of seats across various seating options.

2.2.3 Model Building

Our model is in part an inventory management system with assignable probabilistic properties for the inputs we are modeling. Customer likelihood to purchase concert tickets is based on their flexibility of sitting with others, available inventory per price tier, the number of tickets they desire, and an average cart abandonment rate. Customer arrival is according to a configurable exponential distribution. We also modeled purchase decision time according to a normal distribution. That said, processing resources are less of a concern for this model since modern web ticketing platforms can handle large amounts of requests simultaneously.

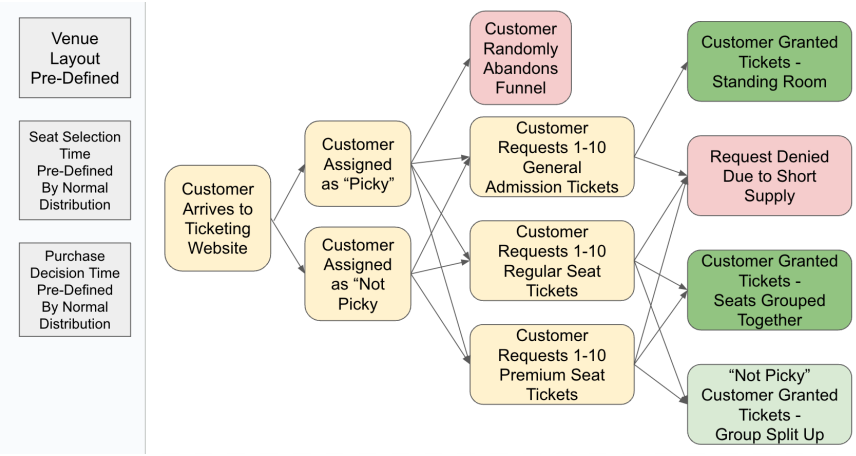

Figure 1: Model Event Map

2.2.4 Data Collection & Behavioral Assumptions

All of our distributions are assumed in our simulation model as we opted not to utilize real-world data to model our imagined venue with our imagined ticketing system which serves a hypothetical set of customer behaviors and preferences. Data generation relies on our intuitive assumptions about seat

preferences, group attendance dynamics, and purchase completion likelihood, and is generated using Python's NumPy and Random libraries. In general, if the tickets are available, our system meets your request to purchase those tickets.

There are demand-side and supply-side exceptions, however. For GA, if there are spots available for the number of requested tickets, we grant those tickets to the purchaser. For the Regular and Premium Seating tiers, we first consider if the customer is (randomly assigned at 50%, in this case) to be "picky" or "not picky", where "picky" indicates that the customer won't complete the purchase unless the entire party is assigned together. "Not picky" customers are willing to be separated within a section to accommodate the group purchase. Additionally, a percentage (a randomly assigned 20%) of "picky" customers unavoidably abandon the purchase funnel early regardless of the availability of grouped seat assignments.

2.2.5 Coding

Our simulation is dependent on the Python libraries SimPy, NumPy, Pandas, and Random. Please see the associated *.py file to run the simulation yourself. It's informative and interesting to toggle the behavioral assumptions that we're holding as default in our analysis, including:

(a) the inter-arrival time (defaulted to an exponential distribution with a mean of 5 min.),

(b) the seat selection time (defaulted to a normal distribution with a mean of 5 min.),

(c) the purchase decision time (defaulted to a normal distribution with a mean of 10 min.),

(d) the venue layout or, stated differently, the available supply of tickets in each pricing tier, and the shape (row count and seats per row) of the seated sections,

(e) the probability of customers being assigned as "picky" or "not picky" (defaulted to 50%/50%),

(f) the probability that customers choose to request General, Regular, or Premium tier tickets (defaulted to 50%/30%/20%, respectively), and

(g) the probability that picky customers randomly leave the purchase funnel before completion (defaulted to 20% probability).

