THE UNIVERSITY OF TEXAS AT AUSTIN
McCOMBS SCHOOL OF BUSINESS

OPTIMIZATION PROJECT 2: DYNAMIC PROGRAMMING

Mar. 29, 2024

Group 24:

Davis Gill, Hayoung Kim, Jiarui Chang, Vanesa Alcantara Panta

Table of Contents

1. Introduction

1.1. Project Overview

In the competitive landscape of the airline industry, strategic overbooking and dynamic pricing play pivotal roles in maximizing revenue while managing customer satisfaction. This report delves into the exploration of optimal overbooking strategies for an airline, focusing on two distinct approaches: a fixed overbooking limit method and a dynamic overbooking method with a no-sale option. Both approaches aim to strike a balance between maximizing expected discounted profit from ticket sales and minimizing the costs associated with overbooking, such as compensating bumped passengers.

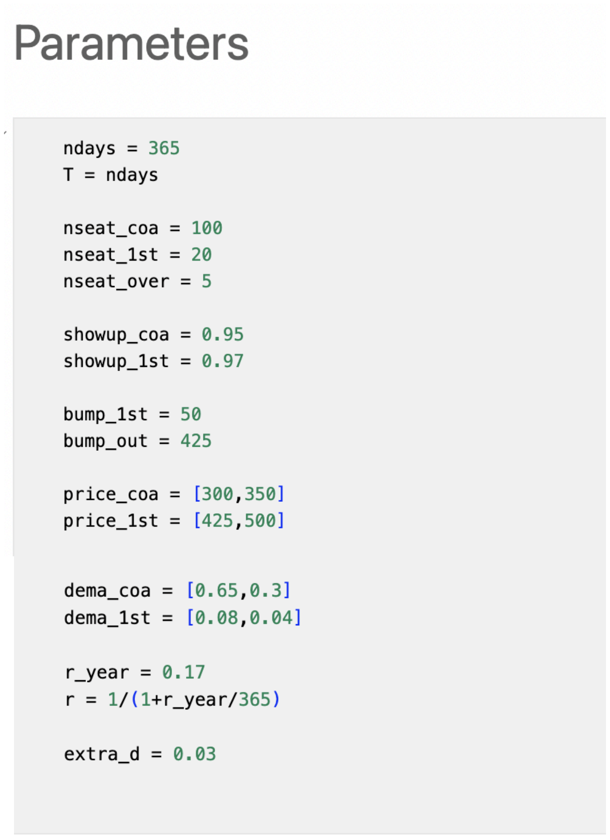
The fixed overbooking limit method sets a predefined threshold for the number of coach tickets that can be oversold, ranging from 5 to 15 seats beyond the plane's capacity. Through dynamic programming, this approach seeks to determine the optimal pricing policy for coach and first-class tickets that maximizes the airline's profit while adhering to the overbooking limit. The dynamic overbooking method with a no-sale option introduces a more flexible strategy, allowing the airline to halt coach ticket sales on certain days to manage the risk of overbooking more effectively. This approach aims to optimize ticket pricing and overbooking decisions dynamically, based on the evolving demand and remaining days until departure.

The effectiveness of these overbooking strategies is further evaluated through forward simulations, which provide insights into the frequency of overbooking occurrences, the incidence of passengers being bumped, and the volatility of discounted profits. The comparative analysis of these simulations helps in understanding the trade-offs between the two approaches and their impact on the airline's profitability. The report culminates in a recommendation that aligns with the airline's objectives of maximizing revenue while maintaining a positive customer experience, backed by a thorough analysis of the overbooking policies and their implications on the airline's bottom line.

1.2. Key Parameters

In this project, we aim to optimize the pricing policy and overbooking strategy for an airline to maximize expected discounted profit. We focus on a flight with 100 coach and 20 first-class seats, considering various overbooking policies for coach while keeping first-class bookings fixed for customer satisfaction. The key factors we need to consider for our dynamic programming analysis include operational constraints, customer behavior probabilities, and financial metrics as outlined below:

- Number of Days until Departure (ndays): 365
- Number of Seats in Coach (nseat_coa): 100
- Number of Seats in First-Class (nseat_1st): 20
- Overbooking Limit for Coach (nseat_over):
    - Approach 1 : 5 (initially, varying up to 15 in simulations)
    - Approach 2 : Up to 20 seats, with the option to not sell on certain days
- Show-Up Probability for Coach (showup_coa): 95%
- Show-Up Probability for First-Class (showup_1st): 97%
- Cost to Bump a Coach Passenger to First-Class (bump_1st): $50
- Cost to Bump a Coach Passenger off the Plane (bump_out): $425
- Coach Ticket Prices (price_coa): $300, $350
- First-Class Ticket Prices (price_1st): $425, $500
- Demand Probability for Coach Tickets (dema_coa): 65%, 30%
- Demand Probability for First-Class Tickets (dema_1st): 8%, 4%
- Annual Discount Rate (r_year): 17%
- Daily Discount Factor (r): 1 / (1 + 0.17 / 365)
- Extra Demand for Coach when First-Class is Sold Out (extra_d): 3%


Parameters

```
ndays = 365
T = ndays

nseat_coa = 100
nseat_1st = 20
nseat_over = 5

showup_coa = 0.95
showup_1st = 0.97

bump_1st = 50
bump_out = 425

price_coa = [300,350]
price_1st = [425,500]


dema_coa = [0.65,0.3]
dema_1st = [0.08,0.04]

r_year = 0.17
r = 1/(1+r_year/365)

extra_d = 0.03
```

2. Approach

2.1. Approach 1: Fixed Overbooking Limit Method

2.1.1. Overview

In our exploration of optimal overbooking strategies, the fixed overbooking limit method serves as a foundational approach. This method involves setting a predetermined threshold for the number of coach tickets that can be oversold, a range that spans from 5 to 15 seats beyond the aircraft's capacity. By employing dynamic programming, we aim to uncover the optimal pricing policy for both coach and first-class tickets, a policy that seeks to maximize the airline's profits while adhering to the constraints of the overbooking limit.

This approach provides a structured framework for evaluating the trade-offs inherent in overbooking. It allows us to assess the potential revenue gains against the risks and costs associated with bumping passengers. As we delve deeper into this analysis, we will juxtapose this method with a more flexible approach in the subsequent sections. The dynamic overbooking method with a no-sale option, to be discussed later, introduces an additional layer of complexity and adaptability to our strategy, enabling a more nuanced management of overbooking risks. Together, these approaches offer a comprehensive understanding of how strategic overbooking can be leveraged to enhance an airline's profitability while maintaining a positive customer experience.

2.1.2. Methodology

In this study, we aim to maximize the expected discounted profits for an airline by optimizing pricing and overbooking strategies. To achieve this, we first initialize the components of dynamic programming, including state variables, value function, and choice function, for an overbooking limit of 5 seats. This initial setup serves as the foundation for our analysis.

Moving forward, we extend our approach to consider overbooking limits ranging from 6 to 15 seats. We define a function that iterates through these overbooking scenarios, leveraging the previously defined components to systematically evaluate the impact of different overbooking policies on the airline's profits.

Let's delve into the details of our methodology and explore how these components are defined and utilized in our dynamic programming approach.

1) Components of Dynamic Programming

This section covers the initial steps taken to define crucial functions and parameters for our model, setting the stage for the optimization process.

State Variables :

Our optimization process begins by defining the state variables crucial for decision-making in the dynamic environment.

```
s_coa_values = np.arange(nseat_coa+nseat_over+1) # number of coach seats sold
s_1st_values = np.arange(nseat_1st+1) # number of 1st_class seats sold

t_values = np.arange(ndays+1) # day
```

- s_coa_values tracks the potential number of coach seats sold, ranging from zero to the maximum allowable oversold capacity.
- s_1st_values accounts for the first-class seats sold, a smaller and more premium range due to no overbooking policy in this class.
- t_values represents the time dimension, capturing the number of days until departure and serving as our temporal decision-making axis.

This careful setup allows us to project the system's state across all possible configurations, laying the groundwork for our dynamic programming approach.

Value and Choice Function:

In the core of our optimization algorithm, we initialize two critical constructs, the value function V and the choice function U, using numpy arrays. These constructs are pivotal for navigating through the decision-making process across various states and time periods.

```
V = np.zeros((nseat_1st+1,nseat_coa+nseat_over+1,ndays+1)) #value function
U = np.zeros((nseat_1st+1,nseat_coa+nseat_over+1,ndays+1)) # choice function
```

Value Function `V`

This three-dimensional numpy array represents the maximum expected discounted profit for each possible state and time.The dimensions of `V` are determined by:

- The number of first-class seats sold (nseat_1st+1) to account for all possible sales outcomes, including no seats sold.
- The total potential coach seats sold, adjusted for overbooking (nseat_coa+nseat_over+1), to encapsulate scenarios from no seats sold up to the maximum allowed by the overbooking policy.
- The time dimension (ndays+1), capturing every day from the start of the selling period to the day of departure.

Each cell within this array (V[s_1st, s_coa, t]) is meant to store the best achievable profit at a given state (s_1st, s_coa) and time (t).

Choice Function `U`

This array tracks the optimal strategy (pricing and overbooking decisions) corresponding to each state and time. It mirrors the structure of the value function but instead records the best action to take:

- The indices of this array directly correlate with those of `V`, signifying that for any given state and time, `U` points to the choice that maximizes future profits as calculated in `V`.
- The values stored in U are decision codes, each referring to a combination of pricing strategies, including the option to halt coach ticket sales. These decision codes, which our team has meticulously defined, guide the airline's pricing and overbooking strategies at each state and time in the optimization process. The decision codes and their corresponding strategies are as follows:

Decision Code	Strategy Description
0	LL - Low price first class & low price coach
1	HL - High price first class & low price coach
2	LH - Low price first class & high price coach
3	HH - High price first class & high price coach
4	L0 - Low price first class & no sale coach
5	H0 - High price first class & no sale coach
6	0L - Sold out first class & low price coach
7	0H - Sold out first class & high price coach
8	00 - Sold out first class & no sale coach

These arrays are meticulously crafted to capture all potential outcomes and strategic options, underpinning the dynamic programming approach to optimizing our decision-making process. By iterating over these structures, we systematically explore and record the optimal decisions and their resultant profits across the entire decision space.

Terminal Function:

The Terminal Function is designed to estimate the expected costs associated with overbooking on the final day before departure (day 365). This involves calculating the costs incurred from potentially having to bump passengers to first class or off the plane due to overbooking. The function iterates through all possible scenarios based on the number of tickets sold in each class (s_1st and s_coa) and the number of passengers who actually show up (n_1st and n_coa).

```
for s_1st in s_1st_values:
    for s_coa in s_coa_values:
        exp_cost = 0

        for n_1st in range(s_1st+1):
            for n_coa in range(s_coa+1):

                prob_show_1st = binom.pmf(n_1st, s_1st, showup_1st)
                prob_show_coa = binom.pmf(n_coa, s_coa, showup_coa)

                n_ava_1st = nseat_1st-n_1st

                if n_coa>nseat_coa:
                    if (n_coa-nseat_coa)<=n_ava_1st:
                        cost = (n_coa-nseat_coa)*bump_1st
                    else:
                        cost = n_ava_1st*bump_1st+(n_coa-nseat_coa-n_ava_1st)*bump_out
                else:
                    cost=0
                exp_cost += cost*prob_show_1st*prob_show_coa
        V[s_1st,s_coa,T] = -exp_cost
```

Iterating Over Possible Scenarios

For each combination of sold seats in first class (s_1st) and coach (s_coa), we initialize the expected cost (exp_cost) to zero. We then consider all possible combinations of passengers showing up in each class.

