Optimization 2 – RM294

Proj 2 – Effectiveness of Overbooking : Dynamic Programming

Group 25 – Surya Prasad Pamireddygari (sp49882), Soumik Choudhuri (sc64856), Rushiil Deshmukh (rsd949), Mahika Bansal (mb62835)

OBJECTIVE

Overbooking has been very common in the airline industry. It is prevalent owing to no-shows and cancellations, which might lead to loss of revenue. As it's difficult to predict such instances, overbooking can be an effective strategy. But it can also affect customer experience, which might lead to churn and eventually heavy losses.

We're trying to find a trade-off so that we can use this strategy. The flight in focus has 2 ticket classes: coach and first-class, both having two possible ticket prices each. Though overbooking first-class is not recommended, we can explore how we can use the same for coach. In case of surplus of customers for coach, they can be bumped to first-class to compensate for their experience

As analysts of the sales department, we must figure the effectiveness of overbooking – optimal pricing and number of tickets to be offered. The objective would be to maximize the expected discounted profit, i.e. revenue of tickets minus the overbooking costs. Price charged will be decided for each day and revenue will be generated from the demand before takeoff. Overbooking penalty will be offered to passengers who couldn't be allotted a seat.

ASSUMPTIONS

The assumptions are as follows:

1. Sales of tickets in first class and coach are independent of each other.

2. Probability of sale depends on the price of the tickets

3. More demand than seats for a particular class wouldn't lead to the surplus booking the second class

4. Plane departs on all 365 days, with 100 seats in coach and 20 in first-class. The prices for coach tickets: $300 and $350 whereas for first-class: $425 and $500. Each day, the probability of tickets being sold are as follows:

    a. $300 – 65%
    b. $350 – 30%
    c. $450 – 8%
    d. $500 – 4%

5. Coach ticket-holders show up with a probability of 95% and first-class with 97%.

6. Cost of bumping passengers to first class is $50, and $425 to bump off the plane.

7. Discount rate is 15% per year, and daily discount factor is 1/(1+0.15/365)

POLICY 1: SETTING A HARD CAP ON NUMBER OF SEATS THAT CAN BE SOLD

For this, first we analyse the use-case by setting a hard cap at 5 seats for overbooking. The first step would be to identify cost of bumping customers to first-class and then figure expected profit.

For 100 or lesser tickets sold, there will be no extra cost accrued. For 100+, customers will be bumped to either first-class based on the availability or bumped off the flight entirely. Also, passengers might not show up based on their probability of showing up.

There will be 2 major vectors: V for expected profits and U, separated for coach and first-class tickets. The set up for dynamic programming starts by finding expected overbooking cost at day 365, being the terminal condition, iterating through each first seat and coach seat combination. Using probabilities of showing up, calculation of expected cost was done with upgrade and bump off costs.

With value function, a similar loop was run in reverse. Post this, 4 scenarios were analysed to form the value matrix – no coach & no first left, either of them left or both available. In the first case, value function: discount rate*V[coach, sold, first sold, t+1].

```
# filling the all possible values of coach and first class seats with expected cost at time tN( at time of departure)
for coach_sold in range(num_coach):
    for first_sold in range(num_first):
        revenue = 0
        for i in range(coach_sold+1):
            for j in range(first_sold+1):
                if i > coach_seats:
                    prob = binom.pmf(i, coach_sold, coach_show) * binom.pmf(j, first_sold, first_show)
                    num_plane_bump = max(i-coach_seats,0)
                    num_first_bump= min(first_seats-j,num_plane_bump)
                    temp = (first_bump_cost*num_first_bump + (num_plane_bump-num_first_bump)*plane_bump_cost)
                    revenue += temp*prob

        V[coach_sold, first_sold , tN-1] = revenue
```

For second and third, it would be: P(high/low price) * high/low price + discount rate * sum(V[coach/coach+1,first/first+1,t+1])

