TEXAS McCombs
The University of Texas at Austin
McCombs School of Business

**Kush Patel**

**April 05, 2025**

# DYNAMIC PROGRAMMING FOR AIRLINE OVERBOOKING OPTIMIZATION

## Project Overview

Airlines continuously challenged to strike the perfect balance between maximizing the revenue and managing the inherent risks of overlooking flights. In an environment where ticket demand is highly variable and passenger show-up behavior follows a probabilistic pattern, it becomes a common practice for airlines to sell more tickets than the available seats. This strategy, while potentially increasing revenue through higher ticket sales, also exposes the airline to significant costs if too many passengers turn up on the day of departure. The essence of the problem lies in managing these costs while still capturing as much revenue as possible.

The approach taken here uses a dynamic programming framework—a mathematical method that breaks down complex decision-making processes into simpler, recursive steps—to determine the optimal ticket pricing and overbooking  policies over a given selling horizon. The problem is carefully modeled with two distinct ticket classes, namely coach and first-class, each offering different pricing options and exhibiting unique demand probabilities. While revenue is generated from the sale of these tickets, costs are incurred when overbooking leads to more passengers showing up than can be accommodated, requiring compensatory measures that diminish overall profit.

In this framework, the decision-making process is governed by a recursive Bellman equation, which systematically accounts for daily sales, discounts future revenues, and integrates terminal costs computed from probability distributions of passenger no-shows. This method enables the evaluation of different policies by simulating the sequential nature of the sales process over a full year (365 days), providing valuable insights into how various overbooking strategies impact the airline's bottom line.

## Problem 1: Fixed Overbooking Policy

### Objective

The first challenge is to calculate the expected discounted profit over a 365-day period when the airline allows coach tickets to be oversold by a fixed number, specifically by 5 extra seats. This fixed policy means that even though the coach section has a nominal capacity of 100 seats, the airline will sell up to 105 tickets

### Methodology

1. State Variables
    i. Days remaining until departure $(d)$: This variable decreases with each day, representing the diminishing opportunities to sell tickets.
    ii. Cumulative coach tickets sold $(s_c)$: This tracks how many coach tickets have been sold up to a given day.
    iii. Cumulative first-class tickets sold $(s_f)$: This monitors the sales in the first-class category.

2. Decisions

Four pricing options are available for each day, represented by pairs of coach and first-class ticket prices:
(300, 425), (300,500), (350,425), (350, 500)

3. Demand and Revenue

   Demand probabilities are sensitive to both ticket price and the current state of seat availability. For example, when first-class seats are fully booked, the probability of selling a coach ticket increases by 3% points, reflecting the shifting customer preference under constrained conditions.

4. Terminal Costs

   Terminal costs are incurred on the flight day (when $d = 0$). On this day, no revenue is generated; instead costs arise from the overbooking scenario. The number of passengers who actually show up is modeled using a binomial distribution—95% for coach and 97% for first-class. If more passengers show up than the available seats in the coach section, costs are calculated based on whether excess passengers can be accommodated in first-class or must be bumped completely.

5. Dynamic Programming Recursion

   The dynamic programming recursion is implemented through a function $V(d, s_c, s_f)$ that computes the maximum expected discounted profit from any given state. This function evaluates all possible pricing decisions for the current day, calculates the immediate revenue from each decision, and adds the discounted expected profit of the future state. To improve computational efficiency, the function leverages caching via an LRU cache, ensuring that previously computed results are reused.

```
@lru_cache(maxsize=None)
def V(d, s_c, s_f):
    """
    Recursively compute the maximum expected discounted profit with:
      d   : days remaining until the flight
      s_c : cumulative coach tickets sold
      s_f : cumulative first-class tickets sold

    At d = 0 (flight day), no revenue is earned; only the overbooking cost is incurred.
    """
    if d == 0:
        # Terminal condition: the expected cost (negative profit)
        return -compute_terminal_cost(s_c, s_f)

    best_val = -float('inf')

    # Loop over each pricing decision
    for (coach_price, fc_price) in decisions:
        # Adjust the coach sale probability based on first-class capacity.
        if s_f == 20:
            p_coach = 0.68 if coach_price == 300 else 0.33
        else:
            p_coach = 0.65 if coach_price == 300 else 0.30
        # If maximum coach tickets have been sold, no more sales are possible.
        if s_c >= 105:
            p_coach = 0

        # First-class sale probability
        p_fc = 0.08 if fc_price == 425 else 0.04
        if s_f >= 20:
            p_fc = 0  # first-class is full

        expected_val = 0.0

        # Outcome 1: No sale occurs in either class.
        prob = (1 - p_coach) * (1 - p_fc)
        revenue = 0
        next_val = V(d - 1, s_c, s_f)
        expected_val += prob * (revenue + beta * next_val)

        # Outcome 2: Only a coach ticket is sold.
        prob = p_coach * (1 - p_fc)
        revenue = coach_price
        next_val = V(d - 1, min(s_c + 1, 105), s_f)
        expected_val += prob * (revenue + beta * next_val)

        # Outcome 3: Only a first-class ticket is sold.
        prob = (1 - p_coach) * p_fc
```

```
        revenue = fc_price
        next_val = V(d - 1, s_c, min(s_f + 1, 20))
        expected_val += prob * (revenue + beta * next_val)

        # Outcome 4: Both a coach and a first-class ticket are sold.
        prob = p_coach * p_fc
        revenue = coach_price + fc_price
        next_val = V(d - 1, min(s_c + 1, 105), min(s_f + 1, 20))
        expected_val += prob * (revenue + beta * next_val)

        # Keep the decision that gives the maximum expected value.
        if expected_val > best_val:
            best_val = expected_val

    return best_val

# Starting with 365 days remaining and no tickets sold.
result = V(365, 0, 0)
print("The expected discounted profit for overbooking by 5 seats is: ${:.2f}".format(result))
```

