TEXAS McCombs

Project 2: Optimization II

# Dynamic Programming Solution for Airline Ticket Pricing & Overbooking

By Ibrahim, Jagruta Advani, Rohan Giri, Hadley Krummel

## Introduction

Airline seat overbooking has been a debated topic within the airline industry since the 1950s (Grivet 3). Anywhere between 5-15% of passengers will miss their scheduled flight due to underestimating traffic and security times, delayed connections, or simply deciding otherwise. Underfilling seats encroaches on the airline industry's tight profit margins, with the global airline industry profit margin forecast to be 3.6% in 2025. This leaves little room for open seating.

The solution is to overbook flights. However, there is a delicate PR balance that must be recognized as involuntarily bumping passengers from flights can give an airline a bad reputation; United Airlines had to forcibly pull and drag a passenger from a flight who was selected to be bumped in 2017. This led to a 1.1% drop in United stock the following day (Kottasová). An instance that airlines would hope to avoid. Thus, overbooking policies tread a fine line between losing revenue and losing customers, and the following analysis will demonstrate how to maximize revenues given various overbooking strategies.

## Methodology

### *Dynamic Programming Setup*

We begin this dynamic programming problem by defining the current state of the situation. In this case, days until takeoff, seats sold in coach, and seats sold in first class represent the **state variables** of this problem:

```
T = 365  # number of days until departure (number of selling opportunities)
coach_capacity = 100  # physical coach seats
first_capacity = 20   # physical first-class seats
```

Where T indicates the maximum number of days until takeoff, and coach_capacity and first_capacity indicate the maximum number of seats on the plane for the baseline problem.

Now, since we will be adjusting the number of seats overbooked, we set the max amount of seats that can be sold in both coach and first class for ease of coding and readability:

```
# Define the maximum number of coach tickets that can be sold (physical + oversold)
C_max = coach_capacity + overbook
F_max = first_capacity  # first-class has a hard cap (no overbooking allowed)
```

Now for the dynamics. In this problem, T regularly decreases by 1 each day, representing time ticking down until takeoff:

```
# Loop backwards over time: from 1 day remaining to T days remaining.
for t in range(1, T + 1):
```

The above code demonstrates the first part of the backwards induction recursion step of the process needed to create the **Bellman Equation**. Each of the possible sales probabilities is tested for each day, starting with 1 and continuing until 365 to maximize total possible revenue.

As for the dynamics for seats sold in coach and first class, there are four possible situations given that only 0 or 1 ticket can be sold in either coach or first class each day. For our solution, we store the current dynamics in a 3D matrix. This matrix stores the maximum expected total discounted profit for each day and sales possibility, or how much the airline can maximally expect to make if they make the most optimal pricing decisions from any select day onwards. This array is initialized with all -np.inf values as a baseline and is then updated later:

```
# DP[t, c, f] represents the optimal expected discounted profit with t days remaining
# and having sold c coach tickets and f first-class tickets.
DP = np.full((T + 1, C_max + 1, F_max + 1), -np.inf)
```

Moreover, it is important to note the **terminal value** and how it is incorporated into the problem. The terminal value in this situation represents that at takeoff, the airline can no longer make any sales and that it can only accumulate costs based on the current state of overbooking. Initially, the terminal value is initialized with zeros:

```
terminal_value = np.zeros((C_max + 1, F_max + 1))
```

Then, we created a for loop to loop over all possible values of c and f to comprehensively estimate overbooking costs:

```
for c in range(C_max + 1):
    for f in range(F_max + 1):
        expected_cost = 0.0

        # Loop over possible show-up outcomes for coach tickets:
        for i in range(c + 1):
            p_i = binom.pmf(i, c, p_show_coach)

            # Loop over possible show-up outcomes for first-class tickets:
            for j in range(f + 1):
                p_j = binom.pmf(j, f, p_show_first)
                if i <= coach_capacity:
                    cost = 0
                else:
                    extra = i - coach_capacity

                    # Available first-class seats for bumped passengers:
                    available_first = max(first_capacity - j, 0)
                    bumped_up = min(extra, available_first)
                    bumped_off = extra - bumped_up
                    cost = 50 * bumped_up + 425 * bumped_off
                expected_cost += p_i * p_j * cost

        # Terminal value is negative cost (since cost reduces profit)
        terminal_value[c, f] = -expected_cost
```