For the last: same equation, just need to figure max of the 4 cases.

```
for t in reversed(range(tN-1)):
    for i in range(num_coach):
        for j in range(num_first):

            # all are sold out
            if (i == num_coach-1) and (j == num_first-1):
                V[i, j, t] = discount * V[i, j, t+1]
                U[i, j, t] = 0

                #coach is sold out and first class is available
            elif (i == num_coach-1) and (j < num_first-1):
                V_CH_FH = first_H * first_H_prob[0] + discount*(first_H_prob[0]*V[i, j+1, t+1] + first_H_prob[1]*V[i, j,
                V_CH_FL = first_L * first_L_prob[0] + discount*(first_L_prob[0]*V[i, j+1, t+1] + first_L_prob[1]*V[i, j,
                V_CL_FH = V_CH_FH
                V_CL_FL = V_CH_FL
                V[i, j, t] = max(V_CH_FH, V_CH_FL, V_CL_FH, V_CL_FL)
                U[i, j, t] = np.argmax([V_CH_FH, V_CH_FL, V_CL_FH, V_CL_FL])+1

                #first class is sold out and coach class is available

            elif (i < num_coach-1) and (j == num_first-1):
                V_CH_FH = coach_H * coach_H_prob[0] + discount*(coach_H_prob[0]*V[i+1, j, t+1] + coach_H_prob[1]*V[i, j,
                V_CL_FH = coach_L * coach_L_prob[0] + discount*(coach_L_prob[0]*V[i+1, j, t+1] + coach_L_prob[1]*V[i, j,
                V_CH_FL = V_CH_FH
                V_CL_FL = V_CL_FH

                if no_selling:
                    V_N_FH = discount*(coach_H_prob[0] * V[i, j, t+1] + coach_H_prob[1] * V[i, j, t+1])
                    V_N_FL = discount*(coach_L_prob[0] * V[i, j, t+1] + coach_L_prob[1] * V[i, j, t+1])
                    V[i, j, t] = max(V_CH_FH, V_CH_FL, V_CL_FH, V_CL_FL, V_N_FH , V_N_FL)
                    U[i, j, t] = np.argmax([V_CH_FH, V_CH_FL, V_CL_FH, V_CL_FL, V_N_FH , V_N_FL])+1
                else:
                    V[i, j, t] = max(V_CH_FH, V_CH_FL, V_CL_FH, V_CL_FL)
                    U[i, j, t] = np.argmax([V_CH_FH, V_CH_FL, V_CL_FH, V_CL_FL])+1
```

```
            # coach and first class seats  are available
        else:
            V_CH_FH = (coach_H + first_H) * first_H_prob[0] * coach_H_prob[0] + \
            coach_H * coach_H_prob[0] * first_H_prob[1] + first_H * first_H_prob[0] * coach_H_prob[1] + \
            discount*(coach_H_prob[0] * first_H_prob[0] * V[i+1, j+1, t+1] + coach_H_prob[0] * first_H_prob[1]*V[i+1,
                      coach_H_prob[1] * first_H_prob[0]*V[i, j+1, t+1] + coach_H_prob[1] * first_H_prob[1]*V[i, j, t+


            V_CH_FL =  (coach_H + first_L) * coach_H_prob[0] * first_L_prob[0] + coach_H * coach_H_prob[0] * first_L_
            first_L * first_L_prob[0] * coach_H_prob[1] + discount*(coach_H_prob[0] * first_L_prob[0] * V[i+1, j+1, t
                      coach_H_prob[0] * first_L_prob[1]*V[i+1, j, t+1] + coach_H_prob[1] * first_L_prob[0]*V[i, j+1,
                      coach_H_prob[1] * first_L_prob[1]*V[i, j, t+1])


            V_CL_FH =  (coach_L + first_H) * coach_L_prob[0] * first_H_prob[0] + \
            coach_L * coach_L_prob[0] * first_H_prob[1] + first_H * first_H_prob[0] * coach_L_prob[1] + \
            discount*(coach_L_prob[0] * first_H_prob[0] * V[i+1, j+1, t+1] + coach_L_prob[0] * first_H_prob[1]*V[i+1,
                      coach_L_prob[1] * first_H_prob[0]*V[i, j+1, t+1] + coach_L_prob[1] * first_H_prob[1]*V[i, j, t+


            V_CL_FL =  (coach_L + first_L) * coach_L_prob[0] * first_L_prob[0] + \
            coach_L * coach_L_prob[0] * first_L_prob[1] +  first_L * first_L_prob[0] * coach_L_prob[1] + \
            discount*(coach_L_prob[0] * first_L_prob[0] * V[i+1, j+1, t+1] +  coach_L_prob[0] * first_L_prob[1]*V[i+1
                      coach_L_prob[1] * first_L_prob[0]*V[i, j+1, t+1] + coach_L_prob[1] * first_L_prob[1]*V[i, j, t+

            # IF NO SALE IS AN OPTION
            if no_selling:
                V_N_FH = first_H * first_H_prob[0] + discount*(first_H_prob[0] * V[i, j+1, t+1] + first_H_prob[1]*V[i
                V_N_FL = first_L * first_L_prob[0] + discount*(first_L_prob[0] * V[i, j+1, t+1] + first_L_prob[1]*V[i

                V[i, j, t] = max(V_CH_FH, V_CH_FL, V_CL_FH, V_CL_FL, V_N_FH, V_N_FL)
                U[i, j, t] = np.argmax([V_CH_FH, V_CH_FL, V_CL_FH, V_CL_FL, V_N_FH, V_N_FL])+1
            else:
                V[i, j, t] = max(V_CH_FH, V_CH_FL, V_CL_FH, V_CL_FL)
                U[i, j, t] = np.argmax([V_CH_FH, V_CH_FL, V_CL_FH, V_CL_FL])+1
```