Calculating Probabilities
We use the binomial distribution to calculate the probability of a given number of passengers showing up (prob_show_1st and prob_show_coa) based on the tickets sold and the historical show-up rates (showup_1st and showup_coa).

Cost Calculation Based on Overbooking
For each scenario, we evaluate the need for bumping passengers based on the available seats:

- First-Class Sold Out: If s_1st equals the total number of first-class seats (nseat_1st), all first-class seats are sold out.
- If coach is also sold out (s_coa == nseat_coa + nseat_over), there are no more seats to sell, and no additional cost is incurred.
- If there are seats available in coach, we calculate the potential costs of bumping passengers to first-class or off the plane, depending on the excess number of coach passengers and the availability of first-class seats.
- First-Class Not Sold Out: If there are unsold first-class seats, we consider the costs associated with bumping coach passengers to first-class or off the plane, similar to the above scenario.

Expected Cost Accumulation
For each combination of passengers showing up, we calculate the cost incurred from bumping passengers. This cost is then weighted by the probability of that particular scenario occurring (prob_show_1st * prob_show_coa) and added to the total expected cost (exp_cost).

Updating the Value Function
Finally, the expected cost for each scenario is stored in the value function V at the corresponding state (s_1st, s_coa) and time (T), representing the cost to the airline on the final day before departure.

Bellman Equation:

The Bellman equation forms the backbone of our dynamic programming approach, enabling us to determine the optimal pricing and overbooking strategies. We start by setting up iterations for each day (t), each possible number of first-class seats sold (s_1st), and each possible number of coach seats sold (s_coa). Within this framework, we consider four main scenarios:

- Both Classes Sold Out: No additional revenue can be generated.
- First-Class Sold Out, Coach Available: Additional revenue can be generated from selling coach seats.
- First-Class Available, Coach Sold Out: Additional revenue can be generated from selling first-class seats.
- Both Classes Available: Revenue can be generated from selling seats in both classes.

Detailed Case Analysis with Code Snippets:

a) Both Classes Sold Out

If all seats in both first-class and coach are sold out, no further sales can occur. The value function `V` is updated to reflect the discounted future value, and the choice function `U` is set to indicate a sold-out scenario (Decision Code = 8)

```
for t in reversed(range(ndays)):
    for s_1st in s_1st_values:
        if s_1st==nseat_1st:
            for s_coa in (s_coa_values):
                if s_coa==(nseat_coa+nseat_over):
                    V[s_1st,s_coa,t] = 0 + r*V[s_1st,s_coa,t+1]
                    U[s_1st,s_coa,t] = 8
```

b) First-Class Sold Out, Coach Available

When first-class is sold out but coach seats are available, we calculate the future value for each pricing strategy in coach.

```
                else:
                    v_coa_1st = [price_coa[0]*(dema_coa[0]+extra_d)+r*(V[s_1st,s_coa+1,t+1]*(dema_coa[0]+extra_d)+
                                                                   V[s_1st,s_coa,t+1]*(1-dema_coa[0]-extra_d)),

                                 price_coa[1]*(dema_coa[1]+extra_d)+r*(V[s_1st,s_coa+1,t+1]*(dema_coa[1]+extra_d)+
                                                                   V[s_1st,s_coa,t+1]*(1-dema_coa[1]-extra_d))]

                    V[s_1st,s_coa,t] = np.max(v_coa_1st)
                    U[s_1st,s_coa,t] = 6 + np.argmax(v_coa_1st)
```

The array v_coa_1st stores these future values, which consider both the immediate revenue from ticket sales and the discounted future value:

For the low price coach strategy, the first element in the v_coa_1st array calculates the expected revenue from selling a coach ticket at a low price (price_coa[0]), along with the discounted future value

- The term price_coa[0] * (dema_coa[0] + extra_d) represents the immediate revenue from selling a coach ticket at a low price, considering the increased demand (extra_d) when first-class is sold out.
- The term r * (V[s_1st, s_coa + 1, t + 1] * (dema_coa[0] + extra_d) + V[s_1st, s_coa, t + 1] * (1 - dema_coa[0] - extra_d)) represents the discounted future value, taking into account two possibilities:
    - V[s_1st, s_coa + 1, t + 1] * (dema_coa[0] + extra_d): The future value if a coach ticket is sold (increasing the number of sold coach seats to s_coa + 1), weighted by the probability of this event (dema_coa[0] + extra_d).
    - V[s_1st, s_coa, t + 1] * (1 - dema_coa[0] - extra_d): The future value if no additional coach ticket is sold, weighted by the probability of this event (1 - dema_coa[0] - extra_d).

The second element in the v_coa_1st array follows a similar structure, calculating the expected revenue and discounted future value for selling a coach ticket at a high price (price_coa[1]):

- The term price_coa[1] * (dema_coa[1] + extra_d) represents the immediate revenue from selling a coach ticket at a high price, considering the increased demand when first-class is sold out.
- The discounted future value calculation is analogous to the low price strategy, with the probability of selling a ticket adjusted for the high price (dema_coa[1] + extra_d).
- The value 4 is added to the index of the maximum value in v_coa_1st to indicate that the optimal strategy involves first-class ticket sales (Decision Code = 4~5)

In both strategies, the discounted future value component accounts for the uncertainty in ticket sales and their impact on future revenue, ensuring that the optimization process considers both immediate gains and long-term outcomes.

c) First-Class Available, Coach Sold Out

When the coach is sold out but first-class seats are available, we focus on the revenue from selling first-class tickets. The future value calculations in v_coa_1st now revolve around first-class pricing strategies:

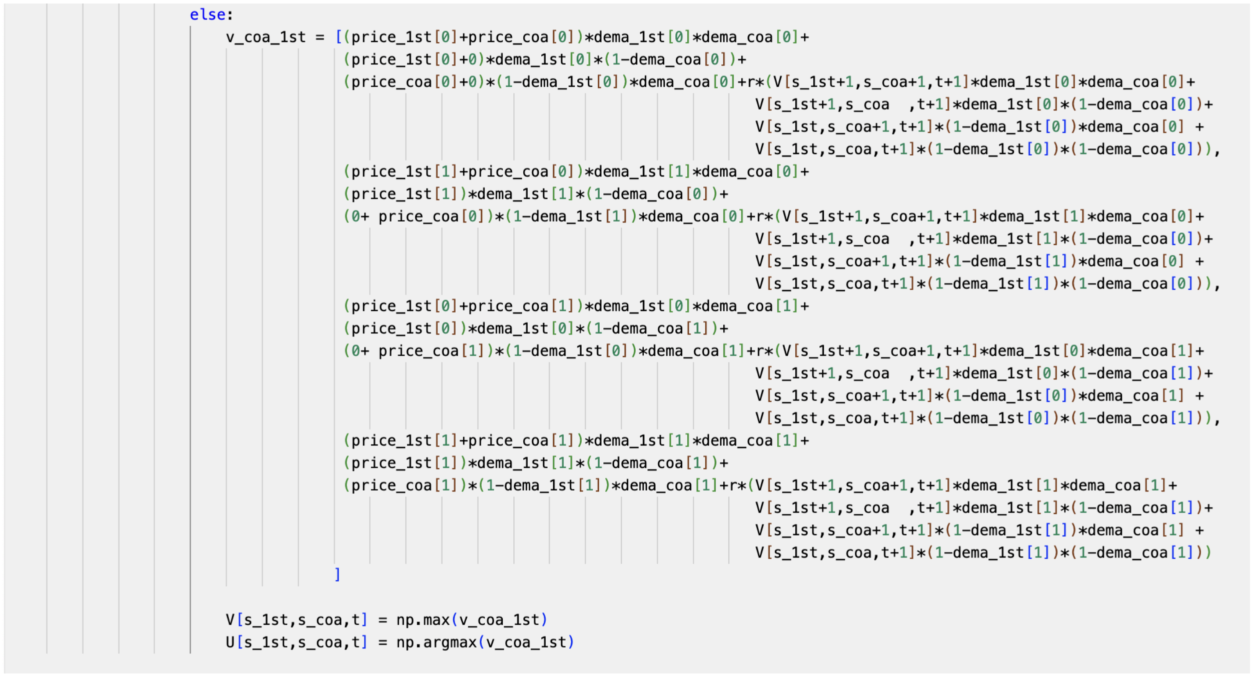
```
            else:
                for s_coa in (s_coa_values):
                    if s_coa==(nseat_coa+nseat_over):
                        v_coa_1st = [price_1st[0]*dema_1st[0]+r*(V[s_1st+1,s_coa,t+1]*dema_1st[0]+
                                                                 V[s_1st,s_coa,t+1]*(1-dema_1st[0])),
                                     price_1st[1]*dema_1st[1]+r*(V[s_1st+1,s_coa,t+1]*dema_1st[1]+
                                                                 V[s_1st,s_coa,t+1]*(1-dema_1st[1]))]

                        V[s_1st,s_coa,t] = np.max(v_coa_1st)
                        U[s_1st,s_coa,t] = np.argmax(v_coa_1st)+4
```

- The first element calculates the expected revenue and discounted future value for selling a first-class ticket at a low price (price_1st[0]), similar to the approach described earlier for coach tickets.
- The second element does the same for selling a first-class ticket at a high price (price_1st[1]).
- The value `6` is added to the index of the maximum value in v_coa_1st to indicate that the optimal strategy involves first-class ticket sales (Decision Code = 6~7)

d) Both Classes Available

When both first-class and coach seats are available, the array v_coa_1st stores the future values for each combination of pricing strategies. The following combinations of pricing strategies are considered

- Low Price for Both First-Class and Coach
- High Price for First-Class, Low Price for Coach
- Low Price for First-Class, High Price for Coach
- High Price for Both First-Class and Coach



```
else:
    v_coa_1st = [(price_1st[0]+price_coa[0])*dema_1st[0]*dema_coa[0]+
                 (price_1st[0]+0)*dema_1st[0]*(1-dema_coa[0])+
                 (price_coa[0]+0)*(1-dema_1st[0])*dema_coa[0]+r*(V[s_1st+1,s_coa+1,t+1]*dema_1st[0]*dema_coa[0]+
                                                               V[s_1st+1,s_coa  ,t+1]*dema_1st[0]*(1-dema_coa[0])+
                                                               V[s_1st,s_coa+1,t+1]*(1-dema_1st[0])*dema_coa[0] +
                                                               V[s_1st,s_coa,t+1]*(1-dema_1st[0])*(1-dema_coa[0])),
                 (price_1st[1]+price_coa[0])*dema_1st[1]*dema_coa[0]+
                 (price_1st[1])*dema_1st[1]*(1-dema_coa[0])+
                 (0+ price_coa[0])*(1-dema_1st[1])*dema_coa[0]+r*(V[s_1st+1,s_coa+1,t+1]*dema_1st[1]*dema_coa[0]+
                                                               V[s_1st+1,s_coa  ,t+1]*dema_1st[1]*(1-dema_coa[0])+
                                                               V[s_1st,s_coa+1,t+1]*(1-dema_1st[1])*dema_coa[0] +
                                                               V[s_1st,s_coa,t+1]*(1-dema_1st[1])*(1-dema_coa[0])),
                 (price_1st[0]+price_coa[1])*dema_1st[0]*dema_coa[1]+
                 (price_1st[0])*dema_1st[0]*(1-dema_coa[1])+
                 (0+ price_coa[1])*(1-dema_1st[0])*dema_coa[1]+r*(V[s_1st+1,s_coa+1,t+1]*dema_1st[0]*dema_coa[1]+
                                                               V[s_1st+1,s_coa  ,t+1]*dema_1st[0]*(1-dema_coa[1])+
                                                               V[s_1st,s_coa+1,t+1]*(1-dema_1st[0])*dema_coa[1] +
                                                               V[s_1st,s_coa,t+1]*(1-dema_1st[0])*(1-dema_coa[1])),
                 (price_1st[1]+price_coa[1])*dema_1st[1]*dema_coa[1]+
                 (price_1st[1])*dema_1st[1]*(1-dema_coa[1])+
                 (price_coa[1])*(1-dema_1st[1])*dema_coa[1]+r*(V[s_1st+1,s_coa+1,t+1]*dema_1st[1]*dema_coa[1]+
                                                               V[s_1st+1,s_coa  ,t+1]*dema_1st[1]*(1-dema_coa[1])+
                                                               V[s_1st,s_coa+1,t+1]*(1-dema_1st[1])*dema_coa[1] +
                                                               V[s_1st,s_coa,t+1]*(1-dema_1st[1])*(1-dema_coa[1]))
                ]

    V[s_1st,s_coa,t] = np.max(v_coa_1st)
    U[s_1st,s_coa,t] = np.argmax(v_coa_1st)
```

- **Low Price for Both First-Class and Coach**

  Immediate Revenue Components:
  - (price_1st[0] + price_coa[0]) * dema_1st[0] * dema_coa[0]: Revenue from selling both first-class and coach tickets at low prices when both types of tickets are sold.
  - (price_1st[0] + 0) * dema_1st[0] * (1 - dema_coa[0]): Revenue from selling only a first-class ticket at a low price when a coach ticket is not sold.
  - (price_coa[0] + 0) * (1 - dema_1st[0]) * dema_coa[0]: Revenue from selling only a coach ticket at a low price when a first-class ticket is not sold.

