The University of Texas at Austin
Texas McCombs
MS Business Analytics
McCombs School of Business

Optimization Project 2

Dynamic Programming for Airline Ticket Sales

Group Members:

Anjali Pillai (ap66745)

Sarah Stephens (sgs2623)

Tara Mary Joseph (tj9763)

Téa McCormack (tsm2423)

Professor:
Dan Mitchell

Spring 2025

Table of Contents

Overview

Background

Airlines constantly face the tricky decision of whether or not to overbook flights. While no-shows can result in empty seats and lost revenue, overbooking too aggressively can lead to costly compensation payouts and frustrated customers. To strike the right balance and maximize expected discounted profits, pricing teams need to carefully determine both the ticket price and how many tickets to sell for a given flight.

Objective

Our goal is to use dynamic programming to identify the optimal pricing policy across different overbooking strategies. This approach breaks the problem into smaller steps, solves them in reverse order, and then simulates outcomes going forward. By applying this method, we aim to find and explain the overbooking strategy that leads to the highest profit.

Algorithm Methodology

Defining Parameters

First, we define all constants, including the number of days until departure, coach seating capacity, probability of showing up, first-class seating capacity, cost of upgrading, cost of bumping off, discount rate, and the current overbooking limit (initially set to 5 seats). All prices and probabilities of booking for each tier are defined in a dictionary. Interestingly, using an array yields a slightly lower profit (about $20 less). Technically, both of these strategies are correct, but we opted for the method that results in the higher profit which is dictionary-based.

The following algorithms assume the following:

- ➢ Coach
    - High price is \$350 with 30% chance of selling
        - Increases to 33% when first-class is sold out
    - Low price is \$300 with 65% chance of selling
        - Increases to 68% when first-class is sold out
    - 100 seats
    - Probability of showing up: 95%
- ➢ First-class
    - High price is \$425 with 8% chance of selling
    - Low price is \$500 with 4% chance of selling
    - 20 seats
    - Probability of showing up: 97%
- ➢ 365 days before take-off
- ➢ Cost of upgrading to first-class is \$50
- ➢ Cost of bumping passenger off the plane is \$425
- ➢ Discount rate is 17% per year

These parameters are defined as follows:

```
T = 365
COACH_SEATS = 100
FIRST_CLASS_SEATS = 20
OVERBOOK_LIMIT = 5
MAX_COACH = COACH_SEATS + OVERBOOK_LIMIT

# Ticket prices & base demand probabilities
coach_prices = {300: 0.65, 350: 0.30}
fc_prices = {425: 0.08, 500: 0.04}

# Show-up probabilities
p_show_coach = 0.95
p_show_fc = 0.97

# Overbooking costs (first-class & off plane)
bump_to_fc_cost = 50
bump_off_cost = 425

# Discounting (17% annual converted to daily)
daily_discount = 1 / (1 + 0.17 / 365)
```

Calculating Overbooking Costs

Before we start the dynamic programming part, we first need to calculate the expected cost of overbooking on the day of the flight. To do this, we generate data that shows what the cost would be for every possible combination of coach and first-class tickets sold. The result is a dataframe that maps out these expected costs for all final-day scenarios.

We use a function called "terminal_value" to estimate these overbooking costs based on how many tickets were sold in each class. Since no more tickets are being sold on the day of the flight, there's no revenue—only potential costs if more passengers show up than we have seats for. This focus on the final day and working backward from there is a core idea in dynamic programming. To make these estimates, we model the number of passengers who show up using a binomial distribution. We explored two ways to code this distribution—the first method offers a more detailed, step-by-step view that highlights the theory behind it more clearly.

$$X \sim \text{Binomial}(n, p)$$
$$n = \text{number of passengers}$$
$$p = \text{probability of showing up}$$

From this binomial distribution, we can calculate the mean and standard deviation. The mean is calculated as the number of passengers (n) multiplied by the probability of showing up (p). The standard deviation is the square root of variance, which simplifies to $n * p * (1 - p)$.

```
   coach_mean, coach_std = coach_sold * coach_show_prob,
np.sqrt(coach_sold * coach_show_prob * (1 - coach_show_prob))

   first_mean, first_std = first_class_sold * first_class_show_prob,
np.sqrt(first_class_sold * first_class_show_prob * (1 -
first_class_show_prob))
```

Then, the range of possible values for the number of passengers that show up is calculated by subtracting and adding three standard deviations from/to the mean. Using three standard deviations is customary practice when dealing with binomial distributions, and this strategy preserves almost 100% of the data while still ignoring the most extreme outcomes.

```
    coach_range = range(max(0, int(coach_mean - 3 * coach_std)),
min(coach_sold + 1, int(coach_mean + 3 * coach_std) + 1))
   first_range = range(max(0, int(first_mean - 3 * first_std)),
min(first_class_sold + 1, int(first_mean + 3 * first_std) + 1))
```

The scipy.stats.binom.pmf function from the scipy.stats package computed the probability mass function of the binomial distribution given the range defined in the previous step.

```
  coach_pmf = stats.binom.pmf(coach_range, coach_sold, coach_show_prob)
   first_pmf = stats.binom.pmf(first_range, first_class_sold,
first_class_show_prob)
```

The steps outlined above require more code that is not inherently necessary to solving the optimization problem. The following code shows a more efficient and straightforward way of finding the probability mass function of a binomial distribution.

Once we define the "terminal_value" function and its inputs, we start by initializing the expected cost to zero. We then loop through all possible values of *x*, representing how many coach passengers actually show up at the gate. By looping from 0 to the total number of coach tickets sold (inclusive), we're covering every possible turnout scenario.

To calculate the likelihood of each scenario, we use the binom.pmf function, which gives us the probability that exactly *x* passengers show up, based on a binomial distribution. This depends on the number of tickets sold and the probability that each passenger shows up. We repeat the same process for first-class passengers, looping through every possible number of first-class show-ups (*y*) and calculating their probabilities.

```
def terminal_value(coach_sold, fc_sold):
    expected_cost = 0.0
    for x in range(coach_sold + 1):
        p_x = binom.pmf(x, coach_sold, p_show_coach)
        for y in range(fc_sold + 1):
            p_y = binom.pmf(y, fc_sold, p_show_fc)
```

Next, we look at the cost associated with each combination of coach and first-class passengers showing up on the day of the flight. If the number of coach passengers who show up is within the available seats, there's no overbooking and the cost is zero. But if more coach passengers show up than there are seats, we calculate the "extra" passengers as the difference between show-ups and coach capacity.