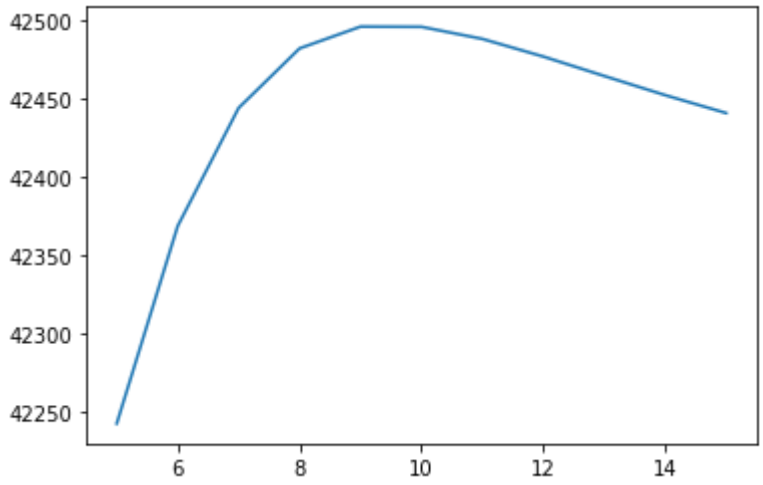
Result:

For hard cap at 5, the expected profit would be at **$ 42242.86**.

After this, we tried to find expected revenue for caps between 6-15 seats. The results:

```
The expected profit with  5  over booking is $ 42242.86219879507
The expected profit with  6  over booking is $ 42368.86899301223
The expected profit with  7  over booking is $ 42444.241213317226
The expected profit with  8  over booking is $ 42482.211237591255
The expected profit with  9  over booking is $ 42496.112304378745
The expected profit with  10  over booking is $ 42495.97195739834
The expected profit with  11  over booking is $ 42488.259535782025
The expected profit with  12  over booking is $ 42476.996208701945
The expected profit with  13  over booking is $ 42464.59794015481
The expected profit with  14  over booking is $ 42452.36220993067
The expected profit with  15  over booking is $ 42440.94484423895
```

So the max profit of **$ 42496.11** was obtained at the hard cap of **9** seats available for overbooking



POLICY 2: HAVING NO HARD CAP ON NUMBER OF SEATS

```
1 V, U = DP_solve(True,20)
2 print('The expected profit with 20 over booking is $',V[0,0,0])
```

```
The expected profit with 20 over booking is $ 42502.67326124381
```

In this situation, there are 3 choices for every coach seat – high price, low price, or no sale. With the assumption that no more than 120 tickets can be sold for coach on a single day, we obtained the maximum profit at **$42502.673** which is higher than policy 1, with 9 as the hard cap.