  Discounted Future Value Components:
  - V[s_1st + 1, s_coa + 1, t + 1] * dema_1st[0] * dema_coa[0]: Future value if both a first-class and a coach ticket are sold, weighted by the probability of this event.
  - V[s_1st + 1, s_coa, t + 1] * dema_1st[0] * (1 - dema_coa[0]): Future value if only a first-class ticket is sold, weighted by the probability of this event.
  - V[s_1st, s_coa + 1, t + 1] * (1 - dema_1st[0]) * dema_coa[0]: Future value if only a coach ticket is sold, weighted by the probability of this event.
  - V[s_1st, s_coa, t + 1] * (1 - dema_1st[0]) * (1 - dema_coa[0]): Future value if neither a first-class nor a coach ticket is sold, weighted by the probability of this event.

- **High Price for First-Class, Low Price for Coach**
    - Immediate Revenue: (price_1st[1] + price_coa[0]) * dema_1st[1] * dema_coa[0] calculates the revenue from selling first-class tickets at a high price and coach tickets at a low price, considering their respective demand probabilities.
    - Discounted Future Value: The calculation is similar to the first strategy, but with adjustments to the prices and demand probabilities to reflect the high price for first-class and the low price for coach.
- **Low Price for First-Class, High Price for Coach**
    - Immediate Revenue: (price_1st[0] + price_coa[1]) * dema_1st[0] * dema_coa[1] calculates the revenue from selling first-class tickets at a low price and coach tickets at a high price, considering their respective demand probabilities.
    - Discounted Future Value: This follows the same pattern as previous strategies, with the future value calculation considering the impact of these specific pricing decisions on future states.
- **High Price for Both First-Class and Coach**
    - Immediate Revenue: (price_1st[1] + price_coa[1]) * dema_1st[1] * dema_coa[1] calculates the revenue from selling both first-class and coach tickets at high prices, considering their respective demand probabilities.
    - Discounted Future Value: The calculation mirrors that of other strategies, accounting for the state where tickets in both classes are sold at high prices, with the future value discounted back to the present.

After calculating the future values for each pricing strategy combination in the array v_coa_1st, the maximum value is selected and assigned to the value function V to represent the optimal expected profit for the current state and time. Correspondingly, the index of this maximum value is recorded in the choice function U, indicating the optimal pricing strategy to be employed for achieving this profit.

2.1.3. Result

1) Overbooking Limits of 5

Expected Maximum Profits:

Our dynamic programming approach has yielded comprehensive insights into the optimal pricing and overbooking strategies for maximizing airline profits. By reshaping the value function V and the choice function U into dataframes and exporting them to CSV files, we have created a structured representation of the expected profits and corresponding optimal strategies across different states and time periods.

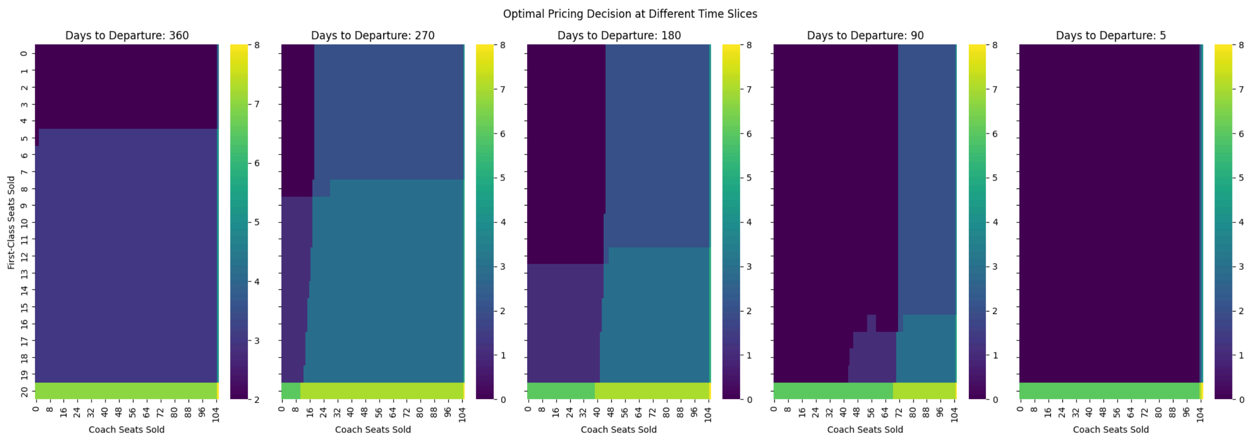
The analysis of the question1_V.csv file reveals the progression of expected profits as we move closer to the flight's departure date. It shows how the value function evolves, reflecting the accumulation of revenue and the impact of strategic decisions on future profits. This data is crucial for understanding the financial implications of different overbooking limits and pricing strategies.

The calculated expected maximum profit of **$41,886.16** at V[0,0,0] exemplifies the successful application of our dynamic programming approach, encapsulating the culmination of an array of optimal strategies devised for different booking scenarios. "This figure stands as the financial benchmark for the airline's potential revenue," showcasing the comprehensive analysis conducted through the value function V, and underscores the effectiveness of our strategy optimization over the entire booking horizon.

Likewise, question1_U.csv maps the optimal pricing strategies for each state and time, highlighting patterns in pricing decisions, such as when to adjust prices or halt sales to manage overbooking risks.

Dynamic Pricing Strategies as Departure Nears:

As we decode the strategies through the lens of the question1_U.csv, we observe a tactical narrative unfold.



360 Days to Departure: The dominant strategy is to start with high prices for both classes (3 - HH), likely reflecting a traditional approach where airlines begin with higher prices, assuming early bookers are less price-sensitive.

270 Days to Departure: While the HH strategy (3) continues to dominate, there is an increase in the LH strategy (2), signaling a pivot to lower first-class prices to stimulate sales in this category. The introduction of strategies LL (0) and HL (1) indicates a diversification in pricing strategies to encourage demand across both classes.

180 Days to Departure: An increase in strategies LL (0) and HL (1) suggests more aggressive pricing adjustments. The presence of strategy OL (6) when first class is sold out represents a third of the selection, pointing to a consistent low-price coach strategy after some time has passed, reflecting the need to ensure coach occupancy as the flight date draws closer.

90 Days to Departure: Strategy 0 (LL) becomes more prevalent, showing an aggressive strategy to increase seat occupancy as the flight date nears. Strategy 6 (OL) becomes more common than strategy 7 (OH) when first class is sold out, reflecting a shift to a low-price coach strategy to fill the plane

5 Days to Departure: Strategy 0 (LL) is mostly adopted, which is consistent with the last-minute effort to maximize occupancy. When first class is sold out, the preferred strategy is 6 (OL), which maintains the low-price coach strategy.

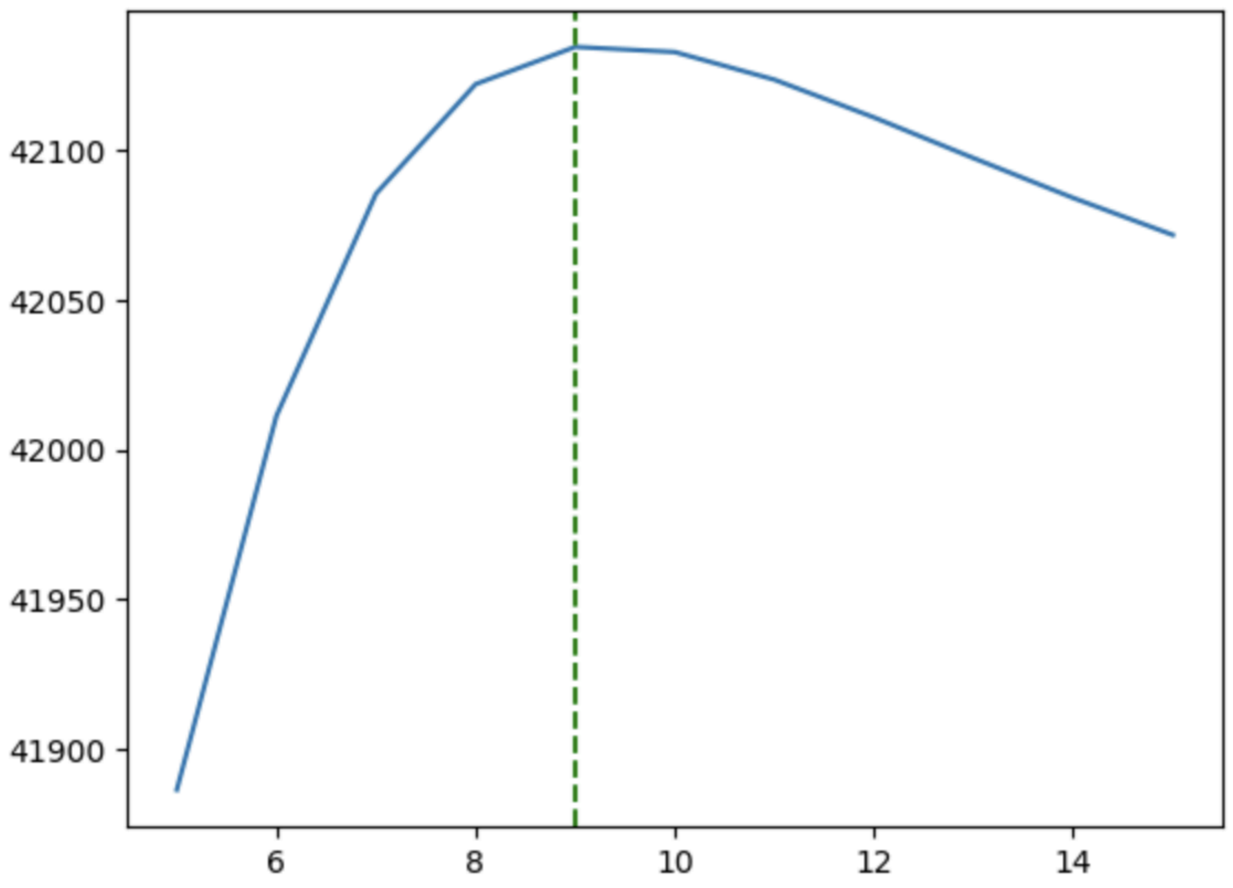
With this framework, the transition towards lower prices for both classes as departure approaches reflects a common airline strategy to ensure as full a flight as possible, adjusting prices according to the remaining time and occupancy. Initially high prices target less price-sensitive passengers, but as the departure date nears, the emphasis shifts to filling all remaining seats to avoid the loss that comes with flying an aircraft that's not full.