At that point, we check how many first-class seats are still available (i.e., the difference between total first-class seats and how many first-class passengers showed up—capped at zero if it's negative). If there are open seats, the airline can upgrade some coach passengers to first class. The number of upgrades is the smaller of the extra coach passengers and the number of available first-class seats. Any remaining extra passengers who couldn't be upgraded are bumped from the flight.

The total overbooking cost is then calculated by combining the cost of upgrades and the cost of bumping passengers off entirely, all weighted by the joint probability that *x* coach and *y* first-class passengers show up.

```
            if x <= COACH_SEATS:
                cost = 0
            else:
                extra = x - COACH_SEATS
                available_fc = max(0, FIRST_CLASS_SEATS - y)
                bumped = min(extra, available_fc)
                bumped_off = extra - bumped
                cost = bumped * bump_to_fc_cost + bumped_off *
bump_off_cost
            expected_cost += p_x * p_y * cost

    return -expected_cost
```

The function returns the negative expected cost which serves as the terminal value in the dynamic programming model. Since costs reduce profit, they must be negative, leading the model to interpret high costs as undesirable and low costs as beneficial. This allows the DP algorithm to maximize total profit by effectively minimizing overbooking costs in its optimization process.

Determining Dynamic Sale Probabilities

After building the “terminal_value” function, we added two helper functions to determine the daily probability of selling coach and first-class tickets. These probabilities depend on current availability and are adjusted dynamically.

For coach tickets, if first-class tickets are already sold out, demand increases slightly—by 3%—which we factor into the recalculated sale probability. This reflects a realistic behavior: when premium options are unavailable, some customers shift to coach.

For first-class tickets, we ensure that sales stop once the cabin is full. To maintain customer satisfaction and avoid service issues, we don’t allow overbooking in first class—the number of tickets sold can never exceed capacity. So if first-class is full or coach is already overbooked, the probability of selling a first-class ticket drops to zero. Otherwise, if both classes still have room, the base sale probability for first-class remains unchanged.

```python
def get_coach_sale_prob(fc_sold, price):

   base = coach_prices[price]

   if fc_sold >= FIRST_CLASS_SEATS:

       return min(base + 0.03, 1.0)

   return base

def get_fc_sale_prob(coach_sold, fc_sold, price):

   if fc_sold >= FIRST_CLASS_SEATS or coach_sold >= MAX_COACH:

       return 0.0

   return fc_prices[price]
```

Overbooking Strategy 1: Hard Cap

With the base case in place—including expected overbooking costs, passenger show-up probabilities, and ticket sale probabilities—we're ready to simulate our first policy: capping overbooking in coach at 5 seats. To track everything, we use a dictionary that stores the maximum expected profit for every possible state, defined by three variables:

- ➢ t: number of days remaining until the flight
- ➢ c: number of coach tickets sold
- ➢ f: number of first-class tickets sold

We start by looping through all possible ticket sale combinations. The outer loop covers coach ticket sales, ranging from 0 up to the overbooking limit (coach capacity + 5 extra seats). The inner loop covers first-class ticket sales, which are limited strictly to the number of available seats—since overbooking isn't allowed in that tier.

Before we simulate daily decisions and revenue, we first set up the base case: the expected cost of overbooking on flight day. This provides the foundation for our dynamic programming model, which works backwards from the day of the flight to determine the optimal ticket-selling strategy.

```
dp = {}

for c in range(MAX_COACH + 1):

    for f in range(FIRST_CLASS_SEATS + 1):

        dp[(0, c, f)] = terminal_value(c, f)
```

Dynamic programming follows a backwards recursion, with *t* representing the number of days until departure. The following for-loop demonstrates the movement backwards in time, from day 1 (1 day prior to departure) to day 365 (a year prior to departure). The next two for-loops pass through all possible numbers of coach and first-class tickets that are already sold.

```
for t in range(1, T + 1):
    if t % 50 == 0:
        print(f"Processing Day {t} of {T}...")
    for c in range(MAX_COACH + 1):
        for f in range(FIRST_CLASS_SEATS + 1):
            best_value = -np.inf
```

For each day (*t*-value) the options for ticket prices must be optimized, as they affect the probability of sale. This for-loop tries all possible combinations of coach and first-class prices and retrieves the adjusted sale probabilities using the previously defined functions.

```
              for coach_price, _ in coach_prices.items():

                  for fc_price, _ in fc_prices.items():

                      p_coach = get_coach_sale_prob(f, coach_price) if c <
MAX_COACH else 0.0

                      p_fc = get_fc_sale_prob(c, f, fc_price) if f <
FIRST_CLASS_SEATS else 0.0
```

Next, we must calculate the probabilities for all four possible sale outcomes—no sale, coach only, first-class only, or both. All of the probabilities are independent of each other, so the formulas are products of individual probabilities.

```
                    no_sale = (1 - p_coach) * (1 - p_fc)
                    coach_only = p_coach * (1 - p_fc)
                    fc_only = (1 - p_coach) * p_fc
                    both = p_coach * p_fc
```

After calculating the probabilities of each possible sale outcome, we can calculate the corresponding revenues. These are straightforward definitions that use the base prices to calculate revenue for each outcome. Immediate revenue is the product of revenue and number of sales for each category, summed together across all types of sales.

```
                        immediate_revenue = (
                            no_sale * 0 +
                            coach_only * base_coach_price +
                            fc_only * base_fc_price +
                            both * (base_coach_price + base_fc_price))
```