In addition to the assumed behavioral distributions above, we also implemented an inventory management system to account for available ticket sales within the simulation framework. Generally, customers enter the system, they are assigned variables, and they move to their relevant ticket purchase process before being disposed.

Each customer enters the simulation in the Customer Arrival function. Customers will continue to enter until the Stop Conditions are met. The Stop Condition for this simulation requires that all three ticket tiers reach 95% sell-through. After arrival, customers move on to the Start Order Process function where they are assigned variables such as the type of customer they are and the type of tickets they are in search of. Each of those decisions are determined by a random choice with probabilities as outlined above. From the Order Process function, customers move into their relevant Order Process which uses our inventory management system.

The inventory management system is built using three-dimensional NumPy arrays that store data in sections, rows, and seats. This is labeled as "tiered_ticket_availability" in the code. All seats are initialized as zero and sold seats convert to one. Once a 'picky' customer enters the ticket purchase process they are not only shown available tickets in their desired seating type, but they are shown only seating options that would allow all 1-10 members of their party to sit beside each other on the same row. We felt this nuance was important as it closely mimicked the behavior we might see in a real online ticketing system.

After tickets are confirmed to be available, a purchase decision is made, inventory is updated or not, and the customer is disposed.

2.2.6 Experiments

To run experiments in our simulation, data points need to be collected at critical moments. The key experiment underpinning our various simulation runs related to the layout of the venue - specifically, the supply of tickets for each of our tiers as described in part (d) of the previous section - General Admission, Regular Seating, and Premium Seating. Ultimately, this type of experimentation momentarily focuses our model on a key use case related to choosing the optimal venue layout for a quick sell-through. By testing different potential layouts, decision-makers about pricing tiers can more easily and accurately choose the optimal layout based on the layout that outputs the lowest time-to-95% sellout in each tier.

For the sake of example, we will test three layouts: one layout that is 100% General Admission where availability generally leads to a purchase, one that is 100% seated and sensitive to group dynamics (regular or premium), and one that is a 50%/50% mix of both. We will test each layout over 100 iterations.

2.2.7 Output Analysis

We decided to run 100 simulations for each of the three different scenarios.

The first scenario covered a venue that only had a general admission option with no assigned seating. In Figure 2, you can see that a fairly normal distribution started to form with a mean between 11 and 12 days. There are no obvious outliers. This tells us that a ticket strategy consisting of only general admission results in fairly dependable estimates of the range of days needed to reach a 95% sell-through.

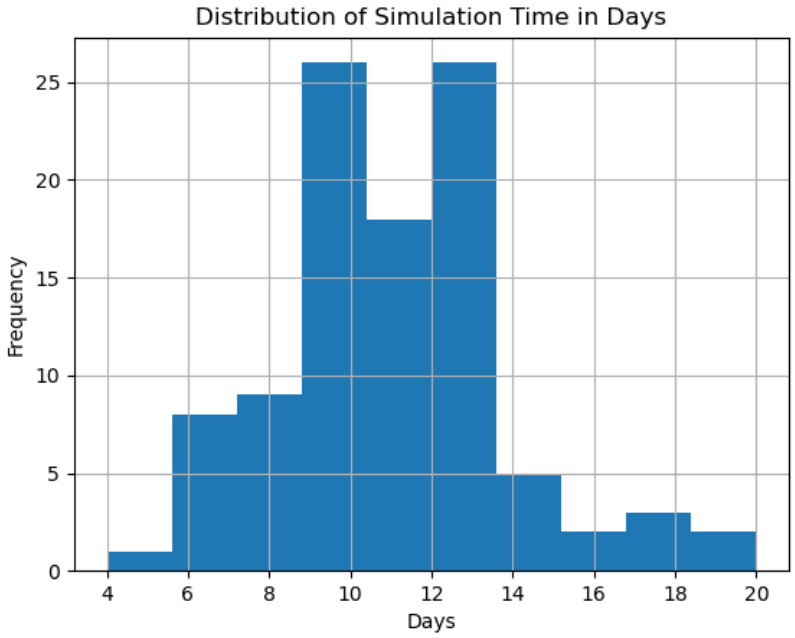

Figure 2: All General Admission Seating

Our second scenario consisted of a ticket strategy that only included seated admission. For seated admission, we allowed 50% of customers in our system to enter as 'picky' which would require that they only purchase seats if they can sit together in the same row. This constraint leads to delays in sell-through as can be seen in Figure 3, where our mean number of days to reach sell-through increases to between 22 and 23. There are additional outliers that extended beyond 40 days with this model of seating.