2) Overbooking Limits of 6 to 15

In order to test a variety of different overbooking limits, we consolidated all the aforementioned code for the state variables, value function, terminal cost, and bellman equation into a function defined as run_get_profit. We then iterated the function over several possible overbooking scenarios with limits ranging from 6 to 15.

```
profit = []
for i in range(5,16):
    V,U = run_get_profit(i)
    profit+= [V[0,0,0]]

profit
```

The results of this loop showed the outcomes of different levels of overbooking permitted with the purpose of highlighting the most profitable overbooking limit given our assumed probabilities.



As shown by the graph above, an overbooking limit of 9 tickets was the most profitable in the given range of scenarios, resulting in a $42,134.62 profit. Increasing the limit on overbooking initially increased calculated profits drastically moving upward from the initial 5 ticket limit, but soon found diminishing returns as the limit reached 9 tickets. Profit slowly declined as the limit went to 12 tickets, with somewhat more pronounced loss upon reaching 15 tickets, suggesting that in simulations there is a potential range of optimal overbooking limits.

2.2. Approach 2: Dynamic Overbooking with No-Sale Option

2.2.1. Overview

In contrast to Approach 1, this strategy employs a third choice on each day when deciding ticket prices where the airline can choose to sell no coach tickets, forcing demand to 0 on any given day. This level of flexibility enables the airline to limit overselling not just based upon how many tickets have been purchased, but also factoring in the remaining number of days until flight. Building off our prior dynamic programming framework, we aim to account for a more proactive and adaptable management approach to travel bookings for flights in order to maximize the airline's profits while still adhering to the constraints of the overbooking limit as well as maintaining quality service for first class passengers.

2.2.2. Methodology

In this approach, we are expanding upon Approach 1 to account for a No-Sell strategy where a third choice is considered where no coach tickets are sold on any given day. This strategy is not implemented for first class as they are the airline's most valued customers and are provided with guaranteed access to tickets at whatever the day's price is slated.

The No-Sell strategy utilizes the same code as Approach 1 in most regards, specifically the initial parameters, state variables, value function, and terminal costs. The key difference between the approaches lies within the Bellman equation, where the third choice of not selling coach tickets is accounted for as two additional scenarios.

Bellman Equation:

The Bellman equation consists of 4 situations: Both Classes Sold Out, First Class Sold Out with Coach Available, First Class Available with Coach Sold Out, and Both Classes Available. The No-Sell strategy only impacts whenever First Class Sold Out with Coach Available and when Both Classes Available

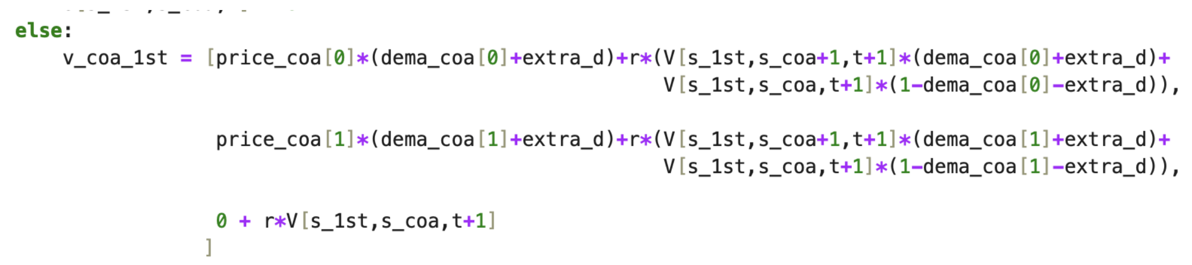
- First Class Sold Out with Coach Available

  In this case we will have the same 2 scenarios outlined in Approach 1:

    - No tickets sold for First-Class, Low Price for Coach
    - No tickets sold for First-Class, High Price for Coach

  But we also add the decision of not selling any seat in coach

    - No tickets sold for First-Class and not selling ticket for Coach



```
else:
    v_coa_1st = [price_coa[0]*(dema_coa[0]+extra_d)+r*(V[s_1st,s_coa+1,t+1]*(dema_coa[0]+extra_d)+
                                                          V[s_1st,s_coa,t+1]*(1-dema_coa[0]-extra_d)),

                 price_coa[1]*(dema_coa[1]+extra_d)+r*(V[s_1st,s_coa+1,t+1]*(dema_coa[1]+extra_d)+
                                                          V[s_1st,s_coa,t+1]*(1-dema_coa[1]-extra_d)),

                 0 + r*V[s_1st,s_coa,t+1]
                ]
```

No tickets sold for First-Class and not selling ticket for Coach

This scenario is similar to when all seats in both first-class and coach are sold out. The value function `V` is updated to reflect the discounted future value, and the choice function `U` is set to indicate a sold-out scenario (Decision Code = 8)

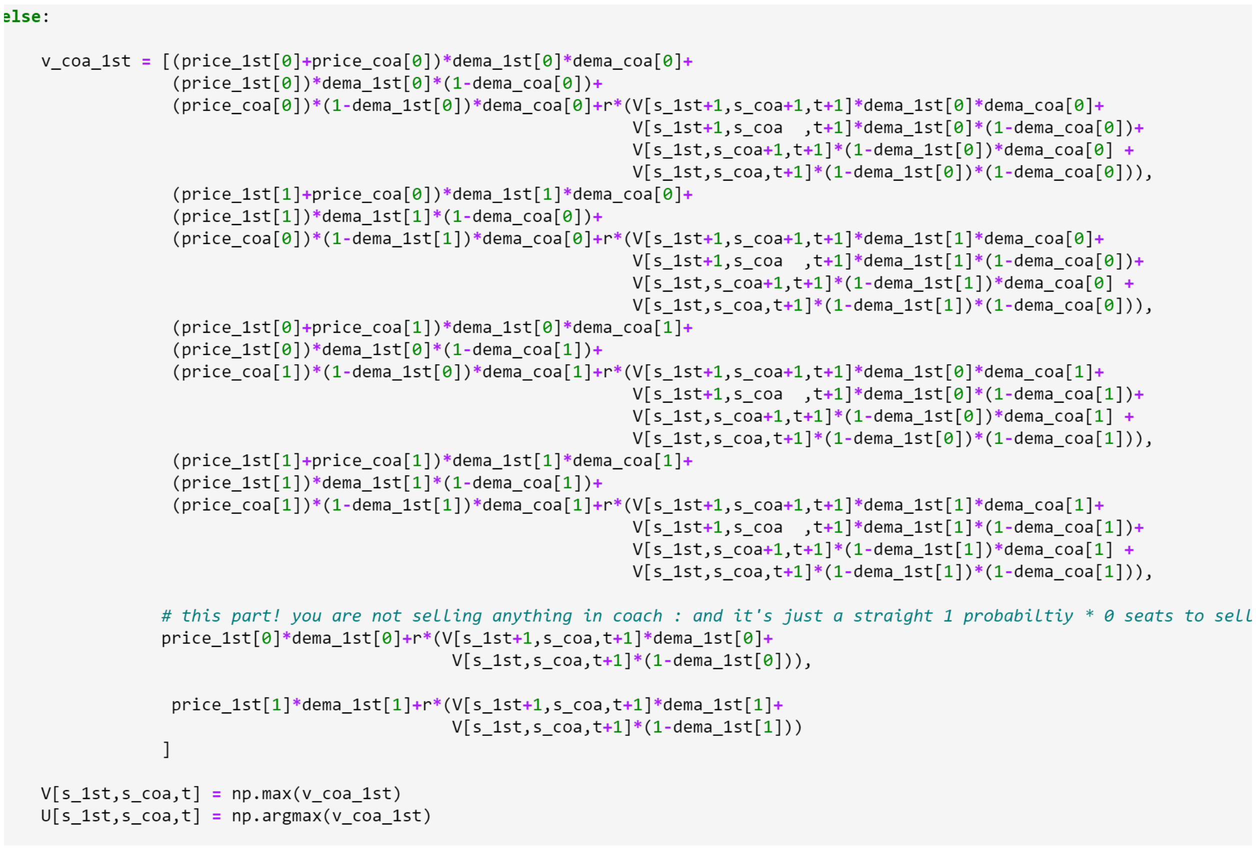
- Both Classes Available

  This scenario still has the four pricing strategies outlined in Approach 1:

    - Low Price for Both First-Class and Coach
    - High Price for First-Class, Low Price for Coach
    - Low Price for First-Class, High Price for Coach
    - High Price for Both First-Class and Coach

But there are two additional strategies now included to account for not selling coach seats on any given day, forcing demand for coach to zero while still enabling first class purchases:

- ○ Low Price for First-Class and not selling in coach
- ○ High Price for First-Class and not selling in coach



```
else:

    v_coa_1st = [(price_1st[0]+price_coa[0])*dema_1st[0]*dema_coa[0]+
                 (price_1st[0])*dema_1st[0]*(1-dema_coa[0])+
                 (price_coa[0])*(1-dema_1st[0])*dema_coa[0]+r*(V[s_1st+1,s_coa+1,t+1]*dema_1st[0]*dema_coa[0]+
                                                              V[s_1st+1,s_coa  ,t+1]*dema_1st[0]*(1-dema_coa[0])+
                                                              V[s_1st,s_coa+1,t+1]*(1-dema_1st[0])*dema_coa[0] +
                                                              V[s_1st,s_coa,t+1]*(1-dema_1st[0])*(1-dema_coa[0])),
                 (price_1st[1]+price_coa[0])*dema_1st[1]*dema_coa[0]+
                 (price_1st[1])*dema_1st[1]*(1-dema_coa[0])+
                 (price_coa[0])*(1-dema_1st[1])*dema_coa[0]+r*(V[s_1st+1,s_coa+1,t+1]*dema_1st[1]*dema_coa[0]+
                                                              V[s_1st+1,s_coa  ,t+1]*dema_1st[1]*(1-dema_coa[0])+
                                                              V[s_1st,s_coa+1,t+1]*(1-dema_1st[1])*dema_coa[0] +
                                                              V[s_1st,s_coa,t+1]*(1-dema_1st[1])*(1-dema_coa[0])),
                 (price_1st[0]+price_coa[1])*dema_1st[0]*dema_coa[1]+
                 (price_1st[0])*dema_1st[0]*(1-dema_coa[1])+
                 (price_coa[1])*(1-dema_1st[0])*dema_coa[1]+r*(V[s_1st+1,s_coa+1,t+1]*dema_1st[0]*dema_coa[1]+
                                                              V[s_1st+1,s_coa  ,t+1]*dema_1st[0]*(1-dema_coa[1])+
                                                              V[s_1st,s_coa+1,t+1]*(1-dema_1st[0])*dema_coa[1] +
                                                              V[s_1st,s_coa,t+1]*(1-dema_1st[0])*(1-dema_coa[1])),
                 (price_1st[1]+price_coa[1])*dema_1st[1]*dema_coa[1]+
                 (price_1st[1])*dema_1st[1]*(1-dema_coa[1])+
                 (price_coa[1])*(1-dema_1st[1])*dema_coa[1]+r*(V[s_1st+1,s_coa+1,t+1]*dema_1st[1]*dema_coa[1]+
                                                              V[s_1st+1,s_coa  ,t+1]*dema_1st[1]*(1-dema_coa[1])+
                                                              V[s_1st,s_coa+1,t+1]*(1-dema_1st[1])*dema_coa[1] +
                                                              V[s_1st,s_coa,t+1]*(1-dema_1st[1])*(1-dema_coa[1])),

                # this part! you are not selling anything in coach : and it's just a straight 1 probabiltiy * 0 seats to sell
                price_1st[0]*dema_1st[0]+r*(V[s_1st+1,s_coa,t+1]*dema_1st[0]+
                                            V[s_1st,s_coa,t+1]*(1-dema_1st[0])),

                 price_1st[1]*dema_1st[1]+r*(V[s_1st+1,s_coa,t+1]*dema_1st[1]+
                                            V[s_1st,s_coa,t+1]*(1-dema_1st[1]))
                ]

    V[s_1st,s_coa,t] = np.max(v_coa_1st)
    U[s_1st,s_coa,t] = np.argmax(v_coa_1st)
```

High Price for First-Class

- ○ Immediate Revenue: (price_1st[1] * dema_1st[1]) calculates the revenue from selling first-class tickets at a high price, considering the respective demand probabilities.
- ○ Discounted Future Value: The calculation is similar to the prior 4 scenarios, but with adjustments to the prices and demand probabilities to reflect the high price for first-class and exclusion of coach in the discounting.