**Results**

This recursive function, starting from 365 days with no tickets sold, provides the expected discounted profit for the policy that permits selling up to 105 coach tickets. The expected discounted profit for **overbooking by 5 seats** is: **$41886.16**

## Problem 2: Evaluating Variable Overbooking Levels

### Objective

The next step is to identify the overbooking level that maximizes the expected discounted profit. Rather than sticking with a fixed overall of 5 extra seats, this analysis explores varying the number of extra coach tickets sold—from 6 to 15—and determines which scenario yields the highest profit.

### Methodology

1. Policy Function

   To address this problem, a helper function, `solve_policy(max_coach),` is used . This function adjusts the maximum number of coach tickets allowed based on the overbooking level. For instance, if the airline wishes to oversell by 6 seats, the maximum coach tickets become 106; if by 7, then 107, and so on.

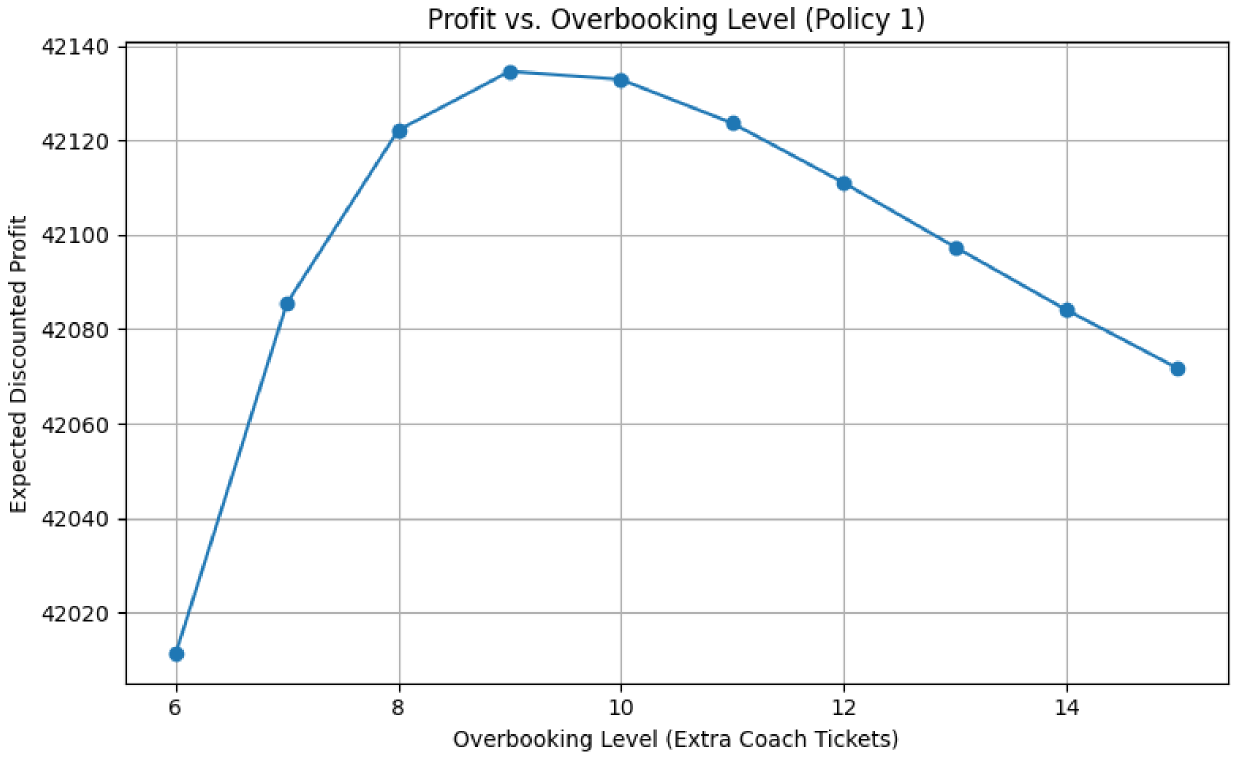
2. Iteration and Comparison

   A loop iterates over the desired range of extra seats. For each overbooking level, the function recalculates the expected profit using the dynamic programming  recursion described in Problem 1. The results of each scenario are stored, and the overbooking level with the highest computed profit is identified. The detailed comparative analysis provides clear insights into how adjusting the overbooking level can influence overall profitability

```
results_policy1 = []
for extra in range(6, 16):
    max_coach = 100 + extra
    profit = solve_policy(max_coach)
    results_policy1.append((extra, profit))
    print(f"Overbooking by {extra:2d} seats (max coach =
{max_coach}): Profit = ${profit:,.2f}")
```

**Results**

The simulation results indicate that the optimal fixed overbooking policy is achieved when the coach section is **oversold by 9 seats**, leading to an expected **discounted profit of approximately $42,134.62**.



**Problem 3: Incorporating a “No-Sale” Option**

**Objective**

The final analysis introduces an enhanced policy that adds an extra decision option: the choice to refrain from selling coach tickets on a particular day. The goal here is to assess whether this additional flexibility—allowing the airline to intentionally avoid making a sale when it might be strategically disadvantageous—can further improve profitability compared to the fixed sales cap strategy of Problems 1 and 2.

**Methodology**

1. Expanded Decision Set

   Under this enhanced policy, the decision set for coach tickets is expanded to include three choices:

   - Sell at a low price: Offer the ticket at $300
   - Sell at a high price: Offer the ticket at $350
   - No sale: Deliberately choose not to sell a coach ticket on that day

2. Increased Flexibility

   For first-class tickets, the available options remain unchanged, with two pricing alternatives. The inclusion of the “no-sale” option allows for better management of the overbooking risk, particularly as the flight date approaches. It provides the model with the capability to moderate ticket sales strategically rather than simply capping the number of tickets sold. Consequently, the maximum allowable coach tickets in this enhanced model is raised to 120.