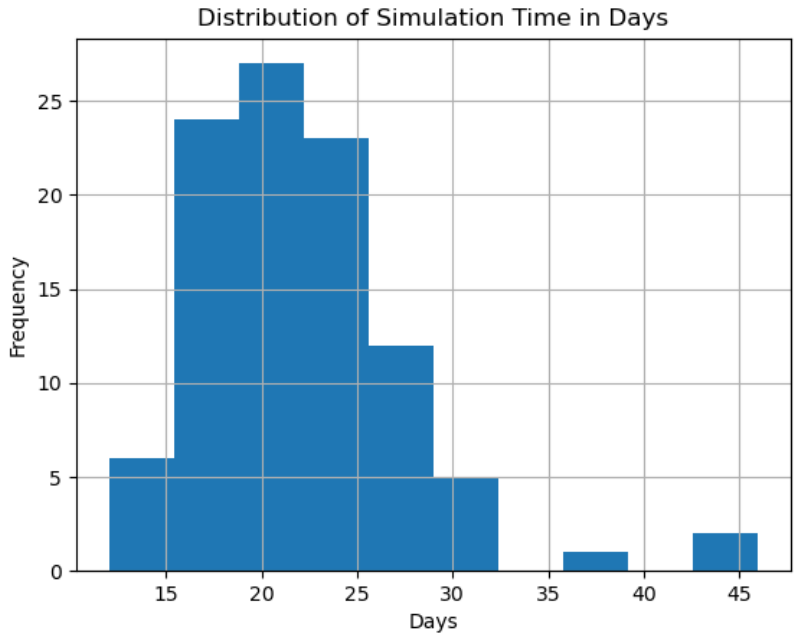

Figure 3: All Seated

In our third scenario, we ran the simulation 100 times on a hybrid seating model where half of the seats were available in general admission and half of the capacity was allocated to seats. While this might seem like the best of both worlds, in reality, we saw that since there are a finite number of customers entering the system, more customers were required to fill the seated sections as many were placed in general admission. This led to even more skewed results as can be seen in Figure 4. The mean in Figure 4 is between 30 and 31 days and some of the outliers reached past 80 days.

