Optimization II Project II: Dynamic Programming

Sp23 OPTIMIZATION II

Table of Contents

I. Introduction

In this project, our team of analysts was tasked with developing an optimal pricing policy for an airline by considering two overbooking strategies: a static overbooking policy and a dynamic overbooking solution. The target of our research was to assess the effectiveness of overbooking. With the goal of maximizing the airline's expected discounted profit today, we determined the optimal policy by considering the daily revenue collected overtime from ticket sales less the costs incurred for overbooking coach seats at the time of the plane's departure. These overbooking costs came in the form of bumping passengers up to first-class, when available, and giving passengers stipends who would not fit on the plane. Utilizing two ticket classes, first-class and coach, and their respective pricing and demand structures, we solved both the static and dynamic overbooking policy solutions backwards in time and verified our results by simulating each solution forwards to obtain a measure of its volatility.

Ultimately, the team discovered that the dynamic overbooking solution was optimal and maximized the airline's expected discounted profit. This solution was confirmed by our simulated results, which suggested that the firm can be confident that 95% of the time, expected discounted profit will fall in the range of \$42,123.49 to \$42,160.49 when using our dynamic overbooking engine. When compared to the suboptimal case where the firm cannot oversell coach tickets, the dynamic policy resulted in an increase in expected discounted profit by nearly \$1,485.60, confirming the efficacy of this policy.

II. Methodology

To begin this project, we first solved the static overbooking problem before transitioning to the dynamic overbooking policy. Although different from the dynamic solution with respect to the flexibility it offered the airline, both the static and dynamic policies were similar in the sense that they utilized the same six foundational components of a dynamic programming problem. The details are shown below.

State Variables: (*t, c, f*)

- *t*: time from day 0 to day 365 (plane departs)
- *c*: number of coach seats sold
- *f*: number of first-class seats sold

Choice Variables:

- *HH*: price coach high, price first-class high
- *HL*: price coach high, price first-class low
- *LH*: price coach low, price first-class high
- *LL*: price coach low, price first-class low
- *NH*: force coach demand to zero, price first-class high **(Dynamic policy only)**
- *NL*: force coach demand to zero, price first-class low **(Dynamic policy only)**
- *HF only:* coach sold out, price first-class high
- *LF only:* coach sold out, price first-class low
- *HC only:* price coach high, first-class sold out
- *LC only:* price coach low, first-class sold out
- *NC only:* force coach demand to zero, first-class sold out **(Dynamic policy only)**

Dynamics:

These dynamics were used to determine where to evaluate the Bellman equation tomorrow.

1. ***HH* Dynamics:**
   - s/s: (t, c, f) → (t+1, c+1, f+1) w.p. (0.3)*(0.04) = 0.012
   - s/ns: (t, c, f) → (t+1, c+1, f) w.p. (0.3)*(0.96) = 0.288
   - ns/s (t, c, f) → (t+1, c, f+1) w.p. (0.7)*(0.04) = 0.028
   - ns/ns: (t, c, f) → (t+1, c, f) w.p. (0.7)*(0.96) = 0.672

2. ***HL* Dynamics:**
   - s/s: (t, c, f) → (t+1, c+1, f+1) w.p. (0.3)*(0.08) = 0.024
   - s/ns: (t, c, f) → (t+1, c+1, f) w.p. (0.3)*(0.92) = 0.276
   - ns/s: (t, c, f) → (t+1, c, f+1) w.p. (0.7)*(0.08) = 0.056
   - ns/ns: (t, c, f) → (t+1, c, f) w.p. (0.7)*(0.92) = 0.644

3. ***LH* Dynamics:**
   - s/s: (t, c, f) → (t+1, c+1, f+1) w.p. (0.65)*(0.04) = 0.026
   - s/ns: (t, c, f) → (t+1, c+1, f) w.p. (0.65)*(0.96) = 0.624
   - ns/s: (t, c, f) → (t+1, c, f+1) w.p. (0.35)*(0.04) = 0.014
   - ns/ns: (t, c, f) → (t+1, c, f) w.p. (0.35)*(0.96) = 0.336

4. ***LL* Dynamics:**
   - s/s: (t, c, f) → (t+1, c+1, f+1) w.p. (0.65)*(0.08) = 0.052
   - s/ns: (t, c, f) → (t+1, c+1, f) w.p. (0.65)*(0.92) = 0.598
   - ns/s: (t, c, f) → (t+1, c, f+1) w.p. (0.35)*(0.08) = 0.028
   - ns/ns: (t, c, f) → (t+1, c, f) w.p. (0.35)*(0.92) = 0.322

5. ***NH* Dynamics: (Dynamic policy only)**
   - s/s: (t, c, f) → (t+1, c+1, f+1) w.p. (0)*(0.04) = 0
   - s/ns: (t, c, f) → (t+1, c+1, f) w.p. (0)*(0.96) = 0
   - ns/s: (t, c, f) → (t+1, c, f+1) w.p. (1)*(0.04) = 0.04
   - ns/ns: (t, c, f) → (t+1, c, f) w.p. (1)*(0.96) = 0.96

6. ***NL* Dynamics: (Dynamic policy only)**
   - s/s: (t, c, f) → (t+1, c+1, f+1) w.p. (0)*(0.08) = 0
   - s/ns: (t, c, f) → (t+1, c+1, f) w.p. (0)*(0.92) = 0
   - ns/s: (t, c, f) → (t+1, c, f+1) w.p. (1)*(0.08) = 0.08
   - ns/ns: (t, c, f) → (t+1, c, f) w.p. (1)*(0.92) = 0.92

7. ***HF only* Dynamics:**
   - s/s: (t, c, f) → (t+1, c+1, f+1) w.p. (0)*(0.04) = 0
   - s/ns: (t, c, f) → (t+1, c+1, f) w.p. (0)*(0.96) = 0
   - ns/s: (t, c, f) → (t+1, c, f+1) w.p. (1)*(0.04) = 0.04
   - ns/ns: (t, c, f) → (t+1, c, f) w.p. (1)*(0.96) = 0.96

8. ***LF only* Dynamics:**
   - s/s: (t, c, f) → (t+1, c+1, f+1) w.p. (0)*(0.08) = 0
   - s/ns: (t, c, f) → (t+1, c+1, f) w.p. (0)*(0.92) = 0
   - ns/s: (t, c, f) → (t+1, c, f+1) w.p. (1)*(0.08) = 0.08
   - ns/ns: (t, c, f) → (t+1, c, f) w.p. (1)*(0.92) = 0.92

9. ***HC only* Dynamics:**
   - s/s: (t, c, f) → (t+1, c+1, f+1) w.p. (0.3)*(0) = 0
   - s/ns: (t, c, f) → (t+1, c+1, f) w.p. (0.3)*(1) = 0.3
   - ns/s: (t, c, f) → (t+1, c, f+1) w.p. (0.7)*(0) = 0
   - ns/ns: (t, c, f) → (t+1, c, f) w.p. (0.7)*(1) = 0.7

10. ***LC only* Dynamics:**
    - s/s: (t, c, f) → (t+1, c+1, f+1) w.p. (0.65)*(0) = 0
    - s/ns: (t, c, f) → (t+1, c+1, f) w.p. (0.65)*(1) = 0.65
    - ns/s: (t, c, f) → (t+1, c, f+1) w.p. (0.35)*(0) = 0
    - ns/ns: (t, c, f) → (t+1, c, f) w.p. (0.35)*(1) = 0.35

11. ***NC only* Dynamics: (Dynamic policy only)**
    - s/s: (t, c, f) → (t+1, c+1, f+1) w.p. (0)*(0) = 0
    - s/ns: (t, c, f) → (t+1, c+1, f) w.p. (0)*(1) = 0
    - ns/s: (t, c, f) → (t+1, c, f+1) w.p. (1)*(0) = 0
    - ns/ns: (t, c, f) → (t+1, c, f) w.p. (1)*(1) = 1

s/s = *sell c ticket and sell f ticket*
s/ns = *sell c ticket and do not sell f ticket*
ns/s = *do not sell c ticket and sell f ticket*
ns/ns = *do not sell c ticket and do not sell f ticket*