The last step in this series of for loops is to determine the expected future value of all sales. The state (time, coach, first-class) changes differently depending on what type of sale occurs. At each step, time decreases by one day. If there is no sale, the state remains unchanged. If there is a coach/first-class sale, the number of coach/first-class tickets sold increases by one, assuming capacity constraints are satisfied.

```
                        future_value = 0
                        future_value += no_sale * dp[(t - 1, c, f)]
                        if c < max_coach:
                            future_value += coach_only * dp[(t - 1, c + 1,
f)]
                        else:
                            future_value += coach_only * dp[(t - 1, c, f)]
                        if f < max_fc:
                            future_value += fc_only * dp[(t - 1, c, f + 1)]
                        else:
                            future_value += fc_only * dp[(t - 1, c, f)]
                        if c < max_coach and f < max_fc:
                            future_value += both * dp[(t - 1, c + 1, f +
1)]
                        else:
                            future_value += both * dp[(t - 1, c, f)]
```

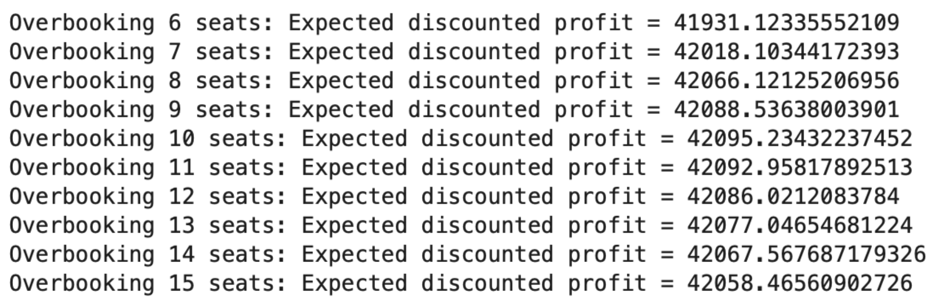
The total value is defined as the immediate revenue plus discounted future profit, and we update the best value throughout each iteration based on the total values obtained.

```
                        total_value = immediate_revenue + daily_discount *
future_value
                        best_value = max(best_value, total_value)
                dp[(t, c, f)] = best_value
    return dp[(DAYS, 0, 0)]
```

When there are 365 days until takeoff, 100 coach seats, 20 first-class seats, and an overbooking limit of 5, the optimal expected profit is **$41,792.48**. When cross checked with the data frame that lists all possible days remaining, coach tickets sold, first-class tickets sold, and expected profit, the result is confirmed.

Knowing that the result is correct for an overbooking limit of 5 days, we can then repeat the same process for the cases when the overbooking cap is 6 through 15 seats. The results are as follows, with the best overbooking policy being a cap of 10 seats, which yields a profit of **$42,095.23**.

```
for overbook in range(6, 16):
   profit = run_dp_for_overbooking_limit(overbook)
   results[overbook] = profit
```



```
Overbooking 6 seats: Expected discounted profit = 41931.12335552109
Overbooking 7 seats: Expected discounted profit = 42018.10344172393
Overbooking 8 seats: Expected discounted profit = 42066.12125206956
Overbooking 9 seats: Expected discounted profit = 42088.53638003901
Overbooking 10 seats: Expected discounted profit = 42095.23432237452
Overbooking 11 seats: Expected discounted profit = 42092.95817892513
Overbooking 12 seats: Expected discounted profit = 42086.0212083784
Overbooking 13 seats: Expected discounted profit = 42077.04654681224
Overbooking 14 seats: Expected discounted profit = 42067.567687179326
Overbooking 15 seats: Expected discounted profit = 42058.46560902726
```

Overbooking Strategy 2: No Sale Choice

The second policy allows a third choice on each day when you're deciding ticket prices—the airline can choose to sell no coach tickets, forcing demand to zero. Thus, the three choices for coach are high price, low price, and no sale, and two choices (high and low price) remain for first-class. We initially assume that the airline will never sell more than 120 coach tickets (max_coach = 120) and the maximum number of first-class seats is still 20 (max_fc = 20). In the previous policy, there were only four possible pricing decisions:

```
coach_prices = {300: 0.65, 350: 0.30}
fc_prices = {425: 0.08, 500: 0.04}
```

```
            for coach_price, _ in coach_prices.items():
                for fc_price, _ in fc_prices.items():
```

In this new strategy, the coding logic stays the same, but there are now five possible pricing decisions. Each action is coded as a tuple.

```
                coach_actions = [
                    ("low", 300, 0.65),
                    ("high", 350, 0.30),
                    ("no_sale", 0, 0.0)]

                fc_actions = [
                    ("low", 425, 0.08),
                    ("high", 500, 0.04)]
```

We then loop through all possible coach and first-class actions and determine the sale probabilities. This process is the same logic as before, but instead of creating functions for calculating each probability, we coded it directly into the main function. Additionally, we incorporated an if/else statement that forces the coach sale probability to zero if there is no sale.

```
                        if coach_label == "no_sale":
                            p_coach = 0.0
                        else:
                            if c < max_coach:
                                p_coach = base_coach_prob
                                if f >= FIRST_CLASS_SEATS:
                                    p_coach = min(p_coach + 0.03, 1.0)
                            else:
                                p_coach = 0.0
```

Since the revenue when there is no sale is zero, nothing needs to be added to the revenue portion of the algorithm. The code logic for future value calculations also remains the same. The result of optimizing the policy for a no-sale option—with a hard cap of 120 coach seats—is a profit of **$42,140.53** This result yields a higher profit by $45.30, as compared to the previous strategy that oversells by 10 seats but does not allow for a no sale option on any day.

Overbooking Strategy 3: Seasonality

The third policy assumes that as the time of departure gets closer, the demand for tickets increases, representing a seasonality element. On each day, we multiply the base probability for each ticket class by a factor (0.75 + t/730). At the beginning of the backward recursion loop in the dynamic programming algorithm, we add a booking day index, where d is the booking day number with day 1 being farthest from departure and day 365 being the day of departure.

```
for t in range(1, DAYS + 1):
        selling_day = DAYS - t
        seasonality_factor = 0.75 + selling_day / 730
```

In this policy, we have not included a no sale possibility. Thus, we have four price combinations for coach and first-class. It is essential to adjust the probabilities to reflect this new seasonality factor, which is done by multiplying the base probability for each tier by the seasonality factor. If first-class is sold, then a fixed bonus of 0.03 is added to the updated coach probability, and the probability is capped at 100%. All probabilities are set to zero if the maximum number of seats (including the overbooking limit) is reached.