Low Price for First-Class

- ○ Immediate Revenue: (price_1st[0] * dema_1st[0]) calculates the revenue from selling first-class tickets at a low price, considering the respective demand probabilities.
- ○ Discounted Future Value: This follows the same pattern as the above strategy, utilizing the lower price first class ticket's respective demand with the future value calculation considering the impact of these specific pricing decisions on future states.

2.2.3. Result

After calculating the future values for both the Approach 1 and Approach 2 No-Sell strategies, we found that the No-Sell strategy has a better expected profit than the best policy from Approach 1.

Expected profits

Policy 1 with 20 seats overbooked	Policy 1 with 9 seats overbooked	Policy 2 with 20 seats overbooked
$42028.74	$42139.89	$42134.63

3. Overbooking Strategies through Forward Simulation

3.3.1. Overview

Now that we have solved the problem backwards, we aim to evaluate the optimal overbooking policy for an airline through simulation under three distinct scenarios:

- The first policy (without a no-sale option for coach tickets) with a maximum overbooking of 20 seats.
- The first policy with an optimal overbooking of 9 seats, as determined from our initial analysis.

- The second policy (with a no-sale option for coach tickets) with a maximum overbooking of 20 seats.

For each scenario, we will investigate the following key metrics:
- The frequency of coach overbooking.
- The incidence of passenger removal from flights.
- The average cost associated with overbooking.
- The volatility of discounted profits.

These metrics will provide insights into the operational and financial implications of different overbooking strategies. By comparing the outcomes across the three scenarios, we aim to identify the best recommendation that maximizes revenue and profit while minimizing volatility and the number of passenger kickoffs. This analysis will enable us to make informed decisions on the optimal overbooking policy that balances profitability with customer satisfaction.

3.3.2. Methodology

Setting the Parameters:

The parameters are similar to the base one but we will define the number of seats available to overbook (nseat_over) later and we add two parameters parameters:

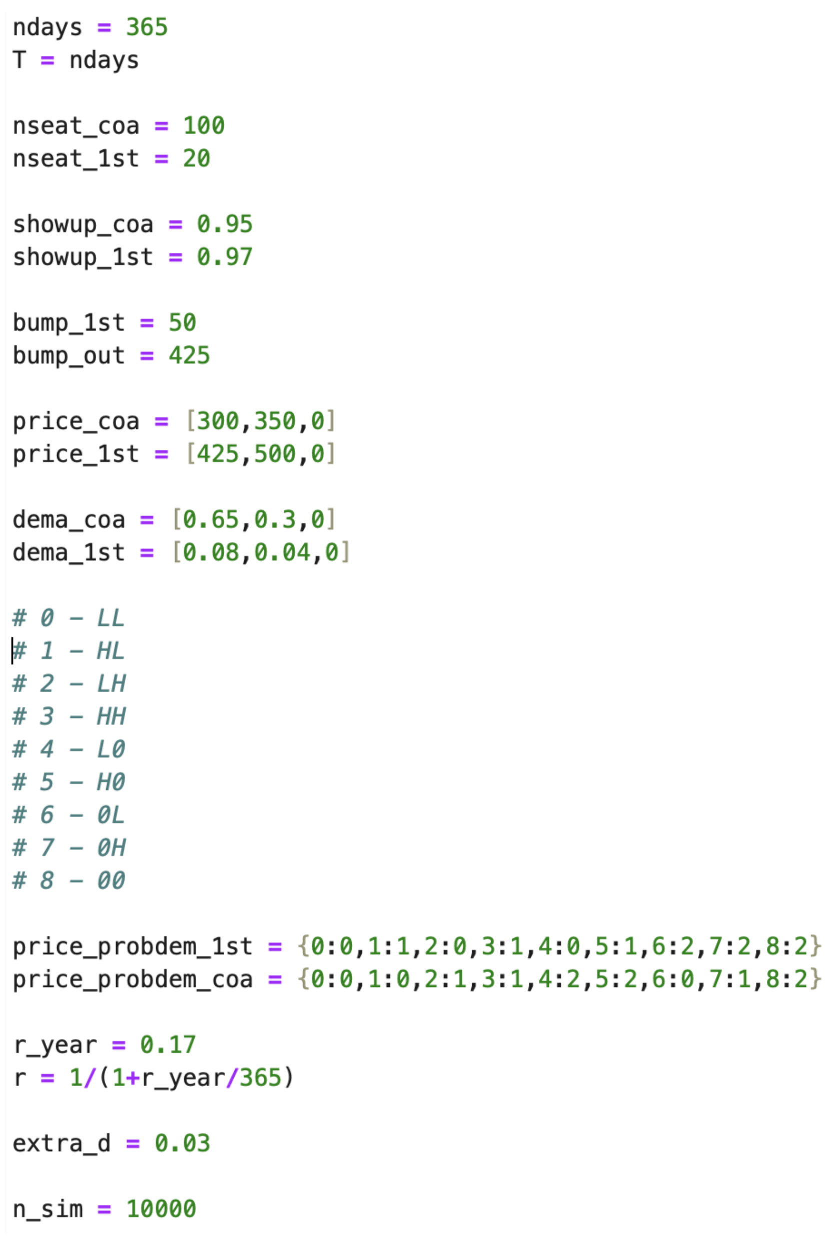
- number of simulations(n_sim)

```
n_sim = 10000
```

- Dictionary with the price/probability index for each of the decision

```
price_probdem_1st = {0:0,1:1,2:0,3:1,4:0,5:1,6:2,7:2,8:2}
price_probdem_coa = {0:0,1:0,2:1,3:1,4:2,5:2,6:0,7:1,8:2}
```

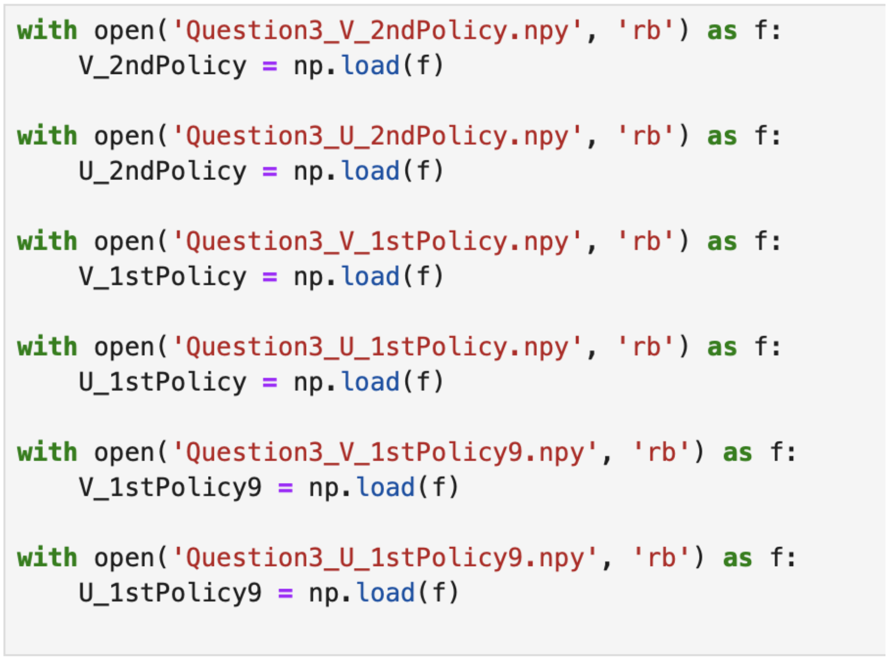
As a result we have the following parameters:



```
ndays = 365
T = ndays

nseat_coa = 100
nseat_1st = 20

showup_coa = 0.95
showup_1st = 0.97

bump_1st = 50
bump_out = 425

price_coa = [300,350,0]
price_1st = [425,500,0]

dema_coa = [0.65,0.3,0]
dema_1st = [0.08,0.04,0]

# 0 - LL
# 1 - HL
# 2 - LH
# 3 - HH
# 4 - L0
# 5 - H0
# 6 - 0L
# 7 - 0H
# 8 - 00

price_probdem_1st = {0:0,1:1,2:0,3:1,4:0,5:1,6:2,7:2,8:2}
price_probdem_coa = {0:0,1:0,2:1,3:1,4:2,5:2,6:0,7:1,8:2}

r_year = 0.17
r = 1/(1+r_year/365)

extra_d = 0.03

n_sim = 10000
```

Import optimal decision for each policy:

So on each time t we will randomly have demand for 1 seat depending in what is our number of seats sold in coach (s_coa) and in 1st class (s_1st), and if we are deciding not to sell coach, so then we will need to import the matrix of decisions (U) for policy to understand what decision to take given the randomness that the process involve.



```
with open('Question3_V_2ndPolicy.npy', 'rb') as f:
    V_2ndPolicy = np.load(f)

with open('Question3_U_2ndPolicy.npy', 'rb') as f:
    U_2ndPolicy = np.load(f)

with open('Question3_V_1stPolicy.npy', 'rb') as f:
    V_1stPolicy = np.load(f)

with open('Question3_U_1stPolicy.npy', 'rb') as f:
    U_1stPolicy = np.load(f)

with open('Question3_V_1stPolicy9.npy', 'rb') as f:
    V_1stPolicy9 = np.load(f)

with open('Question3_U_1stPolicy9.npy', 'rb') as f:
    U_1stPolicy9 = np.load(f)
```

Apply optimal decision in a random simulation:

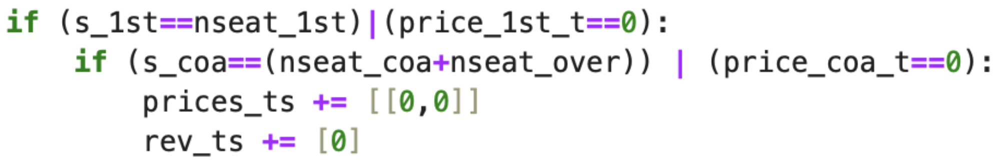
We create a function called "generate_model_overbook" where the input parameter is the U matrix for a given policy.

1) Initialize parameters of the simulation
- Number of seats sold at coach: n_coa=0
- Number of seats sold at 1st class: n_1st=0
- Total revenue: total_rev=0
- Array to save the revenue in each period: rev_ts=[]
- Array to save the pr in each period: prices_ts=[]
- Kick_off to track the cost related to kick off: Kick_off=0
- Cost related to Overbooking: overbook_cost=0
- How much seats are overbook: overbook_level=0

2) Run for the whole time window
- In each time period, we pick our decision depending on the time, the number of coach seats sold, and the number of 1st class sold. Given the decision we get the index for coach and 1st class, and with that the price and the probability of having a ticket sold.