Value Function:

$$V(t, c, f) = \max_{price} E\left[\sum_{i=0}^{T-t} (profit\,@\,t + i) * \delta^{i}\right]$$

$$daily\ \delta = 1/(1 + 0.17/365)^{1}$$

[1] The annual discount rate is 17%

Bellman Equation (General):

$$\begin{aligned} V(t,c,f) = \max_{price} [&E\,[revenue\ today] + \delta\ E[V(tomorrow)]\ if\ HH,\\ &E\,[revenue\ today] + \delta\ E[V(tomorrow)]\ if\ HL,\\ &E\,[revenue\ today] + \delta\ E[V(tomorrow)]\ if\ LH,\\ &E\,[revenue\ today] + \delta\ E[V(tomorrow)]\ if\ LL,\\ &E\,[revenue\ today] + \delta\ E[V(tomorrow)]\ if\ NH,\text{[2]}\\ &E\,[revenue\ today] + \delta\ E[V(tomorrow)]\ if\ NL,\text{[3]}\\ &E\,[revenue\ today] + \delta\ E[V(tomorrow)]\ if\ HF\ only,\\ &E\,[revenue\ today] + \delta\ E[V(tomorrow)]\ if\ LF\ only,\\ &E\,[revenue\ today] + \delta\ E[V(tomorrow)]\ if\ HC\ only,\\ &E\,[revenue\ today] + \delta\ E[V(tomorrow)]\ if\ LC\ only,\\ &E\,[revenue\ today] + \delta\ E[V(tomorrow)]\ if\ NC\ only\text{[4]}] \end{aligned}$$

$$\delta = 1/(1 + 0.17 / 365)$$

Bellman Equation (Expanded):
Expanded for just the *HH* term as an example:

$$V(t,c,f) = \max_{price} [E\,[revenue\ today] + \delta\ E[V(tomorrow)]\ if\ HH, \ldots$$

$$\begin{aligned} = \max_{price} [&(revenue\ if\ s/s) * (p(s/s)) + (revenue\ if\ s/ns) * (p(s/ns)) +\\ &(revenue\ if\ ns/s) * (p(ns/s)) + (revenue\ if\ ns/ns) * (p(ns/ns)) +\\ &V(t+1,\ c+1,\ f+1) * (p(s/s)) + V(t+1,\ c+1,\ f) * (p(s/ns)) +\\ &V(t+1,\ c,\ f+1) * (p(ns/s)) + V(t+1,\ c,\ f) * (p(ns/ns)) + \ldots] \end{aligned}$$

$$\begin{aligned} = \max_{price} [&(high\ c\ price + high\ f\ price) * ((0.3) * (0.04)) + (high\ c\ price) * ((0.3) * (0.96)) +\\ &(high\ f\ price) * ((0.7) * (0.04)) + (0) * ((0.7) * (0.96)) +\\ &V(t+1,\ c+1,\ f+1) * ((0.3) * (0.04)) + V(t+1,\ c+1,\ f) * ((0.3) * (0.96)) +\\ &V(t+1,\ c,\ f+1) * ((0.7) * (0.04)) + V(t+1,\ c,\ f) * ((0.7) * (0.96)) + \ldots] \end{aligned}$$

s/s = *sell c ticket and sell f ticket*
s/ns = *sell c ticket and do not sell f ticket*
ns/s = *do not sell c ticket and sell f ticket*
ns/ns = *do not sell c ticket and do not sell f ticket*

This procedure was repeated for all terms in the general Bellman equation defined on the previous page.

[2] "*E [revenue today] + δ E[V(tomorrow)] if NH*" term only applies to the Dynamic solution
[3] "*E [revenue today] + δ E[V(tomorrow)] if NL*" term only applies to the Dynamic solution
[4] "*E [revenue today] + δ E[V(tomorrow)] if NC only*" term only applies to the Dynamic solution

Terminal Condition:

$$V(T, c, f) = -\, E\, [cost\ given\ c\ seats\ sold\ and\ f\ seats\ sold]$$

$$V(T, c, f) = -\sum_{i=0}^{c}\sum_{j=0}^{f} (cost\ of\ i, j\ showing\ up) * (probability\ of\ i, j\ showing\ up)$$

$$= -\sum_{i=0}^{c}\sum_{j=0}^{f} (cost\ of\ i, j\ showing\ up) * (probability\ of\ i\ showing\ up\ if\ c\ sold) * (probability\ of\ j\ showing\ up\ if\ f\ seats\ sold)$$

$$for\ i = 0, 1, 2, \dots, c$$
$$for\ j = 0, 1, 2, \dots, j$$
$$T = 365$$

A. Static Overbooking Policy

Utilizing the dynamic programming elements defined above, we developed a static overbooking policy that allowed the airline to overbook up to x number of coach seats on the flight. The airline did not allow first-class seats to be oversold in order to preserve the goodwill of high-paying customers. A flight was considered to be overbooked when more passengers purchased coach tickets than there were actual coach seats available on the plane. In our static policy solution, we explored the airline's expected discounted profit for $x = 5, 6, 7, 8, \ldots, 15$ coach seats oversold. This solution relied on the demand and pricing information for the airline's customers displayed in **Figure 1** below.

	Coach Tickets	**First-Class Tickets**
Seats Available	100	20
High Price	$ 350.00	$ 500.00
Low Price	$ 300.00	$ 425.00
High Price Demand	0.30	0.04
Low Price Demand	0.65	0.08
Percent Show-up	0.95	0.97
Ability to Overbook	Yes	No

Figure 1: Demand and pricing information for coach and first-class tickets

If a flight was overbooked, the airline would incur overbooking costs in accordance with the pricing scheme below if more coach passengers showed up than there were seats available.

- Cost to bump a coach passenger to first-class, if available: **$50**
- Cost to bump a coach passenger off the plane: **$425**

As can be seen, the airline incurs a much steeper penalty for kicking a passenger off the plane than simply bumping him or her up to first-class. Therefore, the airline will utilize any extra room in first-class before bumping a passenger off the plane, which is much more costly. While we recognize that the airline faces many other operational costs, we consider them fixed and only optimize over its overbooking costs.

After defining the necessary variables, we set out to solve the static policy program to optimality by determining what pricing structure the airline should set each day leading up to the flight's departure. On any given day, the airline can either sell 0 or 1 ticket in coach and 0 or 1 ticket in first-class. It is important to note that the demand distributions for coach and first-class seats are independent of each other, indicating that their joint probabilities are equal to the product of their marginal probabilities. This relationship is detailed in the example shown below:

*p(sell c high price, sell f high price) = p(sell c high price) * p(sell f high price)*

In this environment, if the demand for first-class seats exceeded the first-class seats available, customers would not buy coach seats. Likewise, if the demand for coach seats were to exceed the coach seats available, customers would not buy a first-class ticket. However, to account for those who would typically buy a first-class seat if they were sold out, the chance of sale of coach tickets increased by a flat rate of 3% in both the high and low pricing schemes in both the static and dynamic policy solutions.