All of these steps are incorporated into the main loop that passes through all the possible days, increasing the demand for tickets as each day before departure passes.

```
                       adjusted_coach_prob = base_coach_prob *
seasonality_factor
                       if f >= FIRST_CLASS_SEATS:
                           adjusted_coach_prob = min(adjusted_coach_prob +
0.03, 1.0)

                       if c >= max_coach:
                           adjusted_coach_prob = 0.0

                       adjusted_fc_prob = base_fc_prob *
seasonality_factor
                       if f >= max_fc or c >= max_coach:
                           adjusted_fc_prob = 0.0
```

The remainder of the algorithm remains unchanged. By incorporating seasonality—without adding the “no sale” option and assuming an overbooking limit of 10 coach seats—the model yields an expected profit of **$41,795.17**.

We also explored the combined effect of introducing the “no sale” option alongside seasonality. Following the approach from Strategy 2, we added “no sale” as a third option in the coach pricing decision, assigning zero revenue and zero probability of sale for that action. Additionally, instead of assuming that an overbooking limit of 10 was optimal, we adopted the approach from Strategy 1 and looped through all possible overbooking limits from 5 to 15 to identify the optimal cap. Overbooking by 10 seats remained the optimal policy. The result of incorporating the “no sale” option, accounting for seasonality, and optimizing the overbooking limit was a profit of **$41,841.73**.