The expected cost starts at 0, then all possible show-up outcomes are looped over for both coach and first class tickets. In this situation, it's helpful to see how the loop is set up to ensure that the bumping up and off policies are correctly formatted and related. The probabilities of passengers' arrivals are incorporated into this loop:

```
# Show-up probabilities at departure:
p_show_coach = 0.95
p_show_first = 0.97
```

The binom.pmf line calculates the probability that exactly i and j passengers arrive within coach and first class. We utilize these values later in the expected cost. Next, we subtract coach capacity from i to get the amount of extra passengers there are. Then, we find the amount of available first class seats in the 'available_first' line, see if extra passengers or available_first is smaller in the 'bumped_up' line, and then calculate the amount of passengers which need to be bumped off voluntarily or involuntarily in the 'bumped_off' line, which is always greater than or equal to zero due to our previous calculation strategy.

Then, the bumped_up and bumped_off costs are calculated and multiplied by the exact likelihood of i and j amount of passengers showing up. These occurrences are assumed to be independent of each other as given, so a simple multiplication function suffices. This cost value is then added to the 'expected_cost' variable, which is ultimately saved in the terminal value array with the respective values of c and f seats sold.

The following code demonstrates how these terminal values are matched to the DP 3D array:

```
# Terminal condition (t = 0): profit equals terminal value
for c in range(C_max + 1):
    for f in range(F_max + 1):
        DP[0, c, f] = terminal_value[c, f]
```

Now we'll get into the actual dynamic choices that can occur since we've covered the required problem setup. The first possible outcome that we included in our code is when no sales occur. This is represented by leaving the c and f variables unchanged in our DP array.

```
# Outcome 1: No sale today. State remains (c, f).
next_val = DP[t - 1, c, f]
revenue = 0.0
expected_value += prob_none * (revenue + beta * next_val)
```

In the above code, we multiply the beta, or discount factor, by the next_val which has been calculated previously and selected from our DP array, then add this to the revenue we get from making no sales, $0, to combine the current value of the sale and maximal value of all future sales as 'expected_value'. This represents the formulation of our **value equation**. Below demonstrates how the beta is coded as given:

```
# Discount factor: Annual discount rate of 17% gives a daily factor:
beta = 1.0 / (1.0 + 0.17 / 365)
```

The probability of no sales is calculated as below:

```
prob_none = (1 - p_sale_coach) * (1 - p_sale_first)
```

There are several possible variations of p_sale_coach and p_sale_first, given the state of the problem. Each of these variations represents how the probability of selling a ticket in either coach or first class varies based on given guidelines.

First, if c is greater than or equal to C_max, which was defined earlier as the number of seats in coach plus the overbooking amount, the probability of selling coach seats equals zero because we cannot sell more than the maximum overbooking plus seat amount.

```
# Effective coach sale probability:
if c >= C_max:
    p_sale_coach = 0.0  # cannot sell more if at max allowed
```

However, if there are seats available, the loop iterates over both options of tickets prices for both coach and first class to document the expected_value in the DP 3D array and stores the optimal pricing decisions at each state in the policy array:

```
# We can also store the optimal pricing decision for each state.
policy = np.empty((T + 1, C_max + 1, F_max + 1), dtype=object)
```

The following code blocks demonstrates how we index into our coach sales dictionary based on price and give a qualifying state that increases the probability by 3 percentage points if first class is sold out:

```
else:
    p_sale_coach = coach_probs[cp]

    # Increase probability by 3 percentage points if first-class is sold out.
    if f >= F_max:
        p_sale_coach += 0.03
    p_sale_coach = min(p_sale_coach, 1.0)
```

Probability dictionaries for reference:

```
# Pricing options and base sale probabilities (if a ticket is available):
# For coach tickets:
coach_prices = [300, 350]
coach_probs = {300: 0.65, 350: 0.30}

# For first-class tickets:
first_prices = [425, 500]
first_probs = {425: 0.08, 500: 0.04}
```

While the relationship that p_sale_first equals 0 when f is greater than or equal to F_max remains the same in the first class pricing, a condition is included to indicate that customers will not buy first class seats if a coach is fully booked:

```
else:
    # If coach tickets sold exceed physical capacity, customers don't buy first-class.
    if c > coach_capacity:
        p_sale_first = 0.0

    else:
        p_sale_first = first_probs[fp]
```