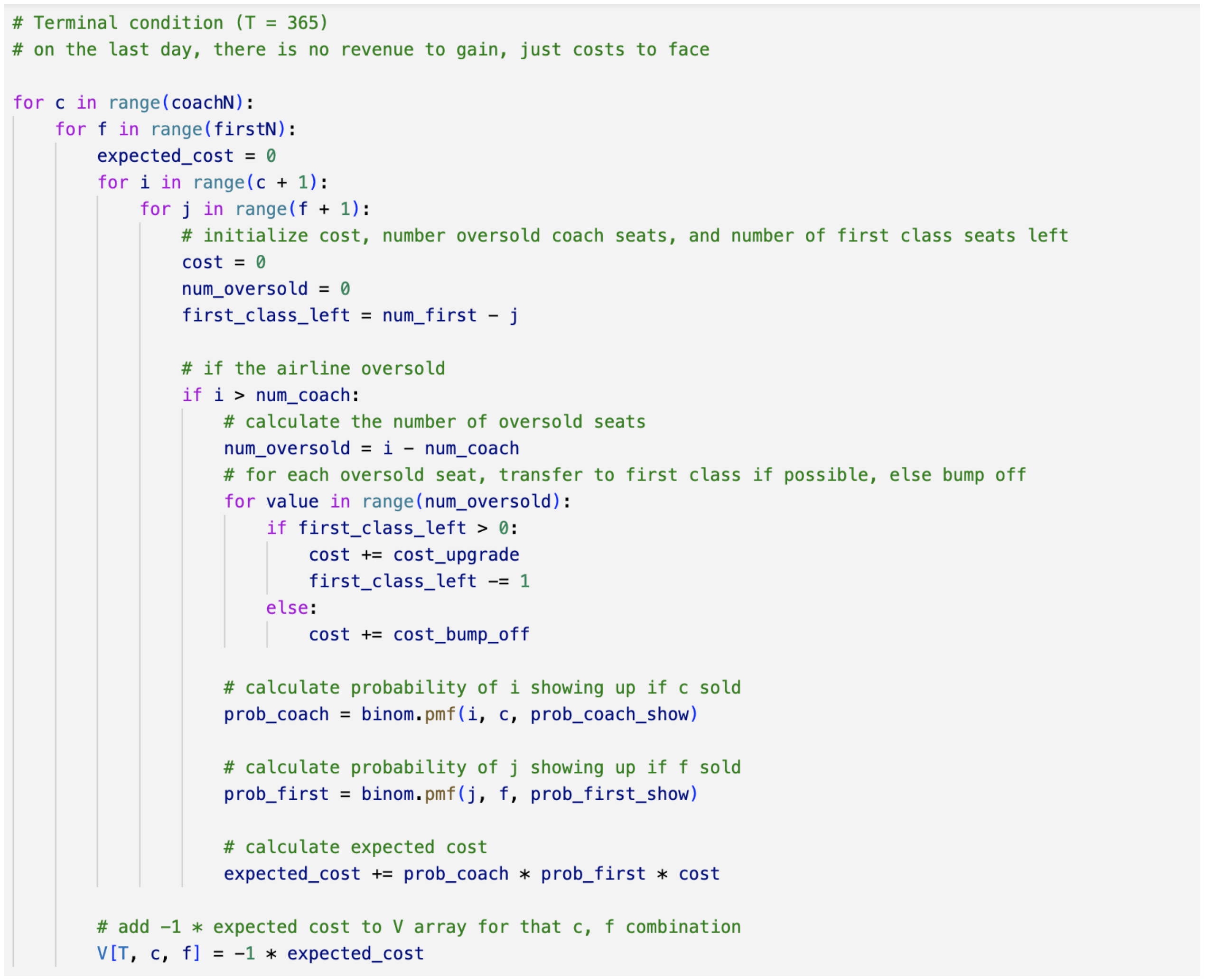
Because there were 365 days until the plane departed, the airline had 365 opportunities to sell both classes of tickets. Therefore, on the days leading up to the plane's departure, there were opportunities to collect revenue, but when the plane took off at time T = 365, the airline faced only overbooking costs depending upon the number of customers in each ticket class that actually showed up. As shown in **Figure 1** earlier, coach passengers showed up to the flight with probability 95% and first-class passengers showed up 97% of the time. In order to determine the expected discounted profit of selling tickets and paying out overbooking costs to its customers over the course of the year, we began by solving the problem backwards in time. To do so, we defined the total number of coach tickets, first-class tickets, and time periods possible. Only the coach ticket values depended on the number of oversold tickets allowed in each iteration of the solution. We then utilized these variables to create the three-dimensional arrays, *V* and *U*, to store the value function and optimal choice decisions for each value of (*t, c, f*) in our solution. The code used is shown here.

```python
def solve_expected_profit(coach_oversell) :

    # Time, coach, and first class values
    coachValues = np.arange(num_coach + coach_oversell + 1) # possible # coach seats sold (0, 1, 2,...,100 + oversell)
    firstValues = np.arange(num_first + 1)                  # possible # first-class seats sold (0, 1, 2,...20)
    tValues = np.arange(T + 1)                              # possible days until takeoff (0, 1, 2...365)

    coachN = len(coachValues) # count possible state values
    firstN = len(firstValues)
    tN = len(tValues)

    # Establish value function array (V) and optimal choice array (U)
    V = np.zeros((tN, coachN, firstN)) # initialize value function
    U = np.zeros((tN, coachN, firstN)) # initialize optimal choice variable; best price to choose
```

Following this step, we addressed the terminal condition as defined in the Methodology section. By looping through all possible values of *c* and *f* at the terminal time, T = 365, we added the expectation of cost incurred by the airline by looping once more through all possible values *i* and *j*, representing the number of passengers who could possibly show up to the flight with coach and first-class tickets, respectively. Using the scipy.stats binomial probability mass function in python, we were able to compute the probability of *i* passengers showing up if *c* tickets were sold, for example. From there, we were able to compute an expected cost by using the following formula. It leveraged the information that the sale of each coach ticket was independent of the sale of each first-class ticket.

*expected cost = – 1 * (prob i show up when c sold) * (prob j show up when f sold) * cost*

The cost was determined by first seeing if each value of *i* exceeded the total number of coach seats available (100 tickets). If more coach passengers showed up than there was room on the plane, they were bumped up to first-class where there was room ($50) and the remaining overbooked passengers were sent home with a stipend ($425). Once each expectation of overbooking cost was determined for each *c* and *f* combination, they were stored in the *V* array for that value of (*T, c, f*). The code to replicate this portion is shown below.

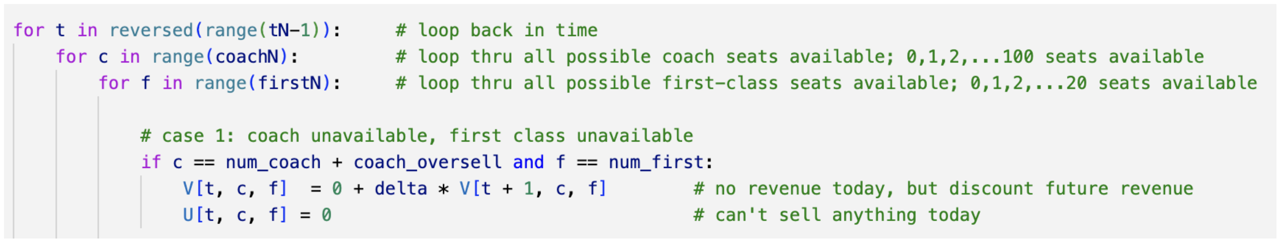


```
# Terminal condition (T = 365)
# on the last day, there is no revenue to gain, just costs to face

for c in range(coachN):
    for f in range(firstN):
        expected_cost = 0
        for i in range(c + 1):
            for j in range(f + 1):
                # initialize cost, number oversold coach seats, and number of first class seats left
                cost = 0
                num_oversold = 0
                first_class_left = num_first - j

                # if the airline oversold
                if i > num_coach:
                    # calculate the number of oversold seats
                    num_oversold = i - num_coach
                    # for each oversold seat, transfer to first class if possible, else bump off
                    for value in range(num_oversold):
                        if first_class_left > 0:
                            cost += cost_upgrade
                            first_class_left -= 1
                        else:
                            cost += cost_bump_off

                    # calculate probability of i showing up if c sold
                    prob_coach = binom.pmf(i, c, prob_coach_show)

                    # calculate probability of j showing up if f sold
                    prob_first = binom.pmf(j, f, prob_first_show)

                    # calculate expected cost
                    expected_cost += prob_coach * prob_first * cost

        # add -1 * expected cost to V array for that c, f combination
        V[T, c, f] = -1 * expected_cost
```

After establishing the terminal condition of our static overbooking policy, we solved this programming problem by moving backwards in time and using the Bellman equation. For each (*t, c, f*) combination, there were a total of four different conditions to address in relation to ticket sales. They are detailed below:

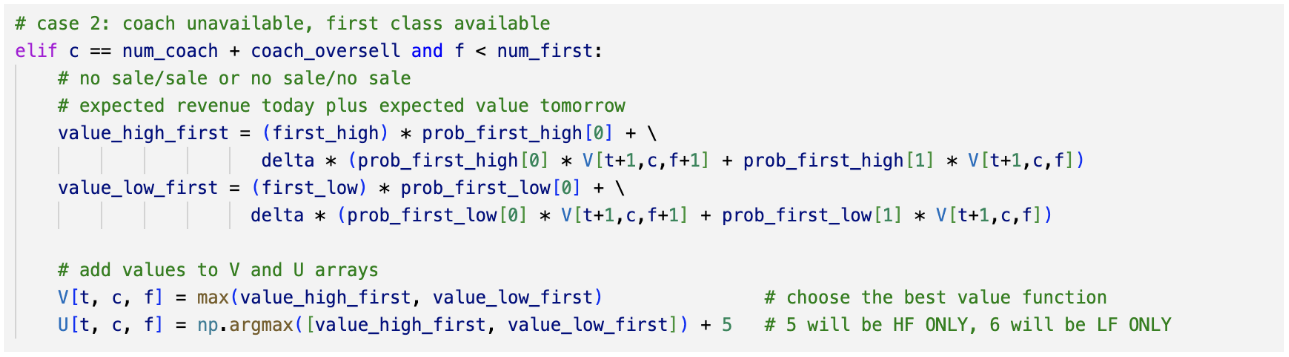
- **Case 1**: coach tickets sold out, first-class tickets sold out
- **Case 2**: coach tickets sold out, first-class tickets available
- **Case 3**: coach tickets available, first-class tickets sold out
- **Case 4:** coach tickets available, first-class tickets available