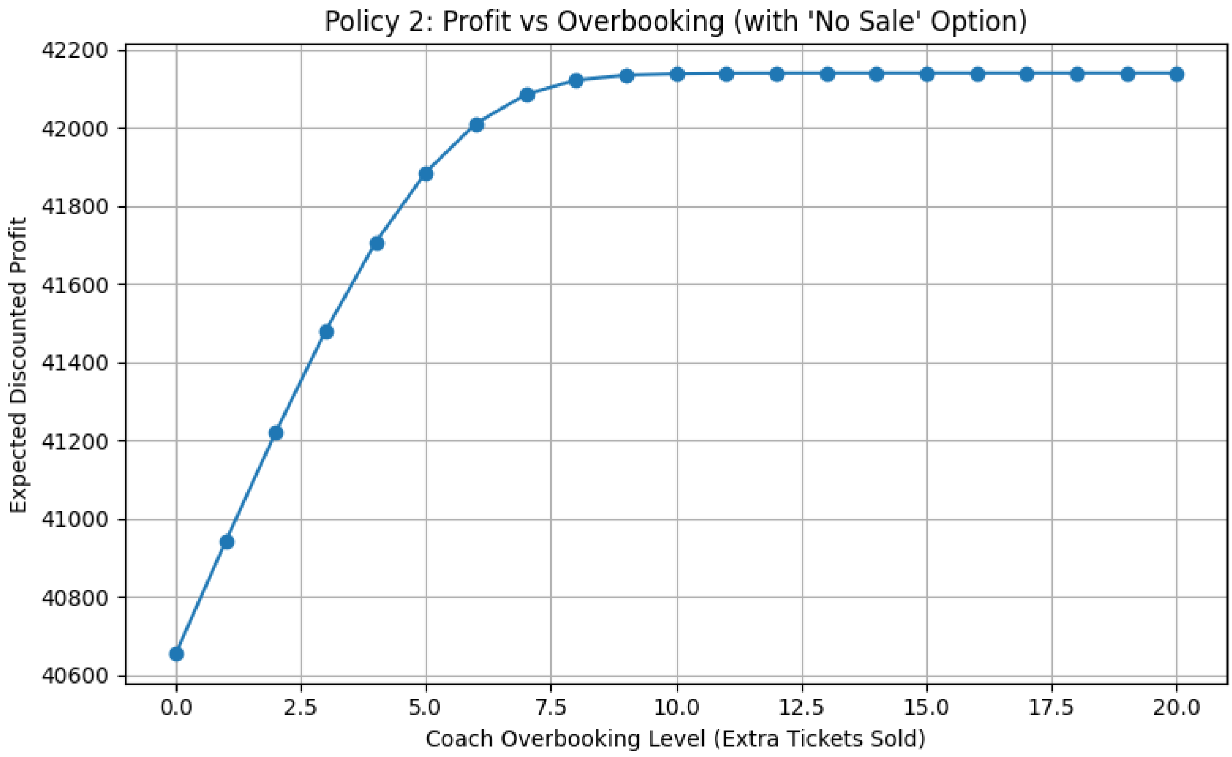
3. Modified Dynamic Programming

   The dynamic programming recursion is modified accordingly, The new recursive function, `solve_policy_no_coach_sale(max_coach),` iterates over all combinations of actions for coach and first-class tickets, including the option not to sell a coach ticket. The expected profit is computed in a manner similar to the previous models, but with the added flexibility in the decision-making process. This is demonstrated through the following code:

```
def solve_policy_no_coach_sale(max_coach):
    @lru_cache(maxsize=None)
    def V(d, s_c, s_f):
        if d == 0:
            return -compute_terminal_cost(s_c, s_f)
        best_val = -float('inf')
        for coach in coach_options:
            p_coach = coach["full_fc_prob"] if (s_f == 20 and coach["action"] != "none") else (coach["base_prob"] if coach["action"] != "none" else 0.0)
            if s_c >= max_coach:
                p_coach = 0.0
            for fc in fc_options:
                p_fc = fc["prob"] if s_f < 20 else 0.0
                expected_val = (
                    (1 - p_coach) * (1 - p_fc) * (0 + beta * V(d - 1, s_c, s_f)) +
                    p_coach * (1 - p_fc) * (coach["price"] + beta * V(d - 1, min(s_c + 1, max_coach), s_f)) +
                    (1 - p_coach) * p_fc * (fc["price"] + beta * V(d - 1, s_c, min(s_f + 1, 20))) +
                    p_coach * p_fc * (coach["price"] + fc["price"] + beta * V(d - 1, min(s_c + 1, max_coach), min(s_f + 1, 20)))
                )
                best_val = max(best_val, expected_val)
        return best_val
    return V(365, 0, 0)
```