With these specifics in mind, we can now cover the remaining sales possibilities. The rest of the sales possibilities follow the below structure. They vary based on whether f or both c and f are being considered and will map to their respective pricing dictionaries and probabilities.

```
# Outcome 2: Only coach ticket sold (if possible).
if c + 1 <= C_max:
    next_val = DP[t - 1, c + 1, f]
    revenue = cp
    expected_value += prob_coach_only * (revenue + beta * next_val)
```

Below are the respective probability calculations. Notice one minus the probability of a sale is incorporated into each situation where a sale is not made to ensure a comprehensive calculation.

```
prob_coach_only = p_sale_coach * (1 - p_sale_first)

prob_first_only = (1 - p_sale_coach) * p_sale_first

prob_both = p_sale_coach * p_sale_first
```

Ultimately, our code iterates through all combinations of t, c, and f, calculating expected value at each step and storing the maximum possible profit in the DP array. The corresponding optimal pricing decisions are recorded in the policy array for every possible state. Together, these steps optimize the airline's seat pricing strategy and provide a comprehensive framework for decision-making across all ticketing scenarios.

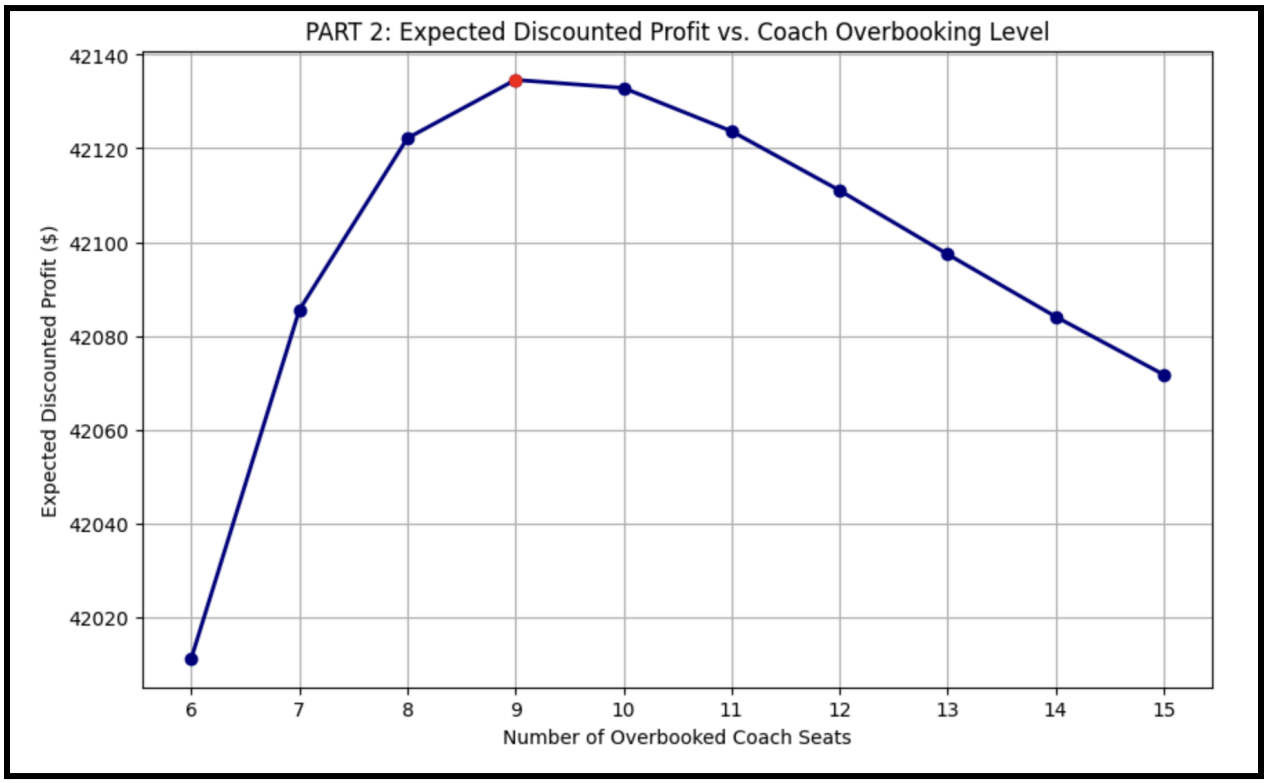
## *Initial Policy Simulation (5-seat overbooking)*