Within each of these four cases, the situations of *sale*/*sale*, *sale*/*no sale*, *no sale*/*sale*, and *no sale*/*no sale* had to be considered where the first value represented a coach ticket being sold and the second was in reference to a first-class ticket. We utilized the dynamics defined in the Methodology section to guide the construction of the Bellman equation in each specific case. In case 1, because both ticket classes were sold out, the airline had no opportunity to receive revenue from sales today, but still had a value for the expected discounted value of all future sales tomorrow. In the *U* array of optimal choices, this was encoded as a "0" to indicate no sales of either ticket class. This was implemented in the code as follows:



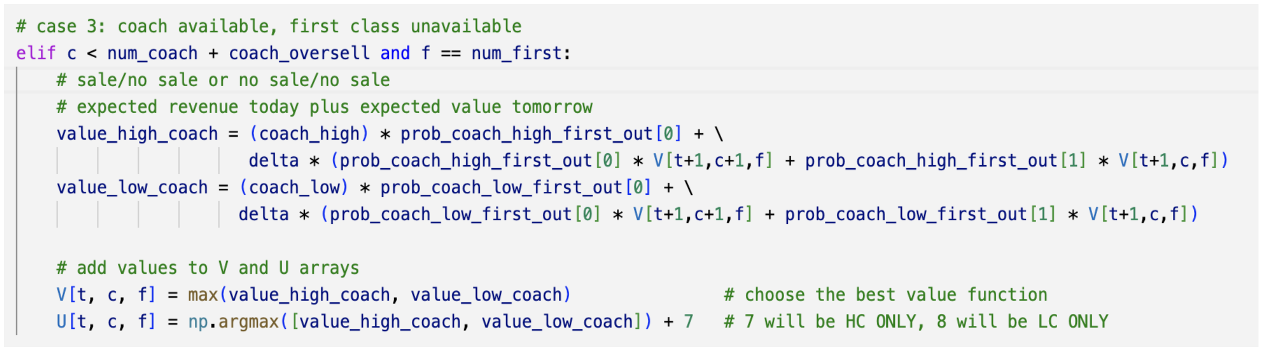
```
for t in reversed(range(tN-1)):     # loop back in time
    for c in range(coachN):         # loop thru all possible coach seats available; 0,1,2,...100 seats available
        for f in range(firstN):     # loop thru all possible first-class seats available; 0,1,2,...20 seats available

            # case 1: coach unavailable, first class unavailable
            if c == num_coach + coach_oversell and f == num_first:
                V[t, c, f]  = 0 + delta * V[t + 1, c, f]        # no revenue today, but discount future revenue
                U[t, c, f] = 0                                  # can't sell anything today
```

To address case 2, when coach tickets were sold out but first-class tickets were still available, the only two options the airline had were to set the first-class ticket price to either high (HF only) or low (LF only). This was encoded in our *U* array as cases "5" and "6", respectively. For example, in the high pricing scenario, the airline had the opportunity to receive the high first-class price as revenue today ($500) with a probability of 4% and had an expectation of future discounted value tomorrow for the cases in which they successfully made a high price first-class sale and the case in which they did not. A similar procedure was utilized to address the airline's other choice to set the low price for first-class on that day. We used this code for case 2:



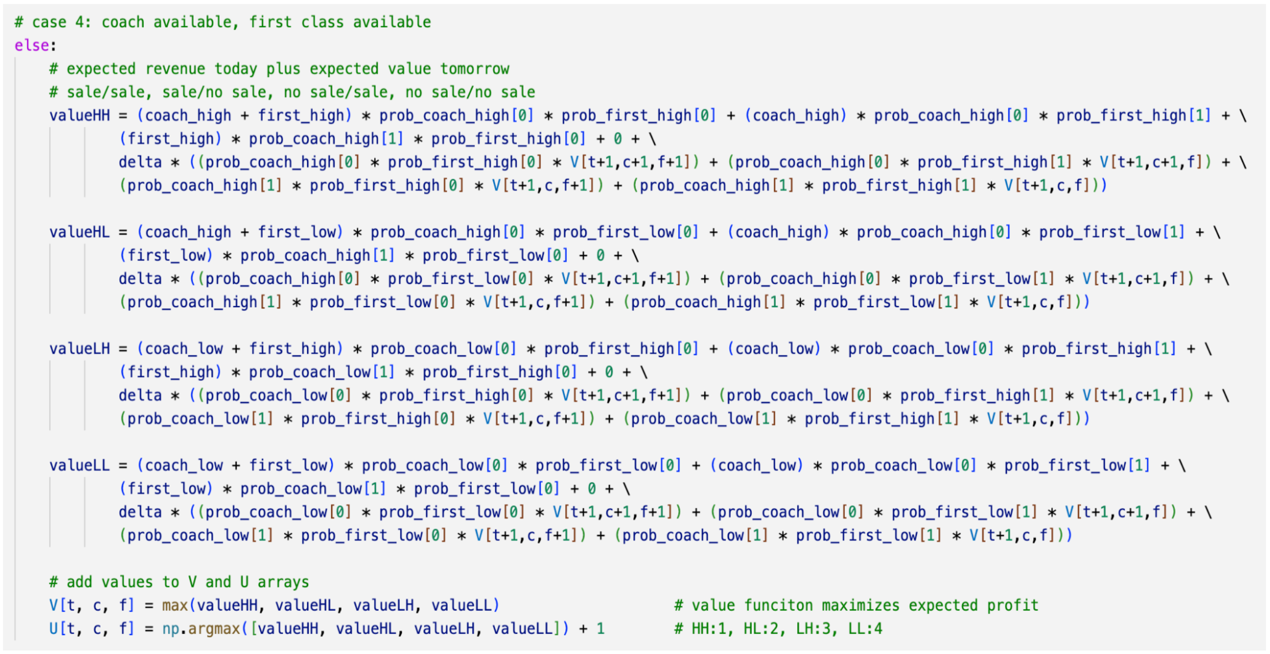
```
# case 2: coach unavailable, first class available
elif c == num_coach + coach_oversell and f < num_first:
    # no sale/sale or no sale/no sale
    # expected revenue today plus expected value tomorrow
    value_high_first = (first_high) * prob_first_high[0] + \
                        delta * (prob_first_high[0] * V[t+1,c,f+1] + prob_first_high[1] * V[t+1,c,f])
    value_low_first = (first_low) * prob_first_low[0] + \
                       delta * (prob_first_low[0] * V[t+1,c,f+1] + prob_first_low[1] * V[t+1,c,f])

    # add values to V and U arrays
    V[t, c, f] = max(value_high_first, value_low_first)                # choose the best value function
    U[t, c, f] = np.argmax([value_high_first, value_low_first]) + 5    # 5 will be HF ONLY, 6 will be LF ONLY
```

To address case 3, where coach tickets were available and first-class tickets were sold out, we recognized that in this case, the airline had only two choices to make, namely to charge the high coach ticket price (HC only) or the low price (LC only). Although similar to case 2, case 3 was slightly different because we had to utilize the updated probabilities of coach sales now that first-class seats were sold out. These probabilities were stored in the *prob_coach_high_first_out* and *prob_coach_low_first_out* variables and were indexed appropriately (0: sale, 1: no sale). Similarly to case 2, we encoded these two scenarios as "7" and "8" in our *U* matrix. The necessary code is shown in the image below.



```
# case 3: coach available, first class unavailable
elif c < num_coach + coach_oversell and f == num_first:
    # sale/no sale or no sale/no sale
    # expected revenue today plus expected value tomorrow
    value_high_coach = (coach_high) * prob_coach_high_first_out[0] + \
                        delta * (prob_coach_high_first_out[0] * V[t+1,c+1,f] + prob_coach_high_first_out[1] * V[t+1,c,f])
    value_low_coach = (coach_low) * prob_coach_low_first_out[0] + \
                       delta * (prob_coach_low_first_out[0] * V[t+1,c+1,f] + prob_coach_low_first_out[1] * V[t+1,c,f])

    # add values to V and U arrays
    V[t, c, f] = max(value_high_coach, value_low_coach)                # choose the best value function
    U[t, c, f] = np.argmax([value_high_coach, value_low_coach]) + 7    # 7 will be HC ONLY, 8 will be LC ONLY
```