- Then we run 4 conditionals and we calculate profit depending on each:

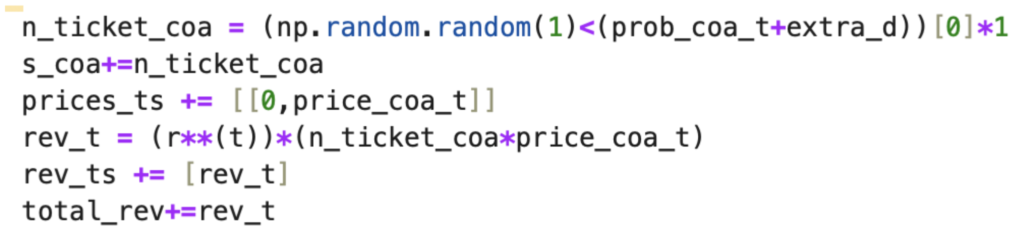
- No seats available for 1st class and coach or price equal to 0 that represents the decision of not selling a seat of coach.



```
if (s_1st==nseat_1st)|(price_1st_t==0):
    if (s_coa==(nseat_coa+nseat_over)) | (price_coa_t==0):
        prices_ts += [[0,0]]
        rev_ts += [0]
```

  Here there is no profit because we are not selling anything

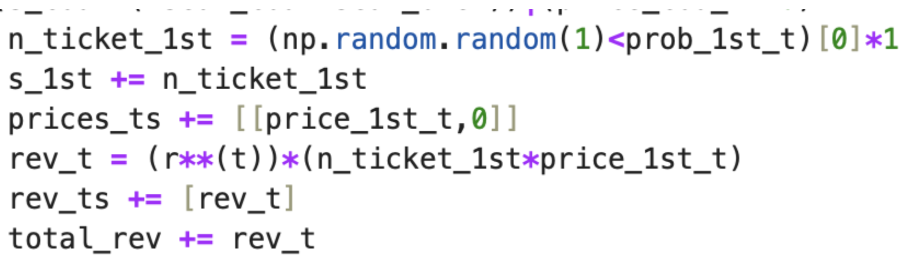
- No seats available for 1st class but seats available in coach and the decision of selling coach seats in that time period



```
n_ticket_coa = (np.random.random(1)<(prob_coa_t+extra_d))[0]*1
s_coa+=n_ticket_coa
prices_ts += [[0,price_coa_t]]
rev_t = (r**(t))*(n_ticket_coa*price_coa_t)
rev_ts += [rev_t]
total_rev+=rev_t
```

  Here, we generate a random number between 0 and 1. If it falls within the probability of selling a coach ticket at the selected price, plus an additional 3% due to the unavailability of 1st class tickets, we then update the number of coach seats sold. Revenue is updated accordingly by multiplying the number of tickets sold by the price, and then discounting it for the 't' periods, noting that the loop starts at 0. We record this revenue in an array and add it to the total revenue for that period.

- Seats available for 1st class and no seats available on coach or price equal to 0 that represents the decision of not selling a seat of coach.



```
n_ticket_1st = (np.random.random(1)<prob_1st_t)[0]*1
s_1st += n_ticket_1st
prices_ts += [[price_1st_t,0]]
rev_t = (r**(t))*(n_ticket_1st*price_1st_t)
rev_ts += [rev_t]
total_rev += rev_t
```

  Here, we generate a random number between 0 and 1, and if it falls within the probability of selling a ticket for 1st class at the selected price, we then update the number of 1st class seats sold. The revenue is updated by multiplying the number of tickets sold by the price, and then we discount it for the t periods. It is worth mentioning that t starts at 0 since the loop begins there. We record this revenue in an array and add the revenue for that period to the total revenue.

- ○ Seats available for 1st class and for coach and the decision of selling coach seats in that time period

```
n_ticket_coa = (np.random.random(1)<prob_coa_t)[0]*1
n_ticket_1st = (np.random.random(1)<prob_1st_t)[0]*1
s_coa+=n_ticket_coa
s_1st += n_ticket_1st
prices_ts += [[price_1st_t,price_coa_t]]
rev_t = (r**(t))*(n_ticket_coa*price_coa_t + n_ticket_1st*price_1st_t)
rev_ts += [rev_t]
total_rev+=rev_t
```

Here, we generate two random numbers between 0 and 1 (one for each type of seat), and if a number falls within the probability of selling a ticket at the selected price for the given ticket type, we update the number of 1st class and coach tickets sold. The revenue is updated by multiplying the tickets sold by their respective prices, and then we discount it for the t periods. It is worth mentioning that t starts at 0 since the loop begins there. We record this revenue in an array and add the revenue for that time to the total revenue

- After running these conditions from time 0 to 364, we calculate the terminal cost at time 365. For this, we simulate people arriving with the probability of each seat depending on the type of seat. After that, we run conditionals: if there is an overbook that can be sorted with 1st class seats, we multiply the extra seats by the cost of bumping people to 1st class. Otherwise, we bump all the people we can to 1st class and then bump out the remaining ones. Lastly, we update the overbook cost variable, the kick-off number of seats, and the overbook number of seats. We also update the total revenue and the arrays of revenues at time t and the prices array.

```
for i in range(s_coa+1):
    n_coa += (np.random.random(1)<showup_coa)[0]*1
for j in range(s_1st+1):
    n_1st += (np.random.random(1)<showup_1st)[0]*1

n_ava_1st = nseat_1st-n_1st
if n_coa>nseat_coa:
    overbook_level = n_coa-nseat_coa
    if (n_coa-nseat_coa)<=n_ava_1st:
        cost = (n_coa-nseat_coa)*bump_1st*(r**(T))
    else:
        cost = (n_ava_1st*bump_1st+(n_coa-nseat_coa-n_ava_1st)*bump_out)*(r**(T))
        kick_off=n_coa-nseat_coa-n_ava_1st
else:
    cost=0

overbook_cost = cost

rev_ts += [-cost]
prices_ts += [[0,0]]
total_rev -= cost
```

- The function returns rev_ts, prices_ts, overbook_level, kick_off, overbook_cost and total_rev.

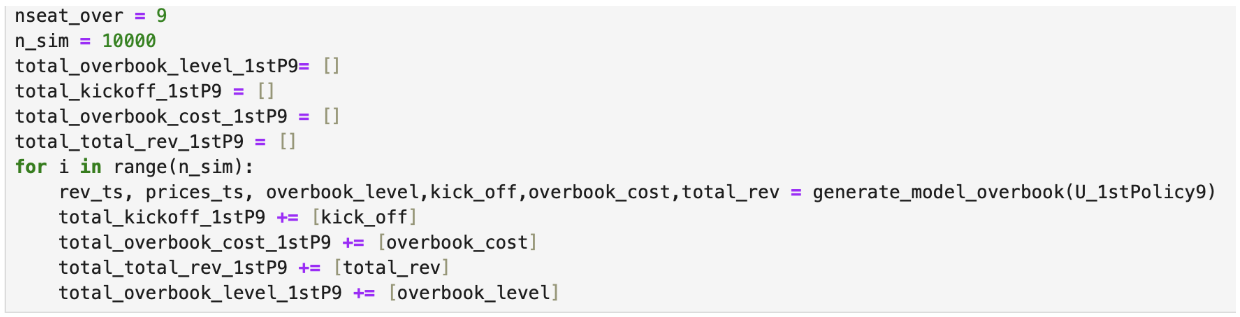
Run 10k simulations for each policy:

The last step is to run this 10,000 times for each policy. For each exercise, we calculate the percentage of times when we overbook (avg_times_kickoff), the percentage of times when we bump out someone (avg_times_overbook), the average number of people being overbook (avg_overbook_level_1stPx), the average number of people being bump out (avg_kickoff_1stPx), average overbooking cost (avg_overbook_cost_1stPx), and average revenue (avg_total_rev_1stPx). We also calculate the same variables but for standard deviation to understand the volatility of each variable.

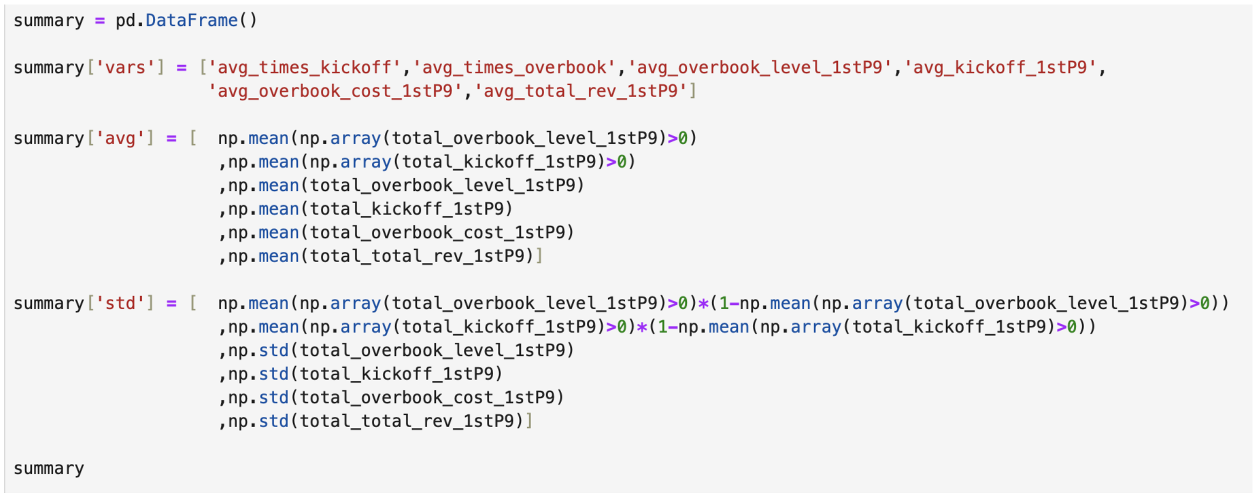
- First Policy with 20 overbooking

```
nseat_over = 20

total_overbook_level_1stP20= []
total_kickoff_1stP20 = []
total_overbook_cost_1stP20 = []
total_total_rev_1stP20 = []
for i in range(n_sim):
    rev_ts, prices_ts, overbook_level,kick_off,overbook_cost,total_rev = generate_model_overbook(U_1stPolicy)
    total_kickoff_1stP20 += [kick_off]
    total_overbook_cost_1stP20 += [overbook_cost]
    total_total_rev_1stP20 += [total_rev]
    total_overbook_level_1stP20 += [overbook_level]
```