**Results**

By running this enhanced policy across a range of overbooking levels—from 0 to 20 extra coach tickets—the simulation reveals that the best performance is achieved when coach tickets are **oversold by 20 seats**, which corresponds to selling up to 120 tickets. This scenario yields an expected discounted profit of approximately **$42,139.89**. Although the profit improvement compared to the fixed policy is modest, the additional decision flexibility afforded by the "no-sale" option provides valuable strategic benefits, particularly in scenarios with heightened uncertainty.



## Problem 4: Adding Seasonality

### Objective

The next step builds upon the enhanced policy from Problem 3 by incorporating seasonality into the demand model. The goal here is to assess whether dynamically adjusting sale probabilities—reflecting the natural increase in last-minute ticket demand as the flight date approaches—can further improve profitability compared to the fixed probability framework used in Problems 1, 2, and the enhanced “no-sale” policy of Problem 3.

### Methodology

1. Expanded Decision Set with Seasonality

   Under this refined policy, the decision set for coach tickets remains expanded to three options:

   - Sell at a low price: Offer the ticket at $300
   - Sell at a high price: Offer the ticket at $350
   - No sale: Deliberately choose not to sell a coach ticket on that day

   For first-class tickets also, the available pricing alternatives remain unchanged

   - Low price: $425
   - High price: $500

2. Increased Flexibility

   With seasonality incorporated, the model adjusts sale probabilities to reflect that, as the departure date approaches, ticket demand increases. To accommodate this increased demand potential.