Finally, to address case 4, in which both ticket classes were available, we determined that the airline would have all four pricing options available, HH, HL, LH, and LL, which were encoded in our *U* matrix as 1, 2, 3 and 4, respectively. In each pricing scheme, the opportunities to *sell*/*sell*, *sell*/*no sell*, *no sell*/*sell*, and *no sell*/*no sell* were all available, and factored into each value function value stored in the *V* array in accordance with their definition in the dynamics section above. This was implemented in code as follows:



```
# case 4: coach available, first class available
else:
    # expected revenue today plus expected value tomorrow
    # sale/sale, sale/no sale, no sale/sale, no sale/no sale
    valueHH = (coach_high + first_high) * prob_coach_high[0] * prob_first_high[0] + (coach_high) * prob_coach_high[0] * prob_first_high[1] + \
            (first_high) * prob_coach_high[1] * prob_first_high[0] + 0 + \
            delta * ((prob_coach_high[0] * prob_first_high[0] * V[t+1,c+1,f+1]) + (prob_coach_high[0] * prob_first_high[1] * V[t+1,c+1,f]) + \
            (prob_coach_high[1] * prob_first_high[0] * V[t+1,c,f+1]) + (prob_coach_high[1] * prob_first_high[1] * V[t+1,c,f]))

    valueHL = (coach_high + first_low) * prob_coach_high[0] * prob_first_low[0] + (coach_high) * prob_coach_high[0] * prob_first_low[1] + \
            (first_low) * prob_coach_high[1] * prob_first_low[0] + 0 + \
            delta * ((prob_coach_high[0] * prob_first_low[0] * V[t+1,c+1,f+1]) + (prob_coach_high[0] * prob_first_low[1] * V[t+1,c+1,f]) + \
            (prob_coach_high[1] * prob_first_low[0] * V[t+1,c,f+1]) + (prob_coach_high[1] * prob_first_low[1] * V[t+1,c,f]))

    valueLH = (coach_low + first_high) * prob_coach_low[0] * prob_first_high[0] + (coach_low) * prob_coach_low[0] * prob_first_high[1] + \
            (first_high) * prob_coach_low[1] * prob_first_high[0] + 0 + \
            delta * ((prob_coach_low[0] * prob_first_high[0] * V[t+1,c+1,f+1]) + (prob_coach_low[0] * prob_first_high[1] * V[t+1,c+1,f]) + \
            (prob_coach_low[1] * prob_first_high[0] * V[t+1,c,f+1]) + (prob_coach_low[1] * prob_first_high[1] * V[t+1,c,f]))

    valueLL = (coach_low + first_low) * prob_coach_low[0] * prob_first_low[0] + (coach_low) * prob_coach_low[0] * prob_first_low[1] + \
            (first_low) * prob_coach_low[1] * prob_first_low[0] + 0 + \
            delta * ((prob_coach_low[0] * prob_first_low[0] * V[t+1,c+1,f+1]) + (prob_coach_low[0] * prob_first_low[1] * V[t+1,c+1,f]) + \
            (prob_coach_low[1] * prob_first_low[0] * V[t+1,c,f+1]) + (prob_coach_low[1] * prob_first_low[1] * V[t+1,c,f]))

    # add values to V and U arrays
    V[t, c, f] = max(valueHH, valueHL, valueLH, valueLL)                # value funciton maximizes expected profit
    U[t, c, f] = np.argmax([valueHH, valueHL, valueLH, valueLL]) + 1    # HH:1, HL:2, LH:3, LL:4
```

Once this was fully coded out, the static policy was run with overbooking seat limits of 5 through 15 seats in order to determine the optimal pricing policy. The expected profit for each solution was determined by the value in the *V* array of *V[0, 0, 0]*, which corresponded to the value of all future sales and costs incurred by the airline discounted to time period 0. The results of this solution are discussed in the Results section. A summary of the encoding scheme for the *U* array is shown in **Figure 2**.

***U* Array Code**	**Definition**	***U* Array Code**	**Definition**
1	**HH**: high price *c* , high price *f*	**5**	**HF only**: *c* sold out, high price *f*
2	**HL**: high price *c* , low price *f*	**6**	**LF only**: *c* sold out, low price *f*
3	**LH**: low price *c* , high price *f*	**7**	**HC only**: high price *c*, *f* sold out
4	**LL**: low price *c*, low price *f*	**8**	**LC only**: low price *c*, *f* sold out

Figure 2: Summary of *U* array codes - Static Overbooking Policy

B. Dynamic Overbooking Policy

After completing the static overbooking policy solution, we moved onto the dynamic overbooking policy that allowed the airline greater flexibility. In contrast to the static method, the dynamic method did not force the airline to commit to an allowed number of overbooked coach seats at the beginning of the year, but instead allowed the airline to overbook up to 120 coach seats over time. On any given day, the airline could choose to sell no coach tickets, which would force the day's demand for coach tickets to 0. In this case, the airline would have three options for coach tickets each day: high price, low price, or no sale. This way, the airline had the added flexibility of deciding to stop overselling coach tickets based both on how many tickets they have sold as well as how many days are left until take-off. For the first-class tickets, however, there were no new options as the airline still had the original two choices of setting either the high price or low ticket price.

Using an identical terminal condition as was used in the static policy and making only slight variations to the bellman equation code, we solved this solution to optimality and were able to draw comparisons to the static solution. As in the static case, we progressed backwards in time and went through each combination of (*t, c, f*) values to determine what to insert into the *V* array by using the dynamics. In case 1 and case 2, there was no difference in the code except to shift the encoded values in the *U* array of the HF only, LF only, LC only, and LF only decisions to the numbers 7, 8, 9 and 10 from their values in the static program of 5, 6, 7, and 8. The differences came along in cases 3 and 4, because we now had to take into account the airline's ability to sell no coach tickets on any given day if it chose to.

In case 3, this was accomplished by adding one more value for the airline to choose from when first-class seats were unavailable: no coach (NC only). Here, the airline would earn zero revenue today, but the discounted value of tomorrow's value function would be non-zero. The code to add this option is shown below. This particular choice was encoded as an "11" in the *U* matrix.

```
# expected revenue today plus discounted expected revenue tomorrow (NC only)
value_no_coach = 0 + delta * V[t + 1, c, f]
```

Finally, in case 4, in which both ticket classes were available, two more choices needed to be added: NH and NL. Here, the airline would choose to not sell any coach tickets that day, but would still need to decide between charging the high and the low first-class price. The two added lines were as follows.

```
# case 4: coach available, first class available - All 6 choices
# additional choices of NH and NL (force demand of coach to 0)
valueNH = (first_high) * prob_first_high[0] + \
            delta * (prob_first_high[0] * V[t+1,c,f+1]) + (prob_first_high[1] * V[t+1,c,f])
valueNL = (first_low) * prob_first_low[0] +  \
            delta * (prob_first_low[0] * V[t+1,c,f+1]) + (prob_first_low[1] * V[t+1,c,f])
```

These final two options were encoded as "5" and "6" respectively in the *U* array that stored the pricing decision the airline should make for each (*t, c, f*) combination. As in the static policy solution, we determined the maximum expected discounted profit of the airline using the dynamic policy solution by indexing the value: *V[0, 0, 0]*. The results of this procedure are discussed in the Results section of the report. Each encoded value is summarized in **Figure 3** below.

***U* Array Code**	**Definition**	***U* Array Code**	**Definition**
1	**HH**: high price *c* , high price *f*	**7**	**HF only**: *c* sold out, high price *f*
2	**HL**: high price *c* , low price *f*	**8**	**LF only**: *c* sold out, low price *f*
3	**LH**: low price *c* , high price *f*	**9**	**HC only**: high price *c, f* sold out
4	**LL**: low price *c*, low price *f*	**10**	**LC only**: low price *c, f* sold out
5	**NH**: no sale *c*, high price *f*	**11**	**NC only**: no sale *c, f* sold out
6	**NL**: no sale *c*, low price *f*	**-**	-

Figure 3: Summary of *U* array codes - Dynamic Overbooking Policy

C. Simulation Procedure: Static Policy

After arriving at the static policy solution for the values of oversold seats from 5 to 15, we determined the optimal value of this parameter. As shown in the Results section of the report, by allowing 9 oversold coach seats, the airline received the maximum expected discounted profit of $42,134.62. Using this optimal static overbooking policy, we simulated the solution forward 10,000 times to assess its volatility and computed the following measures:

1. Average expected discounted profit
2. Average discounted overbooking cost
3. Volatility of discounted profits
4. Percent of time coach is overbooked
5. Percent of time overbooking costs are incurred
6. Percent of time passengers are kicked off the plane
7. Average number of overbooked seats
8. Average number of passengers kicked off the plane