SIMULATED FINDINGS

To properly test and contrast the two policies, we created 1000 simulations of each policy and created plots for certain metrics of evaluation. The simulation computes the optimal price decision at time t using all possible combinations of coach and first class seats left for all possible number days until take-off.

We use the code snippets shown below to compare the two policies using the following metrics:

- Percentage of passengers overbooked
- Percentage of passengers removed from plane
- Average over-booking cost
- Volatility

```
def DP_forward_simulations(U, overbook):

    coach_Values = np.arange(coach_seats+overbook+1) # all possible number of coach seats left
    first_Values = np.arange(first_seats+1)  # all possible number of first class seats left
    tValues = np.arange(T+1)  # all possible days until take off
    num_coach=len(coach_Values)
    num_first=len(first_Values)
    tN=len(tValues)
    revenue,c,f=[0,0,0]


    for t in range(tN-1):
        # getting the optimum price decision at time t
        price_opt = U[c, f, t]

        coach_prob,coach_price,first_prob,first_price=[0,0,0,0]

        # setting coach and first prices based on the optimum price decision
        if (price_opt==1) |(price_opt==2):
            coach_prob = coach_H_prob[0]
            coach_price = coach_H

        if (price_opt==3) |(price_opt==4):
            coach_prob = coach_L_prob[0]
            coach_price = coach_L

        if (price_opt==1) |(price_opt==3):
            first_prob = first_H_prob[0]
            first_price = first_H

        if (price_opt==4) |(price_opt==2):
            first_prob = first_L_prob[0]
            first_price = first_L
```

```
        if price_opt == 5:
            first_prob = first_H_prob[1]
            first_price = first_H

        if price_opt == 6:
            coach_prob = first_L_prob[1]
            first_prob = first_L


        if c == num_coach - 1:
            coach_prob = 0
        if f == num_first - 1:
            first_prob = 0

        coach_sale = (np.random.random(1) < coach_prob)
        first_sale = (np.random.random(1) < first_prob)
        revenue += (coach_sale*coach_price + first_sale*first_price) * discount**t
        c += coach_sale
        f += first_sale


     # number of people showed up
    coach_showup = np.random.binomial(c, coach_show)
    first_showup = np.random.binomial(f, first_show)

    oversold = max(coach_showup-coach_seats,0)
    first_bump = min(first_seats-first_showup,oversold)

    profit = revenue
    num_overbook = max(0, c - coach_seats)
    cost_overbook = (first_bump_cost*first_bump + (oversold-first_bump)*plane_bump_cost)
    bump_out_plane = max(oversold-first_bump, 0)

    return profit, num_overbook, cost_overbook, bump_out_plane
```