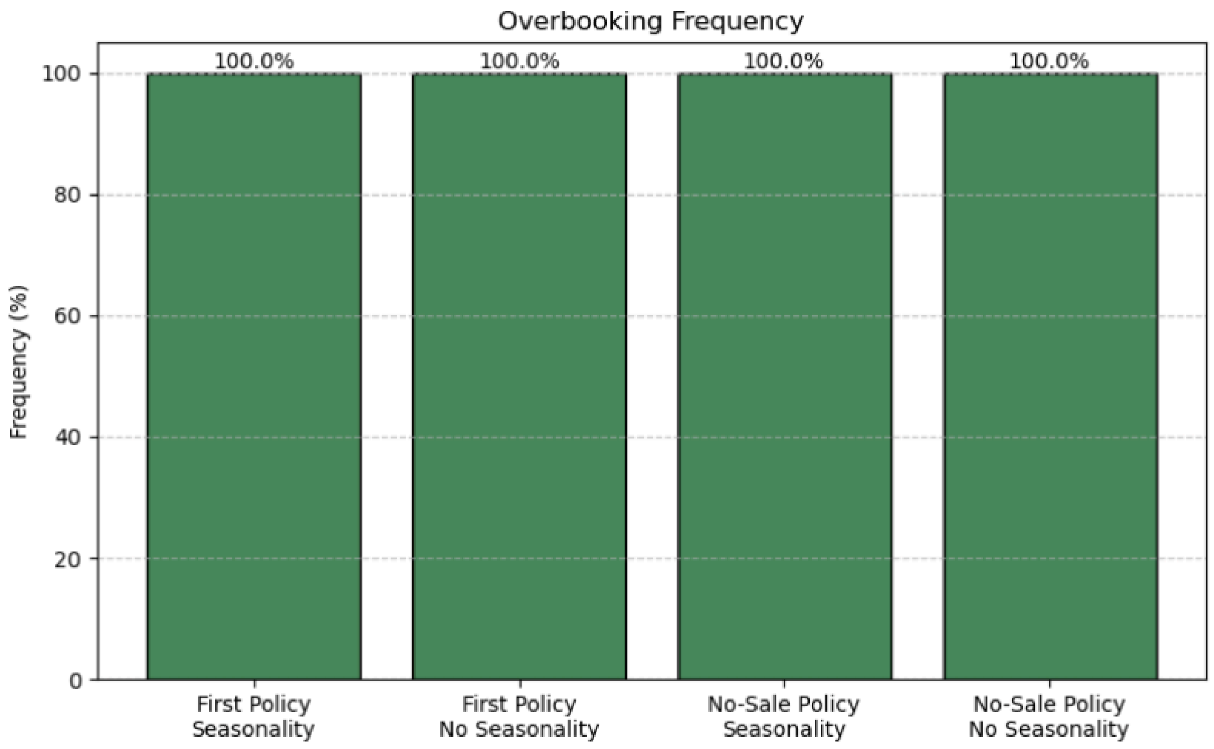
Results

Findings

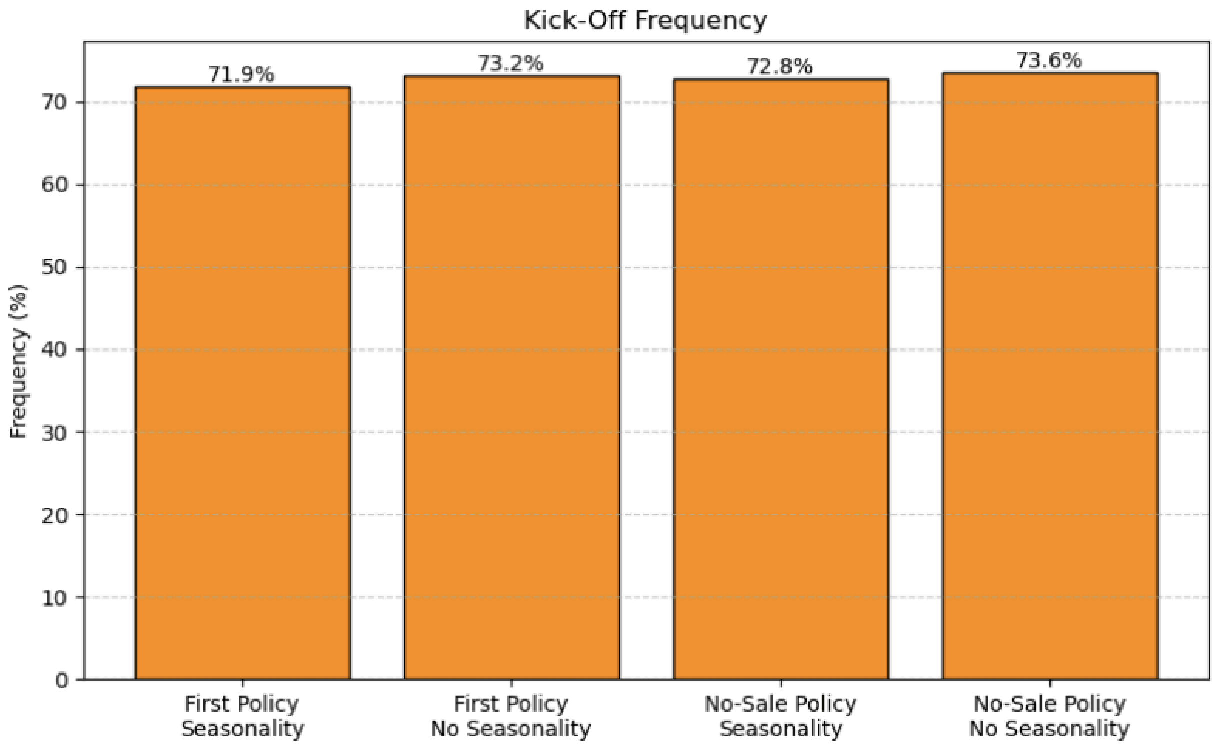
	Strategy 1b: Best Overbooking No Seasonality	**Strategy 2:** Flexible No Sale No Seasonality	**Strategy 3a:** Best Overbooking Seasonality	**Strategy 3b:** Flexible No Sale Seasonality
Overbooking Limit	10	20	10	10
Coach Overbooking Frequency	100.0%	100.0%	100.0%	100.0%
Passenger Kick Off Frequency	73.2%	73.6%	71.9%	72.8%
Average Overbooking Cost	$1,159.75	$1,255.38	$1,112.67	$1,254.96
Profit Volatility	$923.64	$927.31	$892.42	$962.96
Expected Discounted Profit	$42,095.23	$41,140.53	$41,795.17	$41,841.73
Average Discounted Profit	$41,614.76	$41,610.98	$41,949.90	$41,932.71

Visualization Analyses

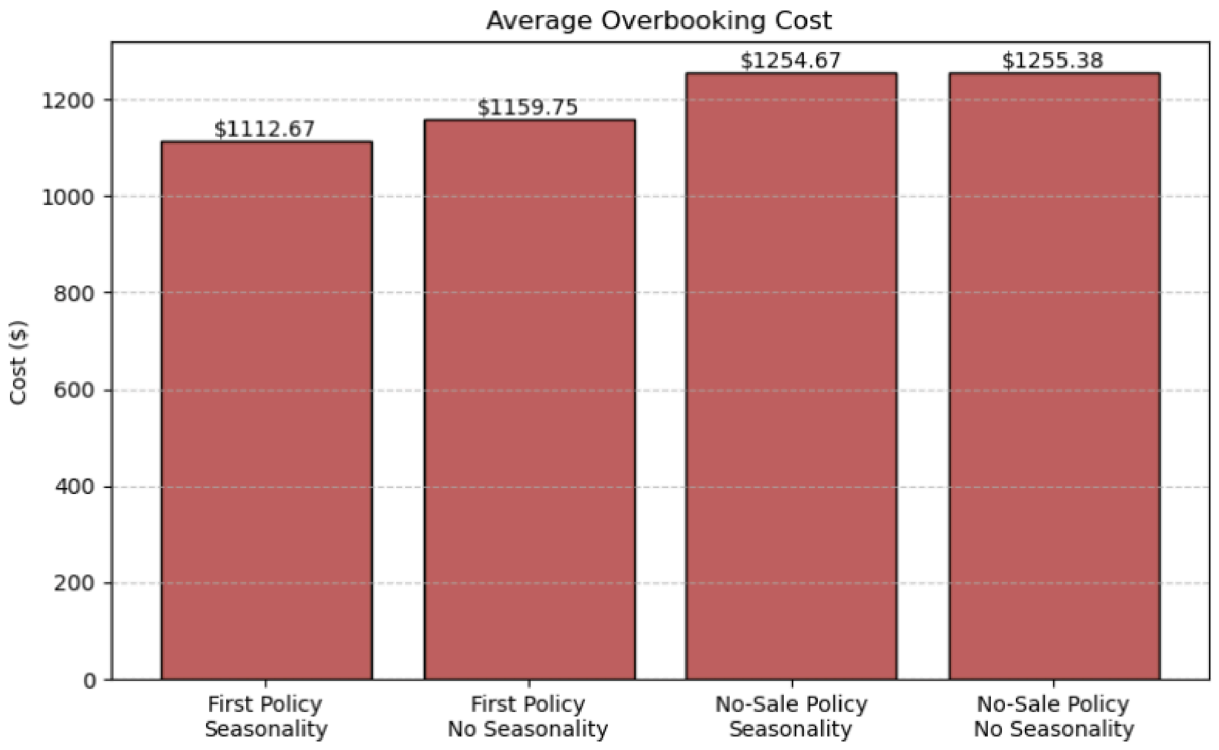
In order to create visualizations that would allow for easy comparison between the different strategies, simulations were used to mimic one year of flight sales. These simulations considered seasonality, coach and first class seating availability, overbooking limits, passenger no show probabilities, and revenue and costs. For each of the four scenarios, performance metrics were computed, including averaging profit, overbooking frequency, passenger kick off frequency, and profit volatility. The results were visualized through histograms and various plots for easy comparison, and key metrics were displayed.