To accomplish the simulations in the code, we created vectors to store whether or not a simulation resulted in an overbooking, if overbooking costs were actually incurred, whether or not passengers were kicked off the plane, the discounted overbooking cost incurred, and the discounted profits corresponding to each simulation. Then, by looping forward in time from day 0 to day 364 and initializing the number of coach and first-class seats sold to 0, we simulated the revenue earned each day according to the appropriate probability of sale and prices for both ticket classes. We used the encoding scheme shown earlier in **Figure 2** to appropriately separate the results of the optimal pricing strategy indexed each day from the static policy *U* array. Using the encoding of "1" for the HH solution in which the airline would charge the high price for both coach and first-class tickets, we provide an example below of how it was accomplished in code as well as the general structure of the simulation code.

```python
# loop through each simulation
for sim in range(nsim):

    # initialize variables
    c = 0      # on the first day, 0 coach seats sold
    f = 0      # on the first day, 0 first-class seats sold
    tN = 365

    total_revenue = 0                     # we haven't made any money yet
    coach_seats_sold_vec = np.zeros(tN)   # store how many c seats sold
    first_seats_sold_vec = np.zeros(tN)   # store how many f seats sold
    price_vec = np.zeros(tN)

    # loop forward in time
    for t in range(tN):                   # time = 0, 1, 2, ..., 364 (we make revenue then)
        opt_price = U_static_opt[t, c, f]
        price_vec[t] = opt_price

        # HH
        if opt_price == 1:
            prob_sale_c = prob_coach_high[0]
            prob_sale_f = prob_first_high[0]
            actual_price_c = coach_high
            actual_price_f = first_high
```

This procedure was repeated for all remaining case codes defined back in **Figure 2.** Using the prices and probabilities of sale, daily revenue was calculated, discounted appropriately to time 0, and summed with the running total of revenue thus far. If it was determined a sale was made in either ticket class that day, *c* and *f* were incremented by 1, appropriately. The code snippet that performed these steps is shown here.

```
# toss coin with appropriate probability from above
sale_coach = (np.random.random(1) < prob_sale_c)        # True if sale, False if no sale
sale_first = (np.random.random(1) < prob_sale_f)        # True if sale, False if no sale

# update c and f
c = int(c + sale_coach)
f = int(f + sale_first)

# update total profit
total_revenue = total_revenue + (sale_coach * actual_price_c + sale_first * actual_price_f) * delta ** (t)

# save the results
coach_seats_sold_vec[t] = c
first_seats_sold_vec[t] = f
```

Finally, after the simulation reached the end of day 364, we assessed whether or not overbooking costs were incurred and if they were, we computed their discounted magnitude. Using the np.random.binomial function, we simulated the number of passengers who showed up for both ticket classes. We then computed the appropriate overbooking costs depending on how many empty seats were available in first-class to bump up oversold coach passengers. Ultimately, the cost for each simulation was discounted back to time period 0 and was subtracted from the simulation's discounted revenue to arrive at the discounted expected profit.

D. Simulation Procedure: Dynamic Policy

In an almost identical procedure to the static policy simulations, the dynamic policy solution was simulated forwards 10,000 times in order to calculate the same statistics as the static method. The only differences made to the static policy simulation code was to index the *U* matrix returned from the dynamic policy solution and adjust the if conditions to incorporate all encoded values up to code “11” as previously defined in **Figure 3**. The results of the simulation are discussed below.

III. Results

A. Static Overbooking Policy

After solving the static overbooking policy for overbooked coach seat values from 5 to 15 seats, it was determined that the optimal policy allowed for 9 overbooked coach seats and corresponded to an expected discounted profit of $42,134.62, as shown in **Figure 4**. To examine the relationship between the expected discounted profits and the number of allowed overbooked seats, we created the plot displayed in **Figure 5** below. As the number of overbooked seats rose from 5 to 9, the curve approached its maximum value of expected profit and appeared to tail off linearly as the number of overbooked coach seats increased above 9. This plot suggests that there was a roughly quadratic relationship between the expected profit and overbooked coach seats using the static policy.

Overbooked Seats	Expected Discounted Profit
5	$41,886.16
6	$42,011.22
7	$42,085.54
8	$42,122.17
9	$42,134.62
10	$42,132.9
11	$42,123.67
12	$42,111.03
13	$42,097.42
14	$42,084.11
15	$42,071.74

Figure 4: Expected Discounted Profit – Static

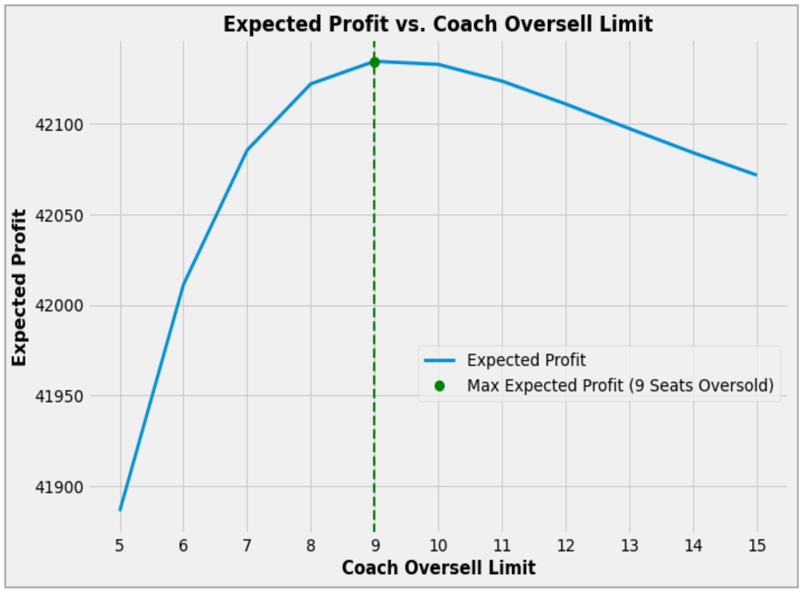


Figure 5: Expected Discounted Profit vs. Coach Oversell Limit – Static

B. Dynamic Overbooking Policy

After solving the dynamic policy solution, we found that the expected discounted profit when the airline was afforded the flexibility of choosing to force any given day's demand for coach tickets to 0 was $42,139.89. When compared to the static policy solution, it is clear the dynamic solution resulted in a marginally higher expectation of discounted profit by approximately $5.27. Therefore, our team determined that the dynamic overbooking policy would be the optimal overbooking pricing strategy for the airline.

C. Simulation Results

In order to determine movement around the expected discounted profit for both the static and dynamic overbooking policies, we recorded the results of 10,000 simulations for each solution by moving forward in time. After this procedure, we were able to obtain the following summary statistics for each method.

Statistic	Static Policy - Optimal	Dynamic Policy
Percent of time coach was overbooked	100.00%	100.00%
Percent of time overbooking costs incurred	80.74%	81.55%
Percent of time passengers kicked off the plane	68.88%	70.48%
Average number of overbooked seats	8.17	8.37
Average number of passengers kicked off	2.22	2.33
Average discounted overbooking cost	$826.61	$869.38
Average discounted expected profit	$42,137.42	$42,141.99
Standard deviation of discounted profit	$935.88	$943.81
Hypothesis test: t - statistic	0.299	0.223

Figure 6: Summary of Simulation Results - Static and Dynamic Overbooking Policies

When comparing the results of the static and dynamic overbooking policies, it can be seen that the measures of percent of time that coach was overbooked, percent of time overbooking costs were incurred, and the percent of time passengers were kicked off the plane were nearly identical, although all were slightly higher in the dynamic case. When comparing the average discounted overbooking cost, discounted expected profit, and standard deviation of expected profit, it can be seen that the trend continued as these measures in the dynamic simulation exceeded those in the static case. The expected discounted profit in the dynamic policy slightly outperformed that of the static solution by approximately $4.57 in our simulation. Although the expected profit was higher in the dynamic solution, so was its standard deviation, indicating the dynamic solution may be slightly more volatile. Interestingly, in both policy solutions, the airline chose to overbook in 100% of the simulations, but not all simulations resulted in overbooking costs being paid. This statistic hovered around 81% in both policy simulations, indicating that there was evidence that the airline would profit from an overbooking policy.

In order to assess the efficacy of these overbooking policies, we computed the expectation of discounted profits when the airline was not allowed to overbook coach seats (*coach_oversell* = 0). As shown in the table, when either the static overbooking policy or the dynamic overbooking policy was restricted in the sense that the airline was not allowed to overbook any coach seats, the expected discounted profit obtained by this suboptimal solution was $40,654.29. By not implementing the static overbooking policy, the airline would be leaving nearly $1,480.33 of expected profits on the table for nearly no added computational cost. This means that the airline would be missing out on an increase in expected profits by 3.64% if they did not allow for coach seats to be oversold. Likewise, when compared to the more flexible dynamic solution, not allowing for coach seats to be oversold was even more suboptimal as it resulted in the airline foregoing nearly $1,485.60 of expected profits by limiting itself to this rigid pricing policy. This is evidence to support the efficacy of overbooking policies, and the dynamic overbooking policy in particular. The airline has a clear financial incentive to oversell coach.