Code for creating simulations

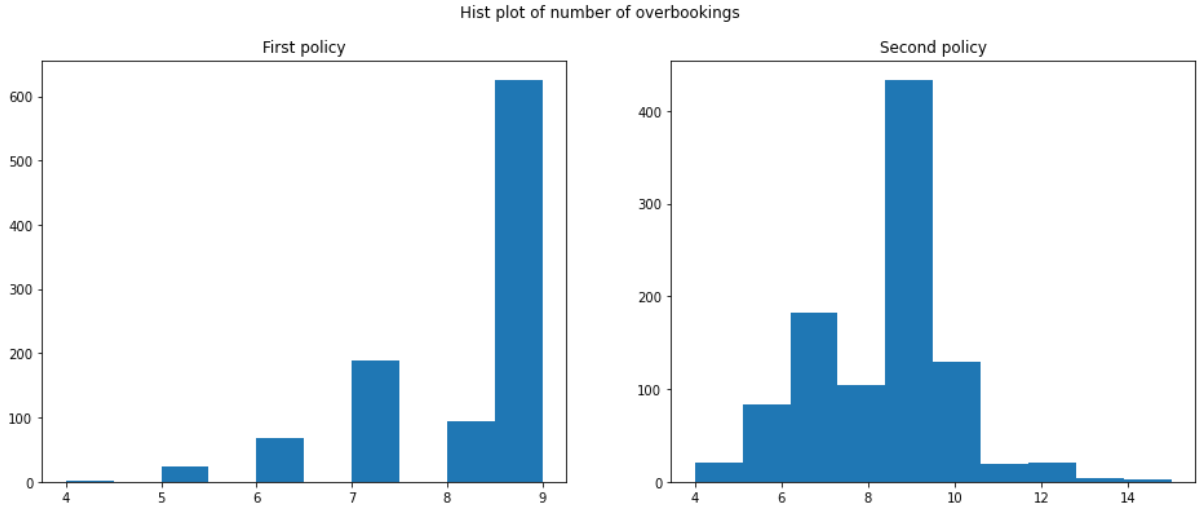
```
nsim = 1000
profit_fpolicy = np.zeros(nsim)
num_overbook_fpolicy = np.zeros(nsim)
cost_overbook_fpolicy = np.zeros(nsim)
bump_out_plane_fpolicy = np.zeros(nsim)
profit_spolicy = np.zeros(nsim)
num_overbook_spolicy = np.zeros(nsim)
cost_overbook_spolicy = np.zeros(nsim)
bump_out_plane_spolicy = np.zeros(nsim)

# getting optimal value fucntion and decision at the best value of 9 over bookings and No sale option with 20 overbookings
V, U_fpolicy = DP_solve(False,9)
V, U_spolicy = DP_solve(True,20)

for i in range(nsim):
    profit_fpolicy[i], num_overbook_fpolicy[i], cost_overbook_fpolicy[i], bump_out_plane_fpolicy[i] = DP_forward_simulations
    profit_spolicy[i], num_overbook_spolicy[i], cost_overbook_spolicy[i], bump_out_plane_spolicy[i] = DP_forward_simulations
```

Code for executing simulations

Following are the results of each policy on the mentioned metrics:

Percentage of passengers overbooked

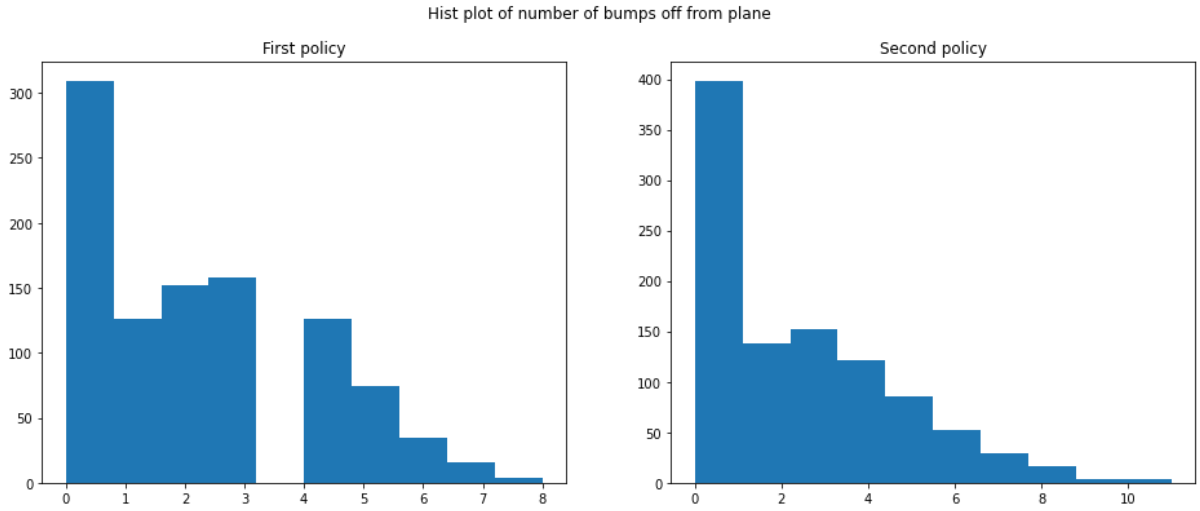
For both policies, we see our simulations overbooking 100% of the time.