3. Modified Dynamic Programming

The dynamic programming recursion is modified accordingly, The new recursive function, `def solve_policy_with_seasonality(max_coach, allow_no_sale)`, iterates over all possible combinations of actions (including the "no-sale" option for coach tickets) using the seasonally adjusted probabilities. The terminal cost is computed as in previous models, using a binomial model for no-shows and associated overbooking penalties.  This is demonstrated through the following code:

```
# Solve the dynamic programming problem with seasonality effects
# allow_no_sale determines if the "no sale" option is allowed for coach seats
def solve_policy_with_seasonality(max_coach, allow_no_sale):
    # Determine available coach options based on whether "no sale" is allowed
    coach_options = coach_options_all if allow_no_sale else
coach_options_all[:2]  # skip "none" if not allowed

    @lru_cache(maxsize=None)
    def V(d, s_c, s_f):
        # Base case: when no days are left, return negative terminal cost
incurred on flight day
        if d == 0:
            return -compute_terminal_cost(s_c, s_f)

        # t represents the current day in the booking period (1-indexed)
        t = total_days - d + 1
        # multiplier introduces a seasonal adjustment to the probabilities
        multiplier = 0.75 + t / 730
        best_val = -float('inf')

        # Loop through each coach decision option
        for coach_label, coach_price, base_prob, full_fc_prob in coach_options:
            p_coach = 0.0
            # For options other than "none", compute the probability of a coach
sale
            if coach_label != "none":
                p_coach = full_fc_prob if s_f == 20 else base_prob
                # Apply the seasonal multiplier to adjust the probability,
capped at 1.0
                p_coach = min(p_coach * multiplier, 1.0)
                # If the number of coach seats sold is already at max_coach, no
further coach sale is allowed
                if s_c >= max_coach:
                    p_coach = 0.0

            # Loop through each first-class option
            for fc_price, base_fc_prob in fc_options:
                # Adjust first-class sale probability with seasonality, capped
at 1.0
                p_fc = min(base_fc_prob * multiplier, 1.0)
                # If first-class seats are already full (20 seats), no sale is
allowed
                if s_f >= 20:
                    p_fc = 0.0

                expected_val = 0.0
                # No sale occurs for both coach and first-class
                expected_val += (1 - p_coach) * (1 - p_fc) * (0 + beta * V(d -
1, s_c, s_f))
                # Only a coach sale occurs
                expected_val += p_coach * (1 - p_fc) * (coach_price + beta *
V(d - 1, min(s_c + 1, max_coach), s_f))
                # Only a first-class sale occurs
                expected_val += (1 - p_coach) * p_fc * (fc_price + beta * V(d -
1, s_c, min(s_f + 1, 20)))
```

```
                # Both sales occur simultaneously
                expected_val += p_coach * p_fc * (coach_price + fc_price +
                                                  beta * V(d - 1, min(s_c + 1,
max_coach), min(s_f + 1, 20)))

                # Update best expected value if current decision yields higher
profit
                best_val = max(best_val, expected_val)

        return best_val
```

### Results

The seasonality-adjusted policy was tested over overbooking levels from 0 to 20 extra coach tickets. Expected discounted profits steadily increased from about $40,358.35 at 0 extra seats, reaching roughly $41,835.69 at 9 extra seats, and plateauing near $41,841.11 for 16 to 20 extra seats. Notably, the model with the “no-sale” option yielded nearly identical results to the fixed policy. Overall, the optimal configuration—overbooking by approximately **16 to 20 seats**—achieves an expected discounted profit of around **$41,841.11**, demonstrating that the seasonal adjustment effectively captures the increase in last-minute demand with modest profit improvements and enhanced strategic flexibility.

## Problem 5: Forward Simulation Results

### Objective

These results were based on averages from 10,000 simulations of each policy fast-forwarded in time. The goal of this analysis is to compare Policy 1 and Policy 2 across key performance indicators, including overbooking rate, passenger rejection rate, overbooking cost, and discounted profit. The purpose is to assess which policy is more effective and consistent when applied in a simulated real-world environment.