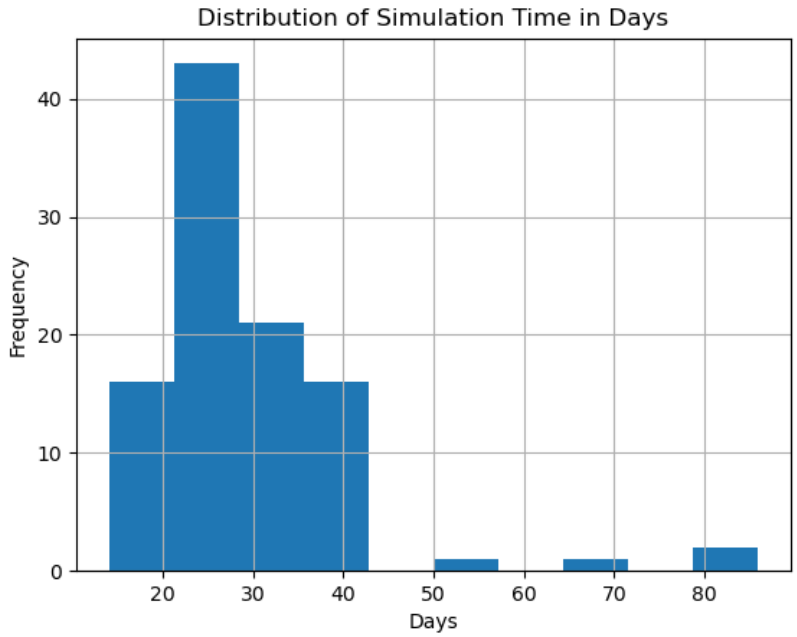

Figure 4: 50% GA / 50% Seated

Ultimately, if speed-to-sellout was the only performance metric needed to decide the optimal venue layout, our simulation might indicate that selling all seats as General Admission would be the obvious

choice. That said, in reality, a venue might want to optimize for total revenue which would make the assumed higher-priced regular and premium seating more attractive.

Additionally, selling all tickets as an assigned seat (regular or premium) roughly doubles the speed-to-sellout metric. Because a much higher % of the venue is seated, a much higher % of our inventory is exposed to 'picky' customers not completing the purchase due to a lack of available seating for their entire party - a situation that does not exist in the 100% GA scenario. More total arrivals are therefore needed to reach the 95% sellout finish line, leading to a higher average number of days to sell out in our simulation than the 100% GA example.

Lastly, the slowest path to sell out comes via a 50/50 mix between GA and seated tickets. First, the GA section is smaller, so it fills up quicker than the 100% GA scenario. Meanwhile, once GA is full in the 50/50 simulation, many customers ($\tilde{5}0\%$) continue to arrive interested in GA tickets. We cannot fulfill those requests, and so require many more arrivals and much more time on aggregate to reach 95%. The additional bottleneck here can be attributed again to the difficulty in filling in the seated sections with 'picky' customer groups. Yet, in this scenario the seated inventory here is lower, and reaching the 95% stop criteria in, for example, the premium section of 312 total seats might necessitate the arrival of a small group count for picky customers in the later sales stages to finally cross the threshold.

Imagine a scenario where sell-through on premium seats is near $\tilde{9}4\%$, while new arrivals happen to skew to large, 'picky' groups, while the largest seats-together inventory is a stretch of 3 seats. We'd likely have to wait longer for a "sellable" customer arrival - in this case, a 'picky' group size of 3 or less, or a 'not picky' group size of less than the total remaining seats. This potential bottleneck is much more likely to occur in the 50/50 GA/seated scenario and, in general, helps explain the presence of the super-slow simulations in Figures 3 and 4.

2.3 Pros and Cons of SimPy

The Good:

1. *Ease of Use*: SimPy's API is straightforward [LMSV13], making it accessible for beginners, but especially for those familiar with Python. This simplicity comes from its clear syntax and integration with Python's native features. It allows users to quickly build and run simulations without having to dive into complex programming.

2. *Flexibility*: Can model a wide range of processes and systems, from simple queues to complex systems, making it a very versatile option.

3. *Integration*: Easily integrates with other Python libraries for data analysis and visualization, like Pandas, NumPy, and Matplotlib.

4. *Open Source and Active Community*: Free to use and has a supportive community. Has a plethora of Medium.com walk-throughs [AM24] and examples, and deep StackOverflow common question documentation. These are invaluable for troubleshooting issues and finding best practices.

The Not-So-Good:

1. *Performance Issues*: SimPy may not be as computationally performant as some specialized simulation engines, particularly for very large-scale simulations. Python's interpreted nature and SimPy's abstraction layers can lead to slower execution times compared to compiled simulation tools.

2. *Limited Built-in Features*: While SimPy is flexible, it does not come with as many built-in features or advanced modeling constructs as some other simulation frameworks. You may need to implement complex features or behaviors yourself, which can be time-consuming and requires more effort. Additionally, SimPy's error messaging capabilities are poor, making debugging often time-consuming.

3. *Scalability Challenges*: For highly complex or large-scale simulations, SimPy can face challenges related to scalability and may require significant optimization. The design and architecture of SimPy might not handle very large numbers of simultaneous events or entities as efficiently as other tools.