Policy 1 favors 9 overbooked seats. Policy 2 has a larger range of overbook seats, with over 40% of the simulations chose 9 as their number of overbooked seats, despite the second policy allowing for up to 20 overbooked seats.

Percentage of passengers removed from plane

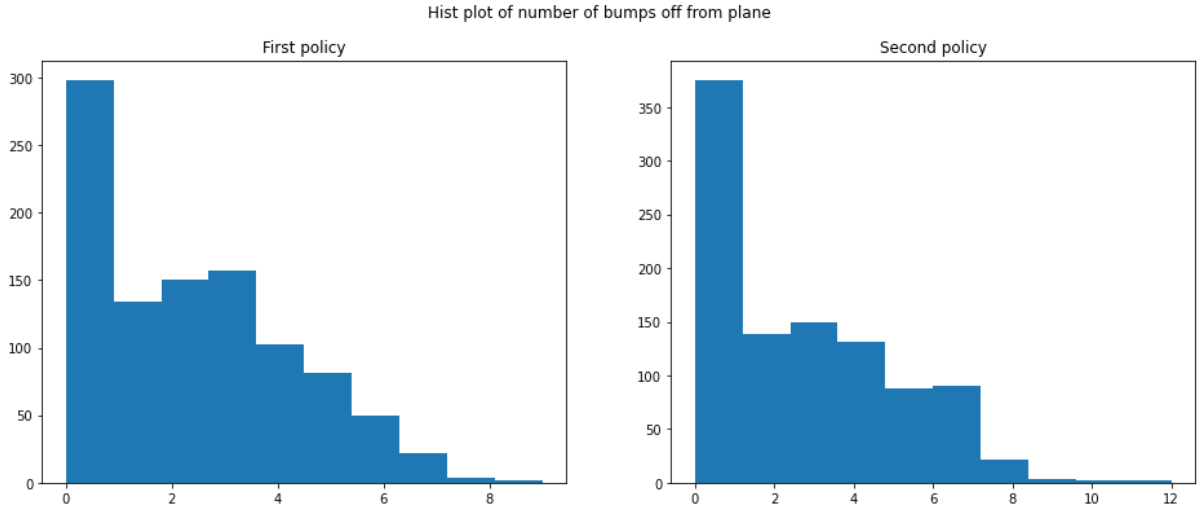
The percentage of times passengers removed from the plane in the first policy is 69.1%, whereas that for the second policy is 73.3%. Thus, the percent of times passengers are removed from the plane are lower in the first policy than in the second policy.



Policy 2 has a few instances where more than 8 people needed to be removed, but a higher number of instance where less than 2 passengers need to be removed. Despite the few outlier instances, we still get a mean of 2 passengers booted for each policy.