## Methodology

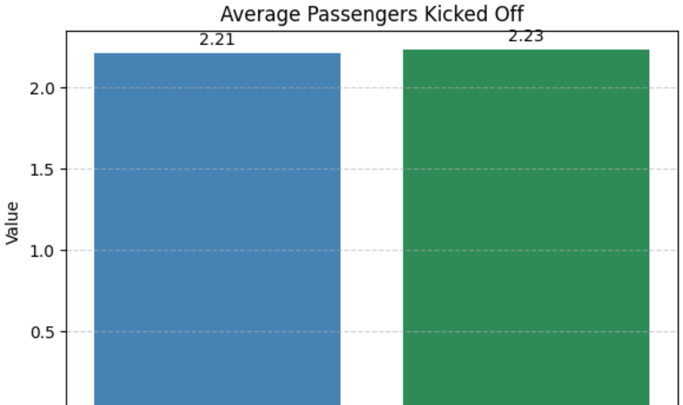
The simulations represent randomized processes, where ticket sales and passenger show-ups are subject to probability, mimicking real operational uncertainty. Results were averaged across 10,000 simulation runs per policy. While randomness means that results can change and sometimes anomalies can pop up, running such a large number of trials ensures that the overall future metrics should look like the results presented below. Each simulation captures a full year of ticket sales, overbookings, and costs, producing reliable estimates of each policy's long-run performance.

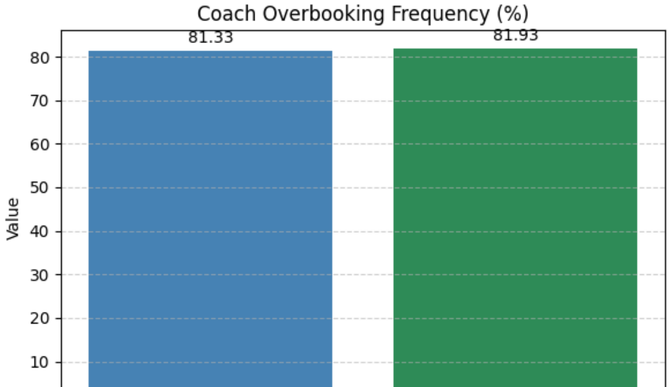
## Results

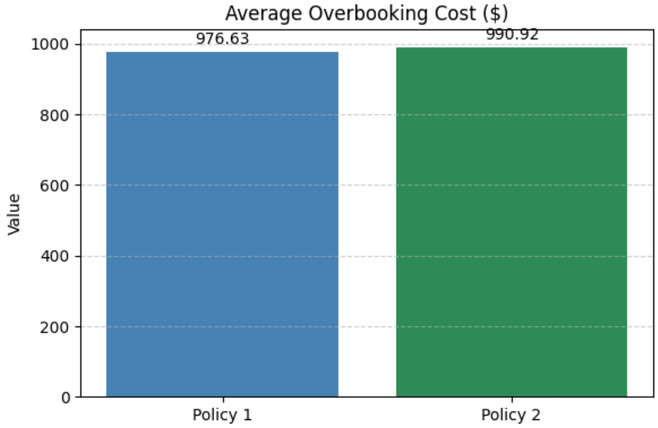
Overall, **the averages are not too far apart** in all of the important metrics: overbooking rate, passenger rejection rate, overbooking cost, and discounted profit are all somewhat close, but a **slight edge can be seen in favor of Policy 1**. Furthermore, **Policy 1 has a higher discounted profit and lower overbooking costs**. So if we base our policy selection on these averages alone, the airline should be more inclined towards Policy 1.

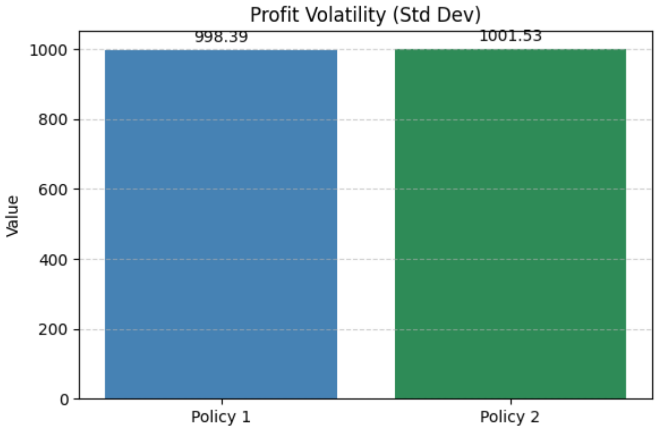
**Policy 1 is also less volatile** based on the average standard deviation, suggesting more predictable results across runs. However, this data is only a simple average and does not show a full picture. To get a more complete view, the distribution of profits across all simulations was analyzed.