In the initial policy simulation, only 5 seats are allowed to be overbooked, totaling to a C_max of 105 tickets. The dynamic programming loop follows the previous outline given a specified overbook amount of 5. After implementing this policy and code, we achieved a total expected discounted profit of $41886.16.

```
Solving DP for fixed coach overbooking = 5
  Expected discounted profit with 5 overbooking: $41886.16
```

## *Policy Comparison (5–15 seats overbooked)*

This policy exactly resembles the previous one, except that we are now testing our code with a range of 5 to 15 seats overbooked to see how that affects the total expected discounted profit. The best policy for this situation is to overbook by 9 seats. The highest expected discounted profit sums to $42134.62. Below is a graph of the profits and number of overbooked seats.




## *Three-Action Coach Policy*

In the baseline model described previously, whether or not a customer buys or doesn't buy a ticket is up to chance. In the Three-Action Coach policy, we add a third option to implement a strategic element which we can optimize. This is the 'no sale' option, which allows the airline to prevent overbooking from occurring too soon in the sales process. This helps to

optimize revenues because first class sales are constrained to whether or not coach seats are available. Thus, in theory, this policy will allow for more first class seats to be sold since the airline can now place coach sales on hold strategically.

The code differs in that we now loop through a list of lists to include all coach ticketing options instead of a dictionary as previously:

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

And we force the sale probability to equal 0 if the action equals 'no_sale'. The best action is then stored exactly as it was in the baseline problem - in the policy array.

```
# If the action is no sale, force sale probability to 0.
if action == 'no_sale':
    p_sale_coach = 0.0
```

This policy has a slightly better result than part 2, where we calculated an expected discounted profit as $42134.62.

```
Overbooking by 20 seats: Optimal expected discounted profit = $42139.89
```

### *Seasonal Demand Adjustment*

To adjust for seasonality, we included calculating the current_day, which counts from 1 to 365 rather than backwards.

```
current_day = T - t + 1  # current simulation day (1 <= current_day <= T)
```

This is then incorporated into the problem as an adjustment to the calculated probabilities for both coach and first-class seats as seen below:

```
# Incorporate seasonality:
p_sale_coach = min(prob * (0.75 + current_day/730), 1.0)
```

Thus, as the current day counts from 1 to 365, the probability of a ticket purchase increases. Ultimately, this model performed worse than all previous models. However, since we still have a cap on how many tickets can be purchased each day, this is most likely the reason for the reduction in profit. It would be easier to capitalize on this situation if the airline were able to sell multiple seats during a busy season when the probability of buying a ticket is higher.

```
Overbooking by 20 seats: Optimal expected discounted profit = $41841.11
```

## Forward Simulation Results and Comparison and Contrast of Policies

To account for the simulation task, we adjusted the previous code to comprehensively document all decisions and outcomes required to complete the task. After this was complete, all that we needed to add was a for loop that repeated the dynamic programming code many times and saved that information, so we could make insightful graphs and histograms demonstrating the distribution of data explored.

The following code demonstrates the beginning of the simulation within our code as well as how the time and seats sold are re-initialized at zero with each runthrough of the loop.

```
for sim in range(num_simulations):
    t = T
    c = 0  # coach tickets sold
    f = 0  # first-class tickets sold
    discounted_revenue = 0.0
```