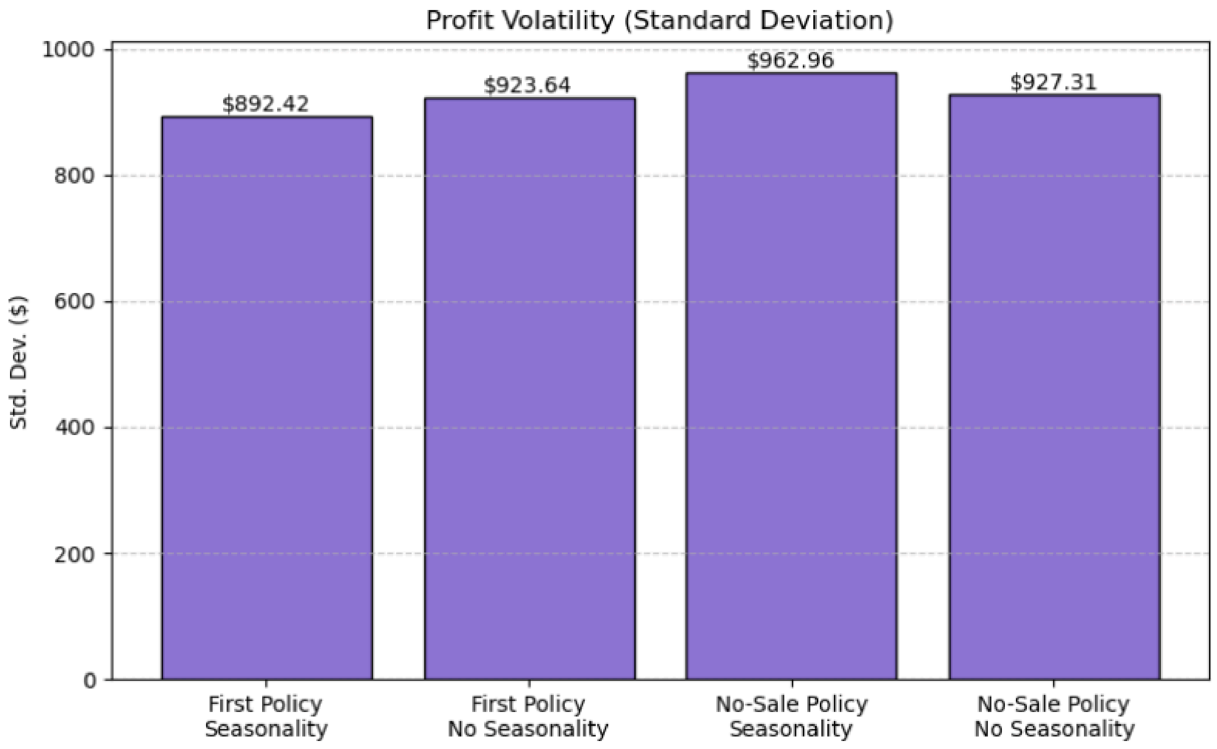
Graph 1. The above graph displays the coach overbooking frequency for each of the strategies. In all four of the strategies tested, the frequency of coach overbooking is 100%.



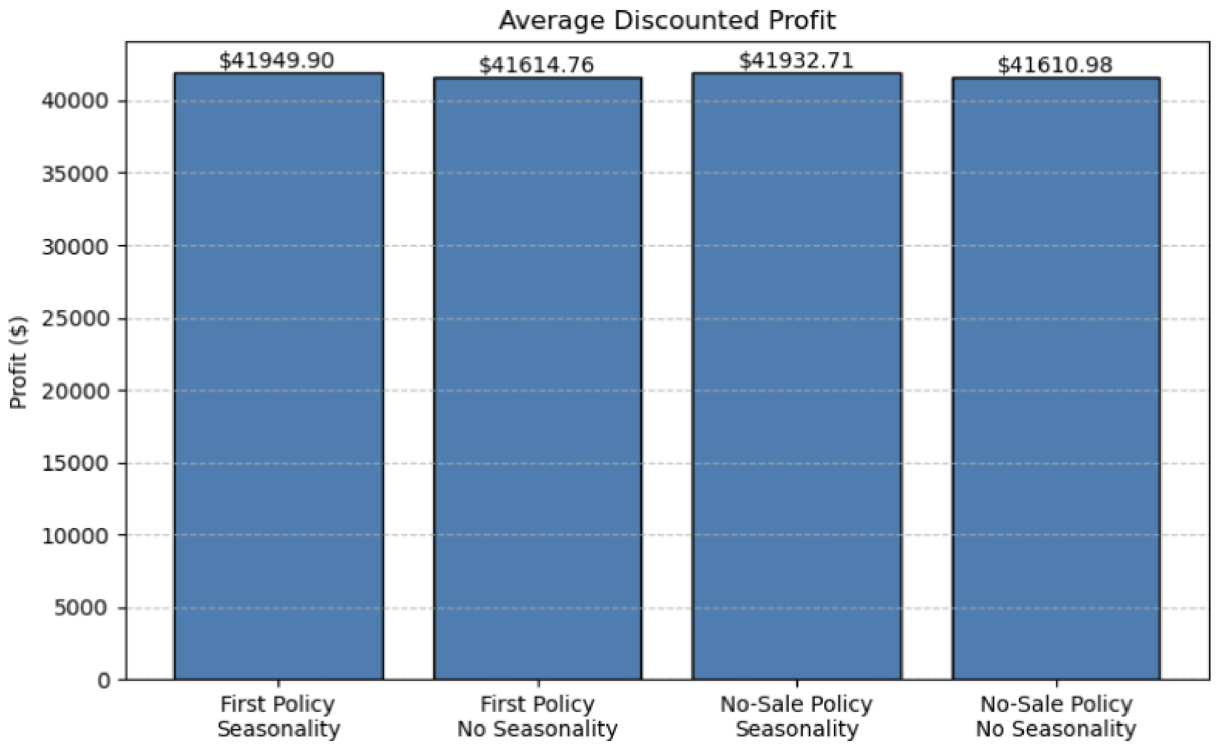
Graph 2: The graph above shows the passenger kick off frequency. In all four of the strategies tested, the kick off frequency is relatively the same. The strategy with the highest kick off frequency is flexible no sale without seasonality at 73.6%. On the other hand, the first policy with seasonality has the lowest kick off frequency.