- First Policy with 9 overbooking

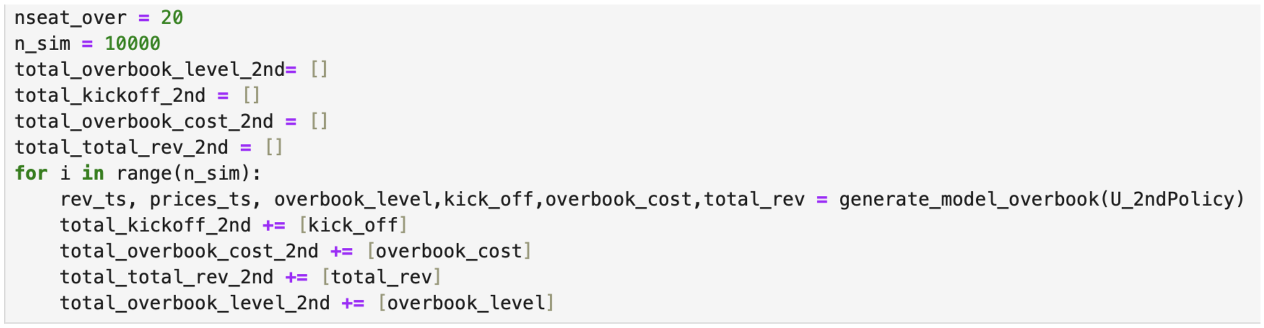


```
nseat_over = 9
n_sim = 10000
total_overbook_level_1stP9= []
total_kickoff_1stP9 = []
total_overbook_cost_1stP9 = []
total_total_rev_1stP9 = []
for i in range(n_sim):
    rev_ts, prices_ts, overbook_level,kick_off,overbook_cost,total_rev = generate_model_overbook(U_1stPolicy9)
    total_kickoff_1stP9 += [kick_off]
    total_overbook_cost_1stP9 += [overbook_cost]
    total_total_rev_1stP9 += [total_rev]
    total_overbook_level_1stP9 += [overbook_level]
```



```
summary = pd.DataFrame()

summary['vars'] = ['avg_times_kickoff','avg_times_overbook','avg_overbook_level_1stP9','avg_kickoff_1stP9',
                   'avg_overbook_cost_1stP9','avg_total_rev_1stP9']

summary['avg'] = [  np.mean(np.array(total_overbook_level_1stP9)>0)
                    ,np.mean(np.array(total_kickoff_1stP9)>0)
                    ,np.mean(total_overbook_level_1stP9)
                    ,np.mean(total_kickoff_1stP9)
                    ,np.mean(total_overbook_cost_1stP9)
                    ,np.mean(total_total_rev_1stP9)]

summary['std'] = [  np.mean(np.array(total_overbook_level_1stP9)>0)*(1-np.mean(np.array(total_overbook_level_1stP9)>0))
                    ,np.mean(np.array(total_kickoff_1stP9)>0)*(1-np.mean(np.array(total_kickoff_1stP9)>0))
                    ,np.std(total_overbook_level_1stP9)
                    ,np.std(total_kickoff_1stP9)
                    ,np.std(total_overbook_cost_1stP9)
                    ,np.std(total_total_rev_1stP9)]

summary
```

- Second Policy with 20 overbooking



```
nseat_over = 20
n_sim = 10000
total_overbook_level_2nd= []
total_kickoff_2nd = []
total_overbook_cost_2nd = []
total_total_rev_2nd = []
for i in range(n_sim):
    rev_ts, prices_ts, overbook_level,kick_off,overbook_cost,total_rev = generate_model_overbook(U_2ndPolicy)
    total_kickoff_2nd += [kick_off]
    total_overbook_cost_2nd += [overbook_cost]
    total_total_rev_2nd += [total_rev]
    total_overbook_level_2nd += [overbook_level]
```

```
summary = pd.DataFrame()

summary['vars'] = ['avg_times_kickoff','avg_times_overbook','avg_overbook_level_2nd','avg_kickoff_2nd',
                   'avg_overbook_cost_2nd','avg_total_rev_2nd']

summary['avg'] = [  np.mean(np.array(total_overbook_level_2nd)>0)
                    ,np.mean(np.array(total_kickoff_2nd)>0)
                    ,np.mean(total_overbook_level_2nd)
                    ,np.mean(total_kickoff_2nd)
                    ,np.mean(total_overbook_cost_2nd)
                    ,np.mean(total_total_rev_2nd)]

summary['std'] = [  np.mean(np.array(total_overbook_level_2nd)>0)*(1-np.mean(np.array(total_overbook_level_2nd)>0))
                    ,np.mean(np.array(total_kickoff_2nd)>0)*(1-np.mean(np.array(total_kickoff_2nd)>0))
                    ,np.std(total_overbook_level_2nd)
                    ,np.std(total_kickoff_2nd)
                    ,np.std(total_overbook_cost_2nd)
                    ,np.std(total_total_rev_2nd)]

summary
```

3.3.3. Results

We have computed the averages of the key metrics outlined in the overview, along with their standard deviations to capture the volatility of the resultant figures. Below are the detailed results for each scenario.

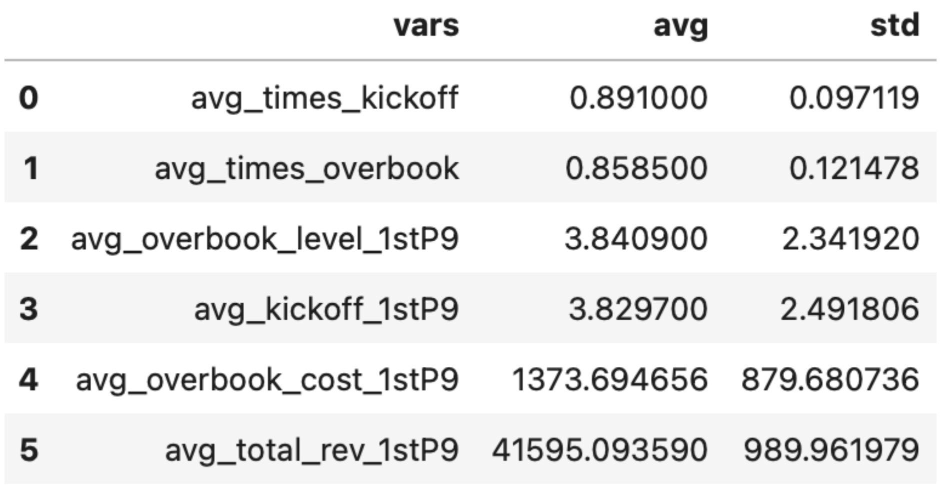
Metrics Results by each scenario:

First Policy with an overbooking of 20 seats

	vars	avg	std
0	avg_times_kickoff	0.911000	0.081079
1	avg_times_overbook	0.890900	0.097197
2	avg_overbook_level_1stP20	7.148900	5.123254
3	avg_kickoff_1stP20	6.732400	4.880552
4	avg_overbook_cost_1stP20	2431.618383	1752.829531
5	avg_total_rev_1stP20	41471.998036	999.822578

For the first policy without a no-sale option for coach tickets and a maximum overbooking of 20 seats, the simulation results indicate that the average level of overbooking in coach was 7.14. Instances of passenger removal were quite frequent, with an average of 6.73 removals per simulation. The financial toll of overbooking under this policy led to an average cost of $2,431.62. Despite this, the total revenue averaged a substantial $41,472.00. When considering volatility, the standard deviations of overbooking costs and total revenue were 1752.83 and 999.82, respectively, reflecting moderate fluctuations in financial outcomes.

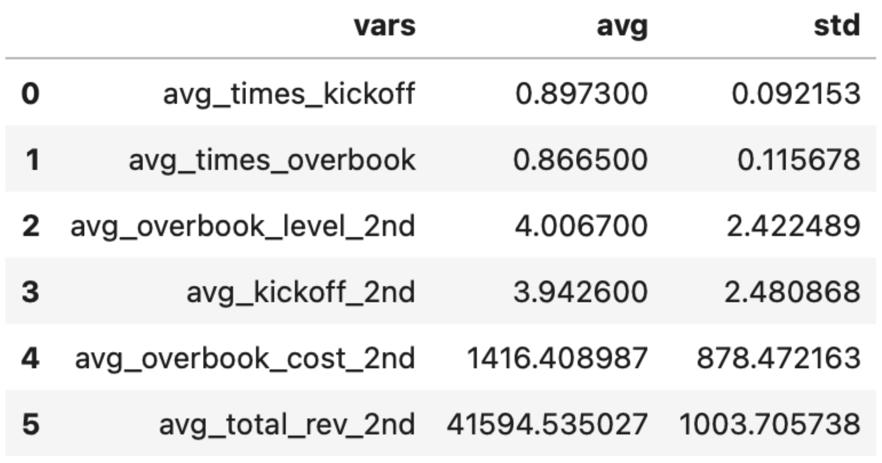
First Policy with an overbooking of 9 seats



	vars	avg	std
0	avg_times_kickoff	0.891000	0.097119
1	avg_times_overbook	0.858500	0.121478
2	avg_overbook_level_1stP9	3.840900	2.341920
3	avg_kickoff_1stP9	3.829700	2.491806
4	avg_overbook_cost_1stP9	1373.694656	879.680736
5	avg_total_rev_1stP9	41595.093590	989.961979

In the scenario employing the first policy with an optimal overbooking of 9 seats, the simulations reveal a reduced average overbooking level in coach of 3.84, with fewer occurrences of passenger removal, averaging at 3.83 instances. The average overbooking cost decreased to $1,373.69, contributing to an improved total revenue average of $41,595.09. Volatility was observed to be less pronounced in this scenario, with standard deviations of overbooking cost and total revenue at 879.68 and 989.96, respectively, suggesting a more stable financial performance.

Second Policy with an overbooking of 20 seats

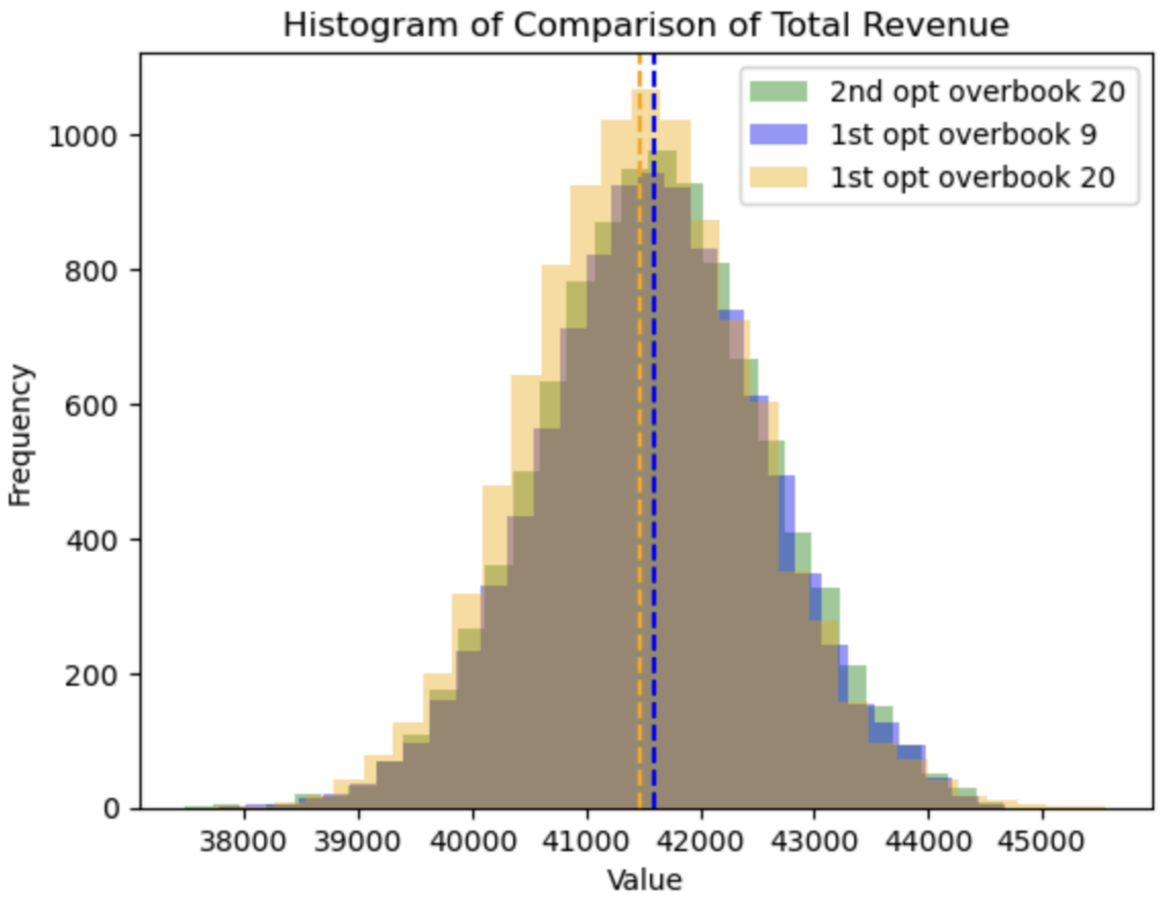


	vars	avg	std
0	avg_times_kickoff	0.897300	0.092153
1	avg_times_overbook	0.866500	0.115678
2	avg_overbook_level_2nd	4.006700	2.422489
3	avg_kickoff_2nd	3.942600	2.480868
4	avg_overbook_cost_2nd	1416.408987	878.472163
5	avg_total_rev_2nd	41594.535027	1003.705738