Overbooking Policy	Coach Oversell Allowance	Expected Discounted Profit
No Overbooking	0	$40,654.29
Static - Optimal	9	$42,134.62
Dynamic	20	$42,139.89

Figure 7: Comparison of Overbooking Strategies to No Overbooking

Additionally, the average number of overbooked seats was slightly higher in the dynamic solution as it reached 8.37 seats while the static metric was only 8.17. This was interesting to observe. The respective bar plots are shown below in **Figures 8** and **9** on the following page. In the static solution, the airline faced a hard cap and was locked into selling up to 109 coach tickets. In light of this information, it was not unsurprising that in a majority of simulations, the airline did just that and oversold coach by exactly 9 tickets. In a few simulations, the airline did not fully overbook to this limit and as a result, the distribution had a leftward skew. When looking at the dynamic solution, once again, the most frequent number of overbooked coach seats was 9, which was interesting. However, because the dynamic solution allowed for up to 120 coach seats to be sold, in around 1,500 of our simulations, more than 9 coach seats were oversold, contributing to its higher mean value. This distribution, in comparison to the static result, was slightly more normal in shape and was not skewed as strongly to the left. These plots show that in either case, the airline overbooked coach 100% of the time and more often than not oversold coach tickets by 9 seats.

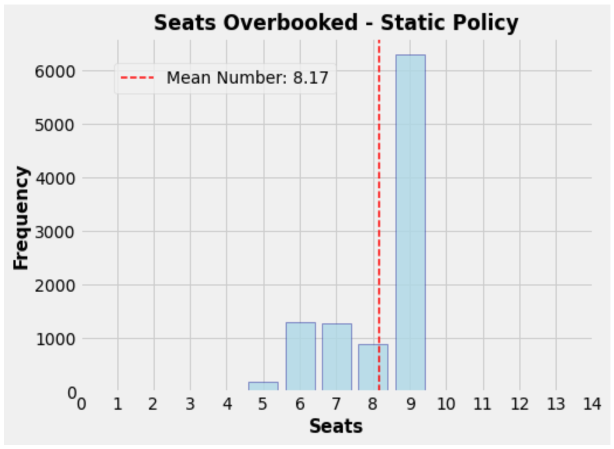


Figure 8: Simulated Oversold Seats - Static Policy

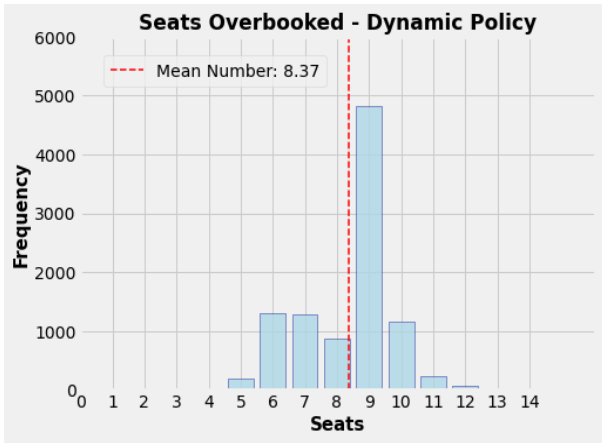


Figure 9: Simulated Oversold Seats - Dynamic Policy

Taking a closer look at the average number of customers kicked off the plane in **Figure 10** and **11** below, it can be seen that once again, the average metric was higher for the dynamic solution than the static policy, although minimally so (2.33 vs. 2.22 removed passengers). When examining the two bar plots side-by-side, it was difficult to identify much of a difference between the plots as their shapes were very similar and both were skewed to the right in the same manner. Looking at the levels of the lower number of removed passengers, however, it was determined that the static policy recorded more instances of 0 passengers being removed from the plane than the dynamic solution, while all other levels were comparable. Combining this observation with the static policy's slightly lower mean indicated that the static method was superior in this regard. Given this information, it is possible that implementing the static overbooking policy may result in fewer disgruntled passengers, on average, when compared to the dynamic policy. It would be important to incorporate a measure of customer dissatisfaction and experience into further analysis, if that data were to become available to the airline. However, in its absence, we still observe that the dynamic policy generated a superior expected discounted profit at a minimal cost to the airline.

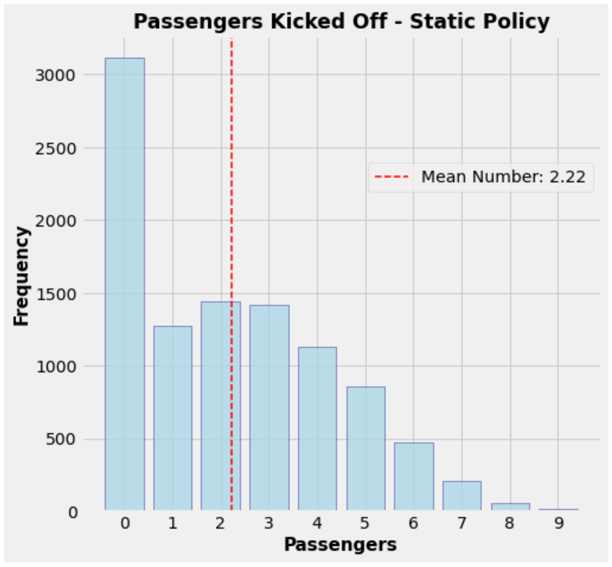


Figure 10: Simulated Passengers Kicked Off - Static Policy

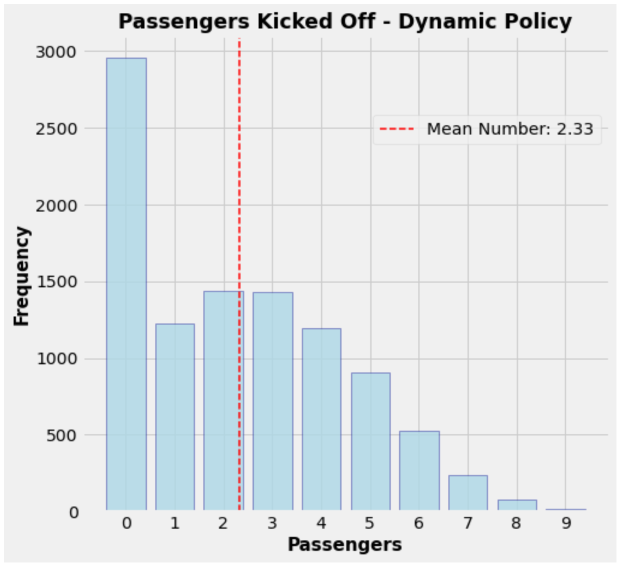


Figure 11: Simulated Passengers Kicked Off - Dynamic Policy

Next, to take a closer look at the simulated results for expected discounted profits and average discounted overbooking costs, we generated the next few figures shown below. Looking first at the average discounted overbooking costs for each of the pricing strategies, it can be seen that once again, their distributions were incredibly similar across the two policies. Although the average discounted overbooking cost in the static case was roughly $42.77 lower than in the dynamic simulation, it appeared as though both policies resulted in very similar cost distributions. However, it is important to note that there were a few more simulations in the static case that resulted in zero overbooking costs as its frequency slightly exceeded 3,000, while this same count was slightly lower than 3,000 in the dynamic results.