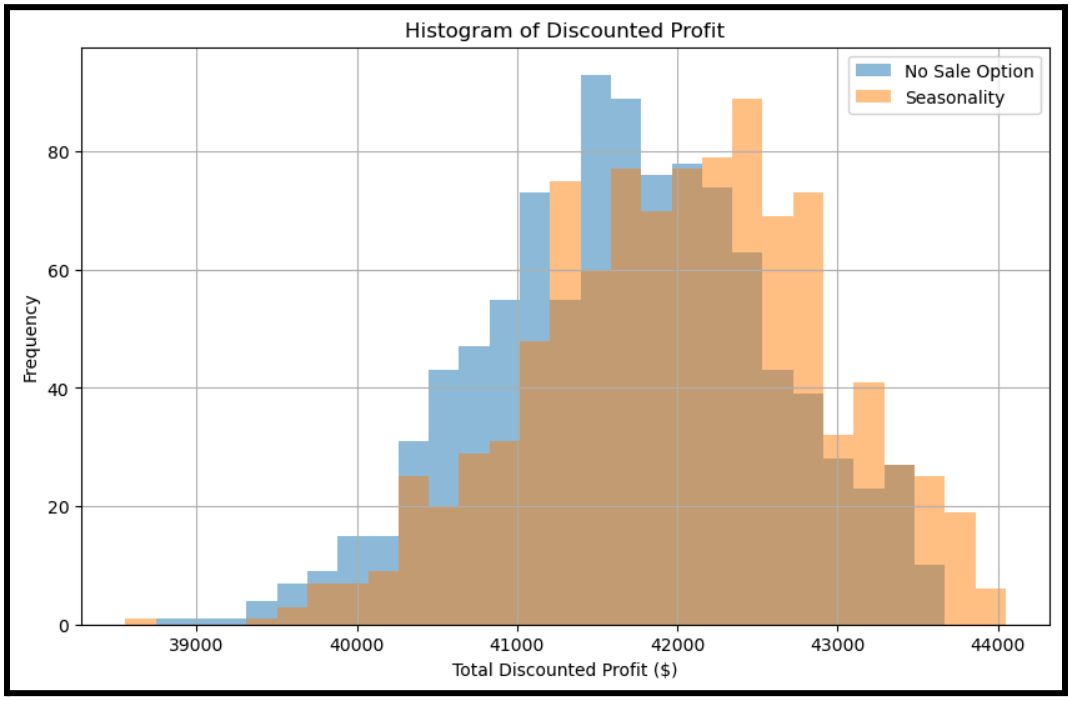
1) *We conducted forward simulations under two key policies, Policy A (No Sale Option) and Policy B (Seasonality Adjustment), each evaluated over 1,000 simulation runs.*

| **Simulation Comparison Table** | | |
|---|---|---|
| | **No Sale Policy** | **Seasonality Policy** |
| **Simulations run** | 1000 | 1000 |
| **Average Discounted Profit** | $41683.30 | $42008.03 |
| **Profit Volatility (Std Dev)** | $885.21 | $897.60 |
| **Coach Overbooking Frequency** | 82.70% | 81.70% |
| **Average Overbooking Cost** | $1040.38 | $1009.75 |
| **Average Bumped Up Passengers** | 0.77 | 0.85 |
| **Average Bumped Off Passengers** | 2.36 | 2.28 |

According to the data we collected in these simulations, the Seasonality Policy performs better than the No Sale Policy in all regards except volatility. The Seasonality Policy may outperform the No Sale policy due to the fact that customers are more likely to show up to their flight if they book it relatively recently. Moreover, the airline is able to capitalize on price optimization when demand is more regular and higher in certain periods as with a seasonal policy.