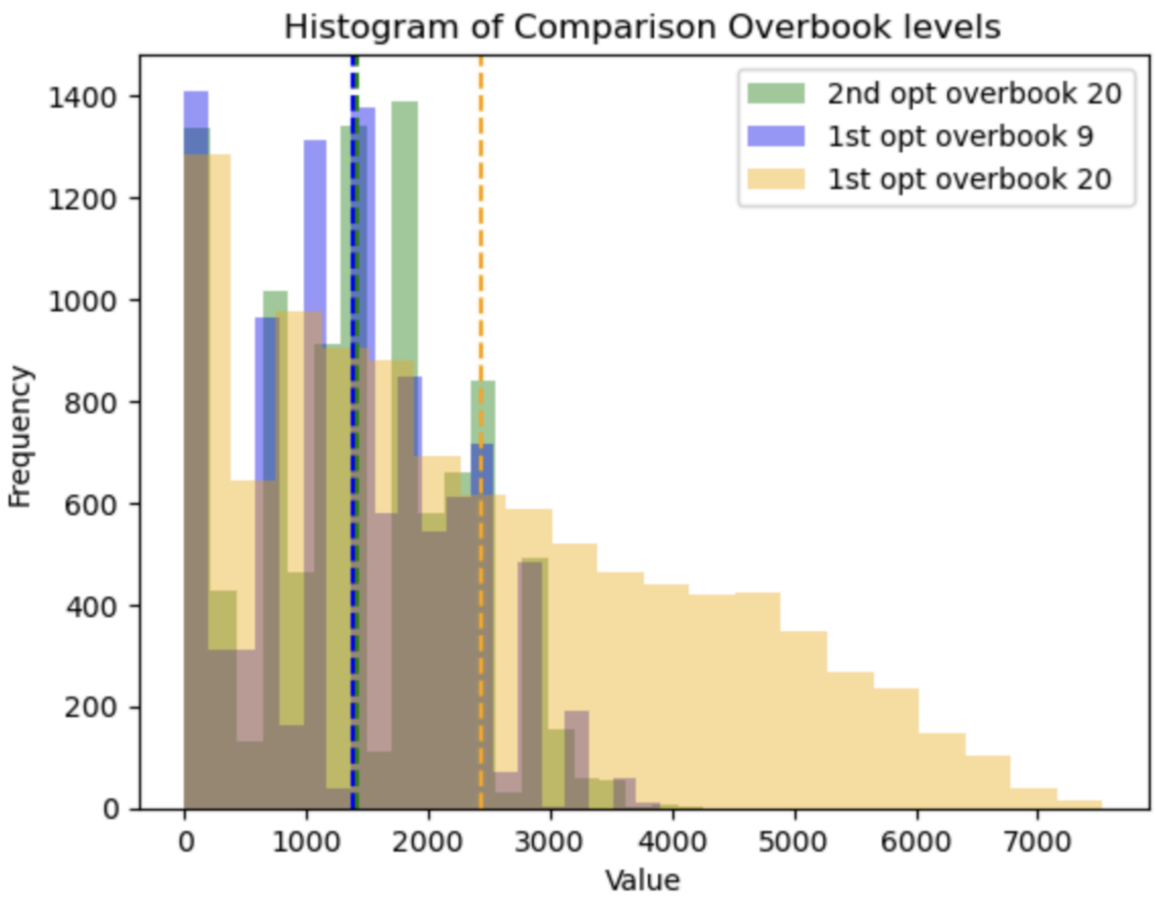
Lastly, applying the second policy with a no-sale option for coach tickets and a maximum overbooking of 20 seats, the simulation indicated an average coach overbooking level of 4.007, with an average of 3.943 passengers being removed from flights. This policy resulted in an average overbooking cost of $1,416.41, which, while slightly higher, still allowed for a robust total revenue of $41,594.54. The standard deviations for overbooking cost and total revenue were 878.47 and 1003.71, respectively, suggesting a volatility level comparable to the optimized first policy scenario. This indicates that both policies maintain relatively stable financial performance with minimal passenger disruption.

Comparative Analysis:

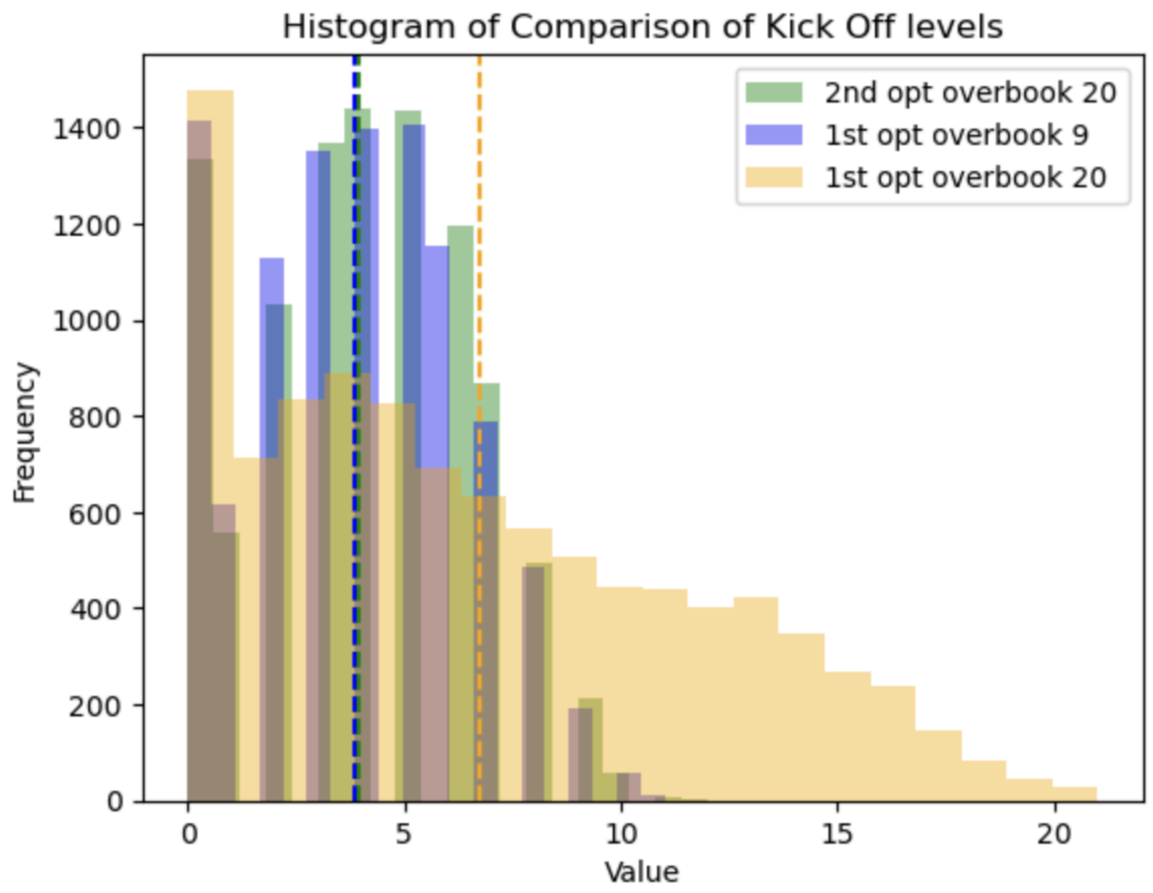
In the pursuit of crafting the best overbooking strategy, our comparative analysis hones in on the three key scenarios outlined. The histograms vividly illustrate the distribution of outcomes, guiding us toward the most prudent policy.



For the revenue comparison, the second policy with a no-sale option and maximum overbooking of 20 seats appears to show a distribution that is wide-ranging, with the mean revenue being nearly equivalent to the first policy with an optimal overbooking of 9 seats. However, the wider spread suggests that while the second policy has the potential to achieve high revenue peaks, it also has instances of significantly lower revenue, contributing to a larger standard deviation and thus greater volatility. This wider distribution indicates a more pronounced financial risk compared to the more consistent performance of the first policy with 9 overbooked seats.



Shifting our focus to overbooking levels, we notice a notable disparity in the spread of frequencies. The first policy with an optimal overbooking of 9 seats displays a tighter distribution, indicating fewer instances of overbooking and subsequently a lower probability of having to deny boarding to passengers due to overfilled flights. In contrast, the second policy with 20 overbooked seats, despite showing similar mean values, presents a wider range of overbooking instances, suggesting a more unpredictable nature of this policy.



When we turn to analyze passenger removals, the second policy with 20 overbooked seats reveals a larger spread in the frequency of removals, indicating potentially higher numbers of removals. This variance points to an increased risk of customer dissatisfaction when compared to the first policy with 9 overbooked seats, which maintains a narrower and more focused distribution of removal events. The first policy with 20 overbooked seats is especially noteworthy for its significantly high standard deviation, signaling a greater chance of encountering substantial customer service issues. From the perspective of maintaining customer relations, the first policy with 20 overbooked seats emerges as a considerably riskier strategy.

4. Recommendation

Upon meticulous examination of the overbooking strategies through forward simulation, we have decided to recommend using policy one with 9 seats of overbooking because even when it has lower returns the difference is just 5 dollars in expectation. Furthermore, after running the simulations this 1st policy with 9 seats overbooked has lower volatility on the profits, a lower number of seats overbooked and it results in less percentage of times with overbooked and bump out passengers. In that sense, it is a better policy for customer satisfaction and, as stated before, it has just five dollars of difference in expectation compared with the second policy overbooking 20 seats.

Moreover, the first policy with 20 overbooked seats is not a suitable option because it has lower expected profit and it also carries a substantial risk of financial volatility and customer dissatisfaction, as indicated by the high standard deviation in passenger removals. Conversely, the second policy, even with the no-sale option at the same level of overbooking, offered slightly improved stability but similarly exhibited broader fluctuations in both revenue and overbooking levels.

The optimal overbooking strategy appears to lie within a moderated approach, specifically the first policy with an optimal overbooking of 9 seats. This policy not only presented the lowest average overbooking cost but also maintained a near-equivalent average total revenue compared to the other more aggressive overbooking strategies. Furthermore, its standard deviation figures for both overbooking cost and total revenue were lower, signaling a reduced risk profile. This balanced strategy yielded a more consistent revenue stream while mitigating the extent and frequency of operational disruptions, such as passenger removals, which are likely to influence customer satisfaction negatively.

In alignment with the overarching goal of this project—to discover a policy that maximizes profits not solely from explicit financial metrics but also by considering implicit profit/loss associated with customer satisfaction—we advocate for the adoption of the first policy with an optimal overbooking of 9 seats. This recommendation is anchored in its demonstrated ability to uphold robust revenue generation while curtailing overbooking costs and enhancing customer experience. By selecting this policy, we affirm our commitment to a profitable yet customer-centric operational ethos, ensuring sustained fiscal health and customer goodwill, which are vital to our airline's reputation and long-term success.