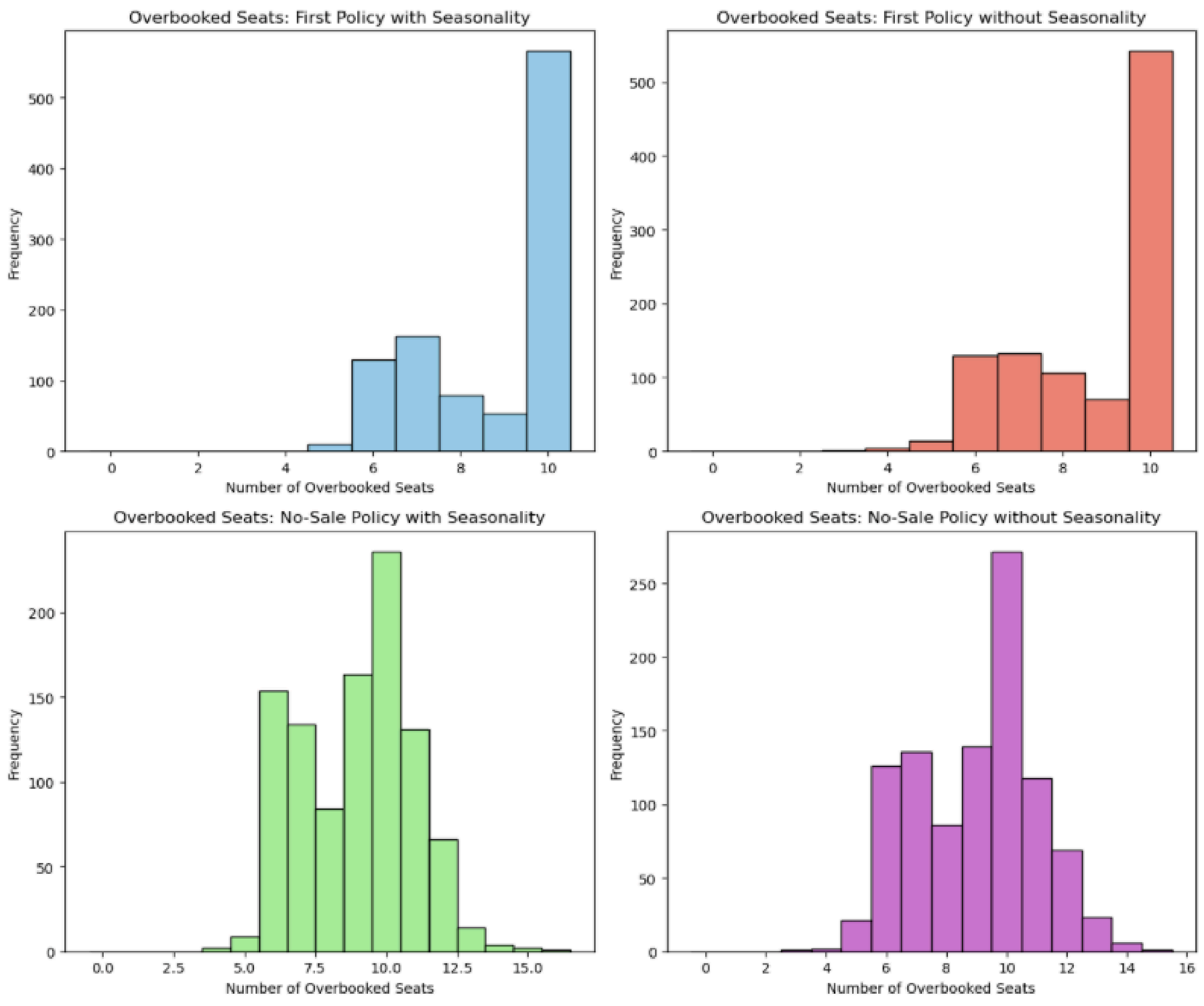
Graph 3: The graph of average overbooking cost shows that—in terms of reducing cost—there is a benefit to using the strategies that do not allow a no sale option. When using the optimal overbooking limit of 10 seats, the average overbooking cost is lower both with and without seasonality compared to the flexible no sale strategies. The average overbooking cost continues to increase in the no sale policy with seasonality, and the no sale policy without seasonality. Using best overbooking (first policy) strategies can reduce the additional cost that an airline incurs when they overbook passengers. Costs arise from compensating passengers for the inconvenience, rebooking them on a more expensive flight, or upgrading them to first-class.



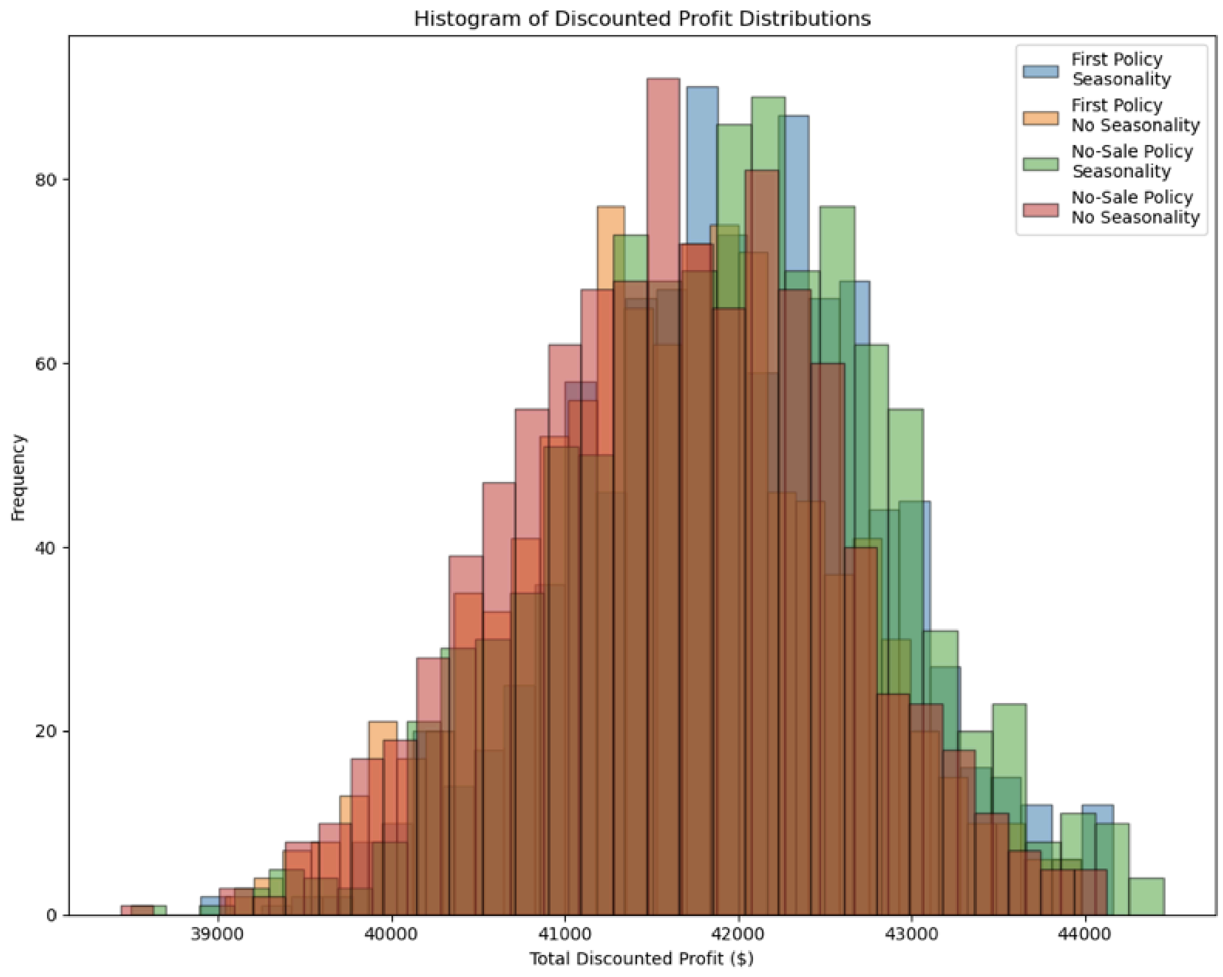
Graph 4: In profit volatility, utilizing the first policy (no "no-sale" option) with seasonality results in the lowest value of $892.42. Using the first policy or no-sale policy without seasonality results in similar profit volatility. The no-sale policy with seasonality has the highest profit volatility at $962.96.