*2) Graphical Analysis and Interpretation of the 2 Policies*

**Profit Distribution Histograms**

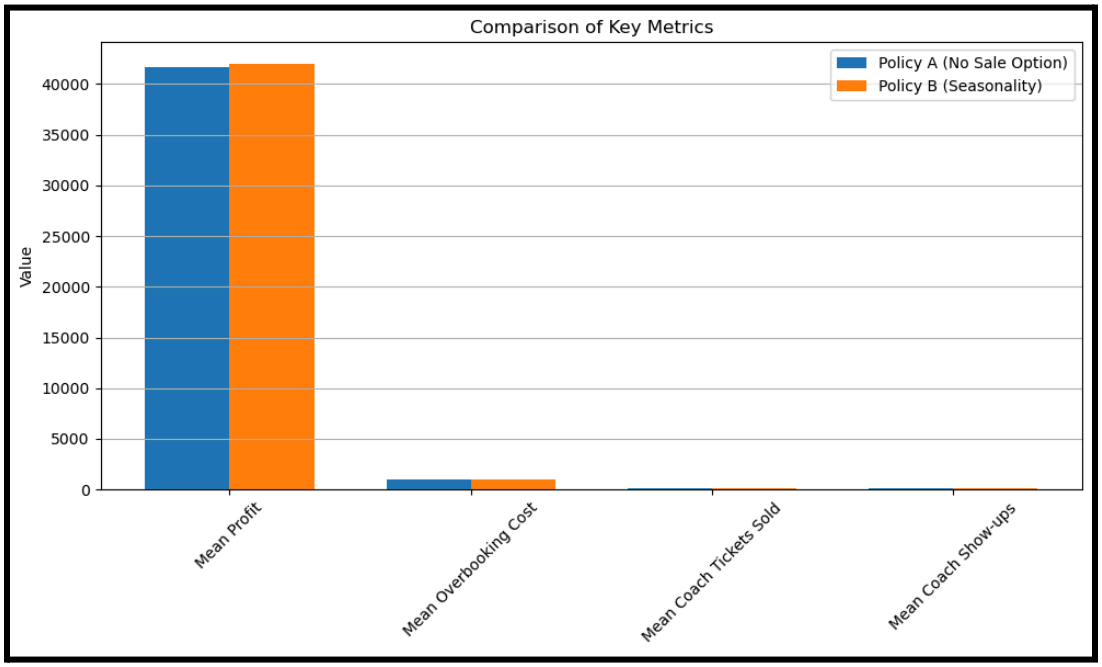




Policy B (Seasonality) shows a histogram shifted to higher profit values, meaning it tends to yield higher profits on average. However, its wider spread indicates greater variability and risk. In contrast, Policy A (No Sale Option) has a tighter profit distribution, suggesting more consistent, predictable outcomes with fewer extreme variations.

## Bar Chart of Key Metrics

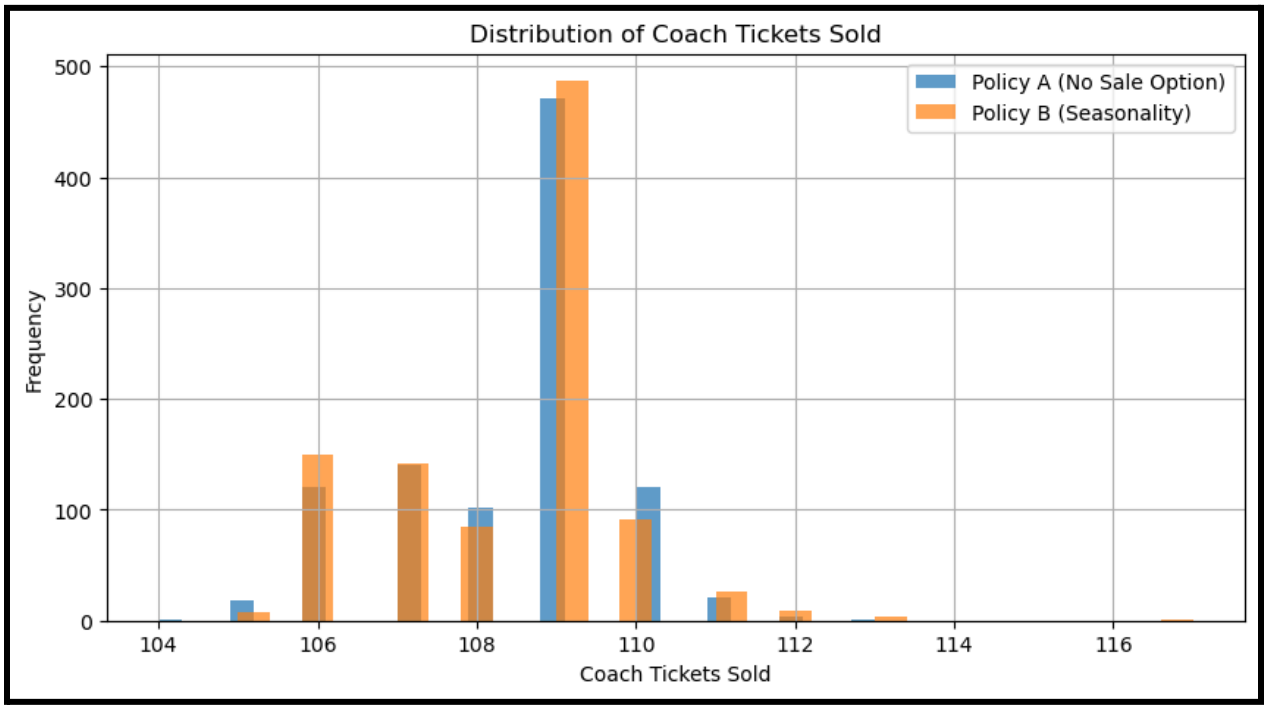
| | Metric | Policy A (No Sale) | Policy B (Seasonality) |
|---|---|---|---|
| 0 | Avg Profit | 41683.303761 | 42008.02603 |
| 1 | Avg Overbooking Cost | 1040.375000 | 1009.75000 |
| 2 | Avg Coach Sold | 108.352000 | 108.34400 |
| 3 | Avg Coach Show-ups | 102.951000 | 102.90800 |
| 4 | Avg First Sold | 19.607000 | 19.54700 |
| 5 | Avg First Show-ups | 19.040000 | 18.91800 |