4. *Limited GUI Support*: Despite helpful integrations, SimPy does not offer built-in support for graphical user interfaces (GUIs) or visual simulation tools, which might be a drawback for users needing interactive or visual components. The absence of GUI features means additional work is needed to visualize and interact with the simulation results, potentially requiring third-party tools or custom solutions.

2.4 Quick Comparison: SimPy vs. Arena

SimPy and Arena serve different needs within the world of discrete-event simulation, each reflecting its unique design and intended user base. SimPy is a free, open-source library for Python, aimed at users who are comfortable with Python programming. Its integration with Python's extensive ecosystem offers high flexibility, which is beneficial for research and custom simulations. However, SimPy may face performance issues with very large or complex models due to the inherent limitations of Python and its abstraction layers. Although it is budget-friendly, users may need to rely on other Python libraries to extend its functionality, which adds to the development effort.

On the other hand, Arena [IKZ24] is a commercial simulation software developed by Rockwell Automation, featuring a user-friendly graphical interface and a comprehensive set of built-in tools. It is designed to handle complex and large-scale simulations efficiently, making it a strong choice for industrial and professional applications. Arena's robust capabilities come with a significant cost and are tailored for users needing advanced features and support. While it provides an integrated environment with professional assistance, the high cost and less flexibility for bespoke use cases might not be ideal for those seeking a more adaptable or economical option like SimPy.

3 Conclusion & Future Work

The results from our simulation output are very promising. We can observe the resulting summary statistics and overall distributions from our venue layout experiments and gain an intuition as to how various seating arrangements can affect the speed at which an event sells out to 95% of inventory. Notably, the key drag on time-to-sellout comes via the 'picky' customers who want to sit together in seated inventory scenarios that have sold out of their large group space. As ticket demand is often limited to a specific artist, removing barriers to entry for anyone interested in an event seems to be an effective way to sell out an event. Without large groups with seating constraints, this is likely to happen faster for venues.

Ultimately, our simulation has its limits as it (1) only enriches a small portion of the decisions a concert venue has to make to optimize their ticket operations, and (2) relies on a variety of intuitive assumptions about customers that could be verified using historical data. For example, the decision for an optimal venue layout in the real world might prefer to focus on the maximization of revenue instead of speed-to-95% sell-through. That approach could benefit from simulating through a variety of tiered pricing schemes. This extra layer of analysis is not simulated here, yet we believe the technology we chose to rely on here can scale to that alternative reality effectively. Additionally, there's an opportunity to seek more nuance, confidence, and accuracy in our underlying input assumptions around arrival times, seat selection times, purchase decision times, and customer preferences using real-world historical data.

We'd focus our future work on these two areas to help us meet the challenge of a more nuanced economic reality of venue ticketing operations. Yet, from what we've experienced in our limited example, we are confident that SimPy scales well to such a challenge. SimPy, for great reasons, asserts itself both to us and a broader world of predictive analysts as an efficient, simple-to-use, well-documented, flexible Python module for discrete event simulation that can be called upon for a wide variety of use cases, environments, and industry settings.

References

[AM24] Lazuardi Al-Muzaki. Discrete event simulation using python simpy — building basic model: Simulating coffee and pizza eatery, Published: 2023; Last Accessed: 21 July 2024. https://medium.com/lazuardy.almuzaki/discrete-event-simulation-using-python-simpy-building-basic-model-1bb34b691797.

[IKZ24] Nathan Ivey, David Kelton, and Nancy Zupick. *Simulation with Arena, 7th Edition.* Number ISBN10: 1264162448 — ISBN13: 9781264162444. McGraw Hill, 2024.

[LMSV13] Ontje Lünsdorf, Klaus Müller, Stefa Scherfke, and Tony Vignaux. Simpy: Official documentation. *Simpy.Readthedocs.io*, 1(1):Overview, Examples, API Reference, 2013.