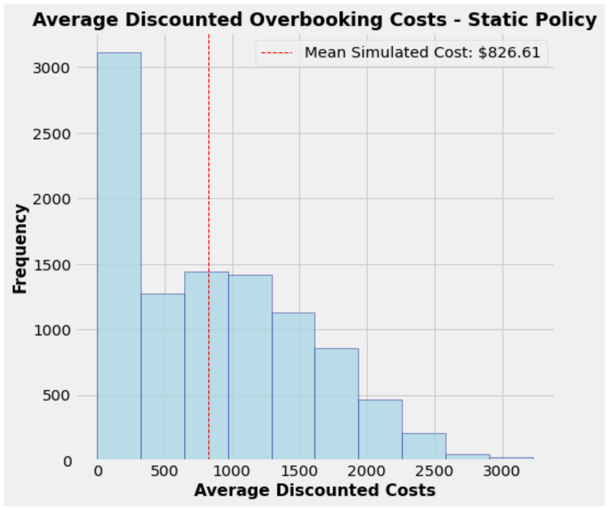


Figure 12: Simulated Avg. Overbooking Costs - Static Policy

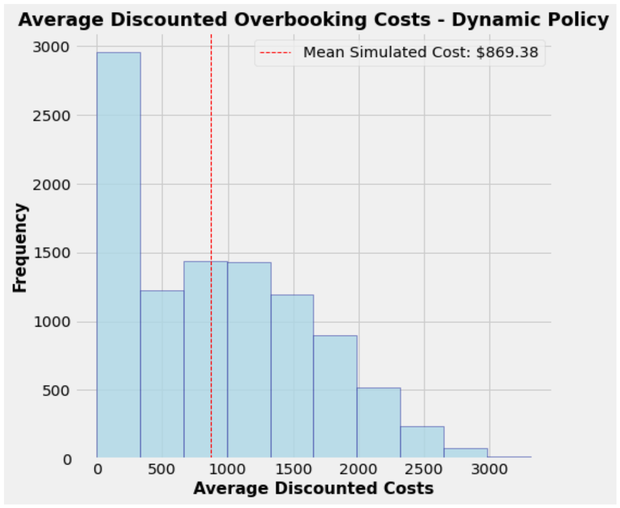


Figure 13: Simulated Avg. Overbooking Costs - Dynamic Policy

Next, we examined the expected discounted profits more closely. For the static overbooking policy, we generated **Figure 14** below. As can be seen, the average simulated expected discounted profit was found to be $42,137.42, which was plotted with a black dashed line in the figure. Just to the left of this measure, plotted with a blue dashed line, was the optimal expected profit of $42,134.62 obtained earlier when coach was oversold by 9 seats. As expected, these values were incredibly close to one another. To assess whether there was any statistical difference between the mean of the simulation and the optimal profit value, we ran a t-test and found that we failed to reject the null hypothesis that the means were statistically different. This is shown by the t-statistic computed in **Figure 6** of 0.299, whose value was less than 1.96, the critical t-value in this two-tailed test. The result of this hypothesis test provided support for the validity of our static overbooking solution as the results of the simulation very closely mirrored it.

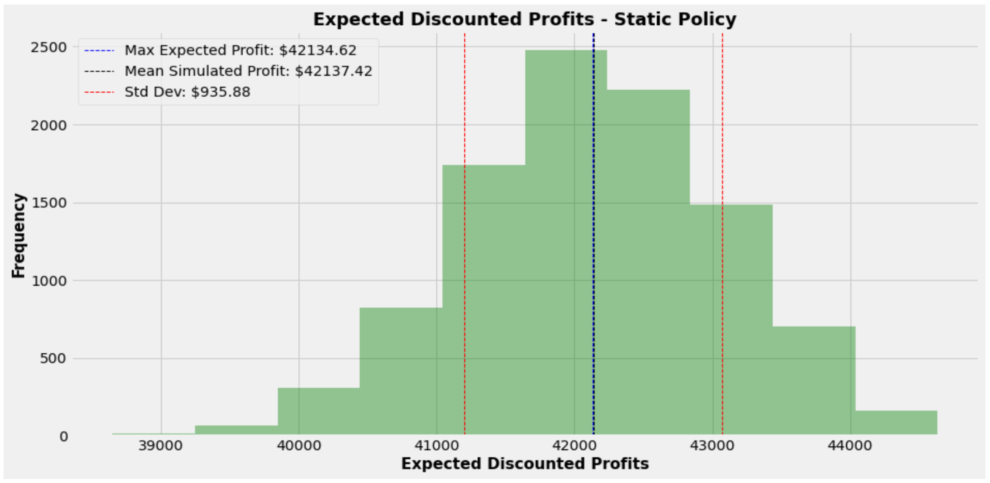


Figure 14: Histogram of Expected Discounted Profits – Optimal Static Policy

Next, to compare the simulation results of the static program to the dynamic policy, we created the same histogram of expected discounted profits, as shown in **Figure 15**. Like the static policy results, the mean simulated expected profit value of $42,141.99 very closely reflected the solution obtained when solving the original dynamic programming problem: $42,139.89. Again, we ran a hypothesis test to assess the statistical difference between these measures and obtained a t-statistic (0.223) less than the critical t-value (1.96) for this two-tailed test. Therefore, we concluded that the simulated results supported the validity of our dynamic programming solution.

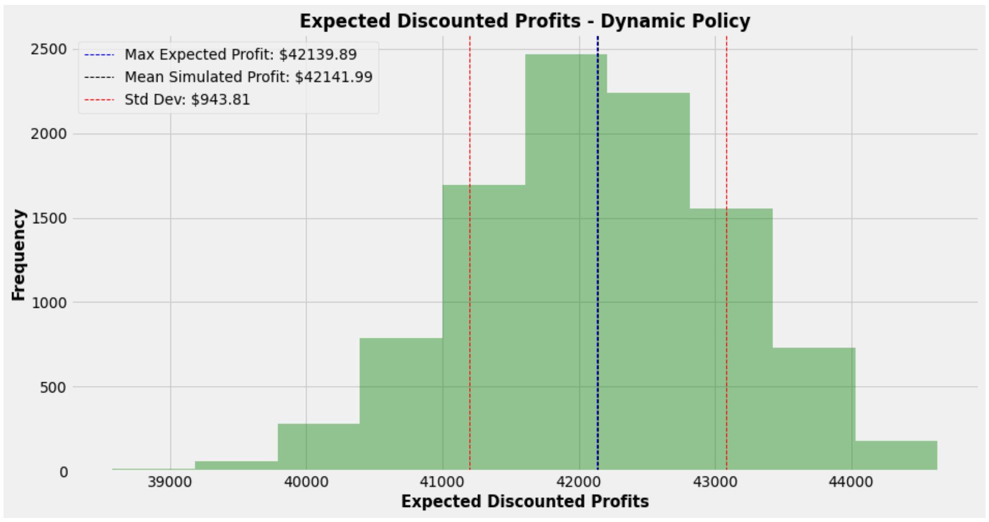


Figure 15: Histogram of Expected Discounted Profits – Dynamic Policy

To get a better sense of the variability around the mean of the simulated results in the measure of the expected discounted profit, we computed a 95% confidence interval around this statistic. To perform these confidence interval calculations, we used the following formula and variable definitions:

$\bar{x}$ = sample mean
s = sample standard deviation
N = number of simulations
$t_{\alpha/2,\, df}$ = t statistic ($\alpha = 0.05$)

$$95\%\ Confidence\ Interval = \bar{x} \pm (t_{0.025,\, N-1}) * \frac{s}{\sqrt{N}}$$

After applying this formula to both the static and dynamic policy solutions, we obtained the results summarized in **Figure 16** below.

Overbooking Policy	Average Expected Discounted Profit	Average Standard Deviation of Expected Discounted Profit	95% C.I. around Average Expected Discounted Profit
Static - Optimal	$42,137.42	$935.88	[$42,119.08, $42,155.77]
Dynamic	$42,141.99	$943.81	[$42,123.49, $42,160.49]

Figure 16: 95% Confidence Intervals around Avg. Expected Discounted Profit - Static and Dynamic Policies

As can be seen in the results above, the 95% confidence intervals for both the static and dynamic policy solutions existed very tightly around each policy's average simulated expected discounted profit. Additionally, these intervals also contained their respective original solutions in both the static and dynamic results ($42,134.62 and $42,139.89). When comparing the actual intervals to each other, it can be seen that the confidence interval built around the dynamic policy's solution was slightly higher on both ends than the corresponding bounds in the static solution. This provided further evidence that the dynamic policy was optimal when compared to the static policy, and would be expected to lead to a higher discounted profit for the airline. These narrow intervals gave precise estimates for the expectation of future profit and provided confidence in our projected dynamic profit solution of $42,139.89.

IV. Conclusion and Recommendations

After analyzing both the static and dynamic methods, our team determined that implementing overbooking policies would be effective for the airline. Moreover, we determined that utilizing the dynamic overbooking solution was optimal due to its increased flexibility for the airline. It resulted in a higher expected discounted profit than the static policy, and led to the maximum expected discounted profit of $42,139.89, which was approximately $5.27 higher than the static solution.

Compared to the suboptimal solution where the airline could not oversell any coach tickets, both the static and dynamic solutions improved upon this baseline profit of $40,654.29 by nearly $1,480.33 and $1,485.60, respectively. This demonstrates the effectiveness of overbooking policies in increasing the airline's expected profits. In order to further assess the validity of our solutions, we ran 10,000 simulations in each case to compute the average expected discounted profit as well as build 95% confidence intervals around these solutions. We found that the average expected discounted profit of the dynamic solution exceeded its static counterpart, as it did in the original policy solution. Additionally, in both simulation analyses, the airline overbooked 100% of its flights, but only incurred overbook costs around 81% of the time. Because implementation of these policies would lead to higher expected profits over the baseline, the airline would benefit financially from their immediate implementation.

Despite the clear boost in expected profit, the team recognizes that utilizing these policies could result in unintended consequences for the airline as frequent overbookings may lead to frustrated passengers who take their business elsewhere. Therefore, in order to guarantee its long-term viability, the airline would also need to consider its impact on customers when selecting the optimal pricing strategy.