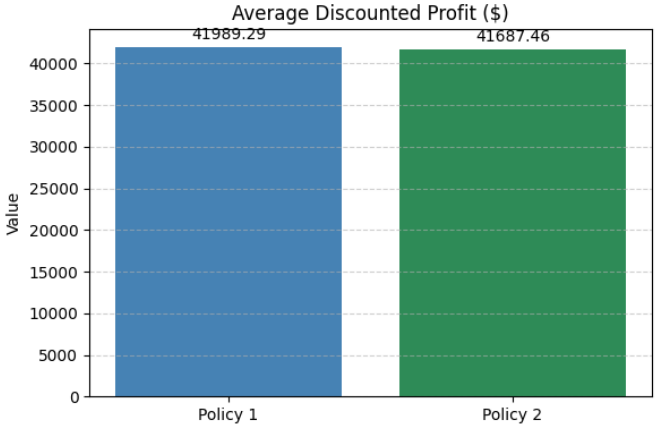
Averages for Profitability and Overbooking Metrics:



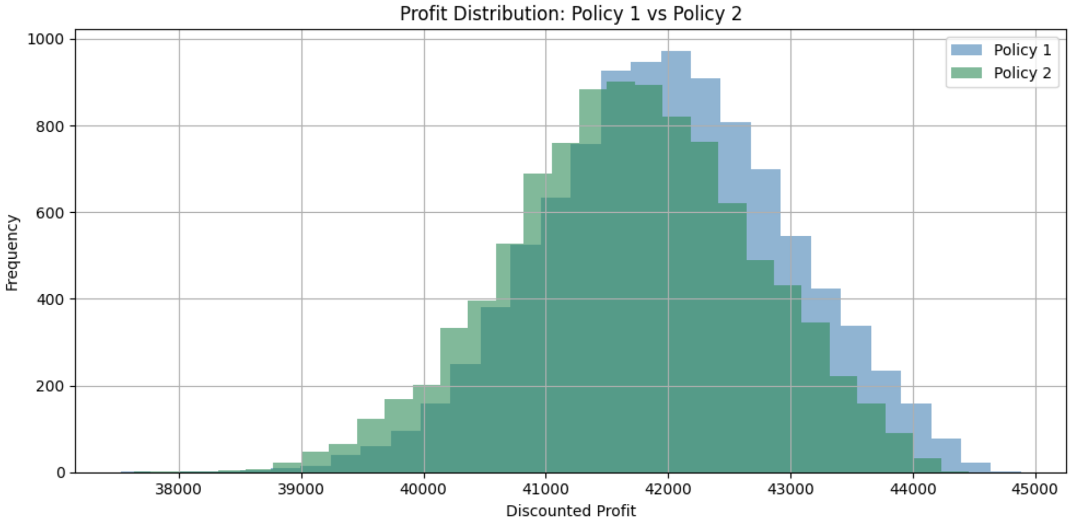



Average Overbooking Cost ($)
Value
0
200
400
600
800
1000
976.63
990.92
Policy 1
Policy 2

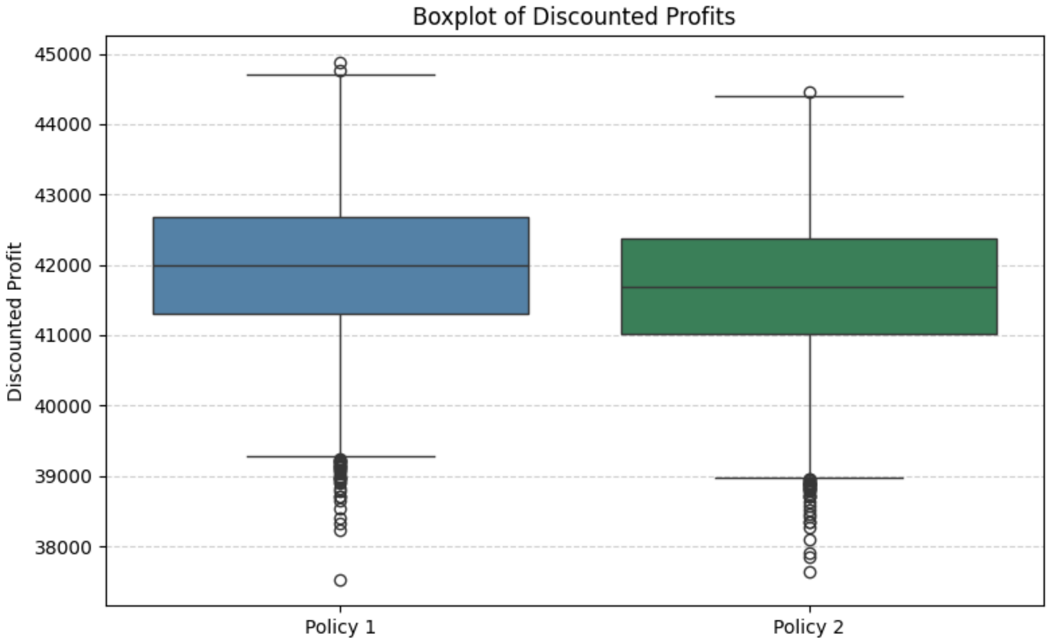

Profit Volatility (Std Dev)
Value
0
200
400
600
800
1000
998.39
1001.53
Policy 1
Policy 2