Graph 5: The average discounted profit for all of the simulations are roughly similar. Since the values are roughly the same, average discounted profit may not be a metric that is extremely important to the airline to focus on.



Graph 6: The four plots above show the distribution of overbooked seats for each of the four strategies. While the distributions are not the exact same, in the top two policy types, 10 overbooked seats has the highest frequency, and there is a similar distribution overall. The same pattern can be found in the no-sale policy graphs; however, there is a wider range of overbooked seats possible for these strategies, reflecting the flexibility offered by the "no-sale" option.



Overall comparison of distribution of the 4 different policies from the simulations

Conclusion

Strengths & Challenges

Strategy 1 uses a fixed overbooking limit without incorporating seasonality or a “no sale” option. It’s a simple and easy-to-understand approach, which makes it very efficient to compute. This makes it a solid starting point for exploring the relationship between overbooking and expected discounted profit. However, it also comes with some major limitations. In practice, airlines rarely use a one-size-fits-all overbooking cap—these limits often vary by route, season, and other risk factors. Plus, the model’s requirement to sell tickets every day, without the flexibility to pause sales when it's strategically wise to do so, can lead to poor outcomes. The lack of adaptability makes this method overly simplistic and likely to underperform in more realistic scenarios.

Strategy 2 builds on the first by adding a “no sale” option for coach tickets. This adds much-needed flexibility, allowing the airline to hold off on selling seats when it makes sense to do so. It enables smarter pacing of ticket sales, which can help avoid unnecessary overbooking and improve overall profits. That said, the added flexibility also brings more complexity: the model now has to make more nuanced decisions each day. And while this is a step forward, it still doesn’t consider external influences like seasonal shifts in demand, which are key in airline revenue management.

Strategy 3 introduces seasonality into the model, making it much more aligned with how real-world demand behaves. As demand increases closer to the departure date, the model can shift ticket sales into those high-demand periods, boosting profitability. This strategy does a better job balancing profit and risk, and offers a more realistic view of how airlines manage bookings. However, it also adds computational complexity and can be harder to interpret. It’s the most representative of real airline operations so far, though there’s still room for improvement, some of which are explored in the “Future Considerations” section.

Future Considerations

Future research could explore additional real world complexities. One idea that should be considered is customer segmentation, where different passenger groups have different sensitivities to price and the risks of overbooking. Utilizing dynamic demand forecasting models that adjust probabilities based on external factors such as competitor pricing, or economic conditions, could further influence decision making. Additionally, analyzing the long term impact of overbooking strategies on customer loyalty and brand perception could be valuable in understanding when overbooking should occur and when it should be avoided. Future models could also examine the potential for partnerships with other airlines that a company may own in order to easily rebook bumped passengers.

Recommendations

Based on the findings of this project, airlines should consider implementing an overbooking strategy that balances maximizing profit with customer satisfaction. Having a flexible overbooking limit that adjusts based on seasonality and real time demand allows airlines to continuously choose the best option based on a given day. These techniques allow airlines to improve their revenue by better managing their seat inventory. Since overbooking is not a new strategy, airlines are typically equipped to compensate overbooked customers. Adopting personalized compensation strategies will incentivize customers to voluntarily rebook their flight. Lastly, integrating predictive analytics into pricing models can help airlines refine overbooking policies and reduce the risk of passenger dissatisfaction when it comes to overbooking and being rebooked.

After analyzing the graphs and key performance metrics, it is clear that the optimal strategy ultimately depends on the airline's priorities. Interestingly, all strategies show a 100% overbooking frequency, making it important for airlines to focus their efforts on kick off frequency, average overbooking cost, profit volatility, and average discounted profit. Among the strategies, using the first policy with seasonality (without a "no-sale" option) results in a lower average overbooking cost than other strategies, which likely contributes to its higher average discounted profit, since they are spending less money overbooking passengers. This strategy also displays decreased profit volatility, offering the airline greater financial stability.

While the no-sale policy with seasonality is attractive due to its flexibility and realistic approach, its resulting profit is too similar to that of the seasonality policy without a “no-sale” option to justify its high profit volatility and overbooking cost. It is recommended that the airline prioritize a stable strategy that achieves a favorable balance between minimizing overbooking expenses and maximizing profitability. The first policy with seasonality satisfies this goal and will greatly help the airline optimize its operations