This table shows average KPIs comparing the No Sale (Policy A) and Seasonality (Policy B) strategies, indicating that while Policy B achieves higher profits and sales, it also incurs slightly higher overbooking costs and risks.




The bar chart shows that Policy B earns more profit on average but also has higher overbooking costs and bumps more passengers. This trade-off means that while higher revenue is possible, it increases risk.

### Coach Ticket Sales Distribution Comparison




Both policies tend to cluster around 109–110 tickets sold, but Seasonality occasionally sells more. This suggests that while Seasonality can drive higher sales, it also introduces greater variability.

## Recommendations & Conclusion

Ultimately, overbooking nine seats and the No-sale policy offer the best pricing strategy for the airline based on the problem guidelines. This solution minimizes the costs of bumping passengers and maximizes revenue for the airline so that we can meet our tight profit margins.

The simulation outcomes further support this recommendation. Under the No Sale Policy (Policy A), our 1,000-run simulation produced an average discounted profit of about $41,683.30, while the Seasonality Policy (Policy B) achieved roughly $42,008.03. Although Policy B shows slightly higher profit and a marginally lower overbooking cost ($1,009.75 vs. $1,040.38), along with a modest reduction in bumped-off passengers (2.28 vs. 2.36), it also exhibits slightly higher profit volatility ($897.60 vs. $885.21). This suggests that while Seasonality can drive higher revenues and sales, it introduces a bit more risk.

When comparing a static, no-action approach to our dynamic strategy, the difference is clear. The no-action option fails to capture the potential for additional revenue from strategic

overbooking and pricing adjustments. In contrast, the optimized strategy—by overbooking nine seats and leveraging seasonality in pricing—improves profits by around $350 to $400 per flight. This optimized policy not only raises average profits but also, in most metrics, reduces the incidence and cost of overbooking, even though it introduces slightly more variability.

While the optimized strategy enhances profitability, it comes with trade-offs. The additional revenue is achieved by bumping a few more passengers on occasion, which could erode customer goodwill if not managed carefully. For instance, even a small increase in the number of bumped passengers or overbooking costs might impact the airline's reputation over time. Therefore, airlines must complement this strategy with robust compensation policies and transparent communication to ensure that customer satisfaction remains high. A way to build on this report would be to add a term beyond the cost of bumping a passenger off that could help quantify how many people an airline could bump off without sustaining PR issues.

## Results Overview

Q1: Overbooking by 5 seats: Optimal expected discounted profit = $41886.16

Q2: Optimal Policy: Overbooking by 9 seats yields the highest profit of $42134.62.

Q3: Overbooking by 20 seats: Optimal expected discounted profit = $42139.89 (no sale policy)

Q4: Overbooking by 20 seats: Optimal expected discounted profit = $41841.11 (Seasonality)

Q5: Simulation Results

## References

Grivet, Marguerite. "Don't Be Fooled by Overbooking Practices." *PM World Journal*, vol. VII, no. III, Mar. 2018, pp. 1–8. PM World Library, https://pmworldlibrary.net/wp-content/uploads/2018/03/pmwj68-Mar2018-Grivet-overbooking-contractual-rights-and-obligations-student-paper.pdf.

Kottasová, Ivana. "United loses $250 million of its market value." *CNNMoney*, 11 Apr. 2017, https://money.cnn.com/2017/04/11/investing/united-airlines-stock-passenger-flight-video/

.