Average Discounted Profit ($)
Value
0
5000
10000
15000
20000
25000
30000
35000
40000
41989.29
41687.46
Policy 1
Policy 2

## Distribution of Discounted Profits



The above distribution overlays Policy 1 and Policy 2 on how often certain buckets of discounted profits occur. It appears that Policy 1’s distribution is similar in shape (normal/slightly left skewed) to Policy 2, but its median/overall distribution is shifted further right than Policy 2’s distribution. Additionally, it appears that Policy 1’s discounted profits occur at higher amounts and higher frequencies when profits are around 42,000 and larger. This exhibit provides more evidence that higher profits should usually be generated under Policy 1. The boxplot below provides a clearer picture of this observation.



## Additional Insights

A stress test will show which policy performs better under worse conditions. Increasing the coach attendance rate (from 95% to 98%) puts pressure on the overbooking process and the number of first class seats available for overbooked customers.

| **Metric** | **Policy 1 (Base)** | **Policy 1 (Stress)** | **Policy 2 (Base)** | **Policy 2 (Stress)** |
|---|---|---|---|---|
| Avg. Profit ($) | 41989.29 | 40750.07 | 41687.46 | 40621.80 |
| Std Dev Profit | 998.39 | 867.91 | 1001.53 | 1021.11 |
| Avg. Overbooking Cost | 976.63 | 2207.95 | 990.92 | 2177.34 |
| Overbooking Frequency | 81.33 | 99.45 | 81.93 | 99.5 |

| | | | | |
|---|---|---|---|---|
| (%) | | | | |
| Avg. Passengers Rejected | 2.21 | 5.08 | 2.23 | 5.00 |

*Table 1: Coach class attendance rate increases/first class stays constant*

As expected the increased economy class attendance lowers profits in both cases. Something interesting to note is that the slippage in Policy 1's case is $1200+ while the slippage in Policy 2's case is only slightly higher than $1000. This could be explained from the due diligence created by including seasonality in the calculations. Based on this simulated data, it appears that under flexible conditions (less coach attendance) Policy 1 is a much better choice due to the much higher profits, lower volatility, lower costs, less overbooking, and lower passengers rejected. However, when the 'show-up' rate of economy class flyers increases, there is less flexibility in operations when it comes to overbooking. This seems to cause some metrics to flip in comparison to Policy 2; for example, more passengers are rejected, more profit slippage is observed, and the overbooking cost is also higher. Strictly speaking, the profit is higher under policy 1 even under duress, but it is not as wide a gap compared to before.

This would indicate under stressed conditions where economy class attendance class increases only, Policy 2 may be more prudent as there is a similar enough profit, a comparable overbooking rate, less overbooking cost, and less passengers rejected (which may provide more goodwill among customers). In better conditions, Policy 1 will have a better yield. Thus, it could be a good idea for the airline to keep both policies while alternating which one is active depending on daily data collected.

| **Metric** | **Policy 1** | **Policy 1** | **Policy 2** | **Policy 2 (Stress)** |
|---|---|---|---|---|

| | **(Base)** | **(Stress)** | **(Base)** | |
|---|---|---|---|---|
| Avg. Profit ($) | 41989.29 | 41874.21 | 41687.46 | 41718.04 |
| Std Dev Profit | 998.39 | 1015.13 | 1001.53 | 1120.20 |
| Avg. Overbooking Cost | 976.63 | 1082.73 | 990.92 | 1086.20 |
| Overbooking Frequency (%) | 81.33 | 81.22 | 81.93 | 82.51 |
| Avg. Passengers Rejected | 2.21 | 2.49 | 2.23 | 2.50 |

*Table 2: First class attendance rate increases/coach class stays constant*

What is interesting in this case is that Policy 2 has increased profits when the first class attendance rate increases. Nevertheless, the overall average profit is still better in Policy 1. Additionally, the overbooking cost/frequency is less under Policy 1, while the rejected passenger average is about the same in both policies. So here, one would be inclined towards Policy 1.