Average over-booking cost

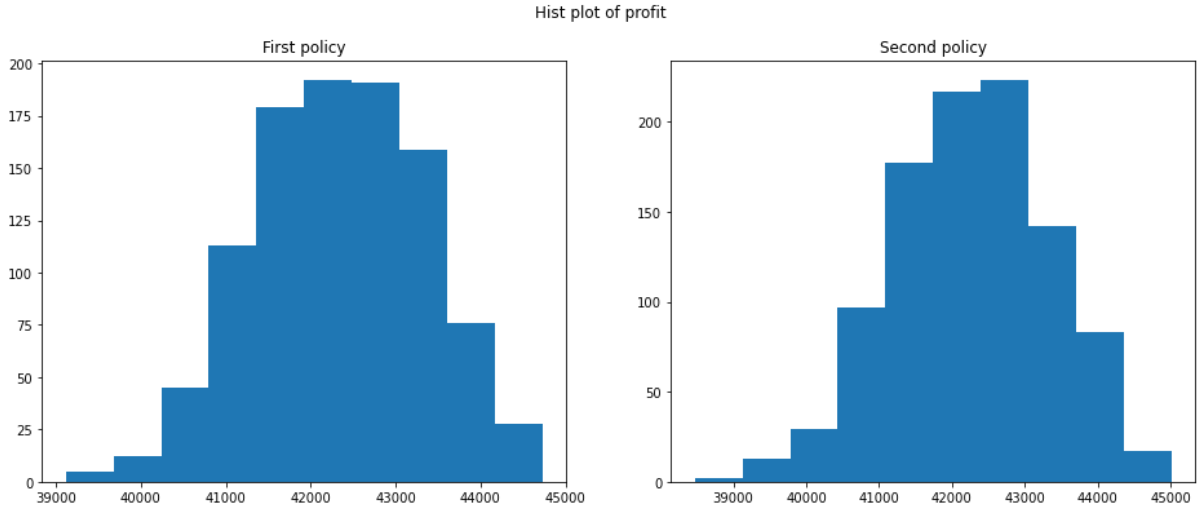
We calculate the overbooking costs for each policy for each iteration and take an average to form a baseline for comparison.



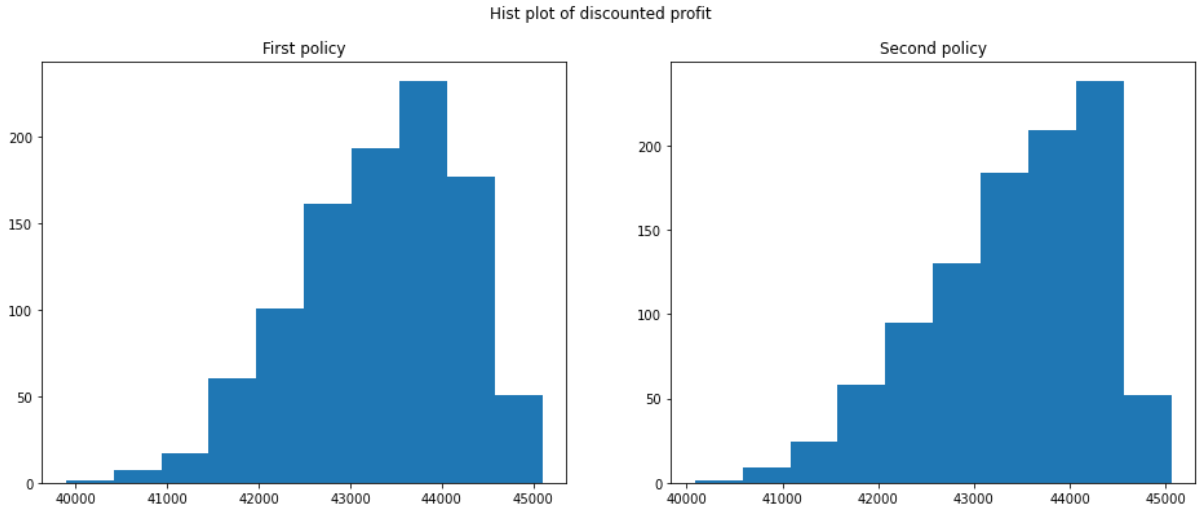
As we can see, the overbooking costs for the second policy is slightly lower at **$42276.08** than that of the first policy, which a value of **$42332.64**. This can initially suggest that the second policy might be more profitable in the long run, however an analysis of volatility will help us better understand the distribution of profits for each policy.

Volatility of profits

As mentioned above, we measure the volatility of profits for both policies.



The first policy has a lower volatility at **$880.792** compared to that of the second policy at **$891.88**. This means that the standard deviation of profits for the second policy is higher than that of the first policy, which could be a negative considering our current risk appetite. However, the mean discounted profit value for the second policy at **$43416.95** is higher than that of the first policy at **$43319.14**. Thus, the second policy is overall more profitable for us to adopt than the first policy.



CONCLUSION

This project relies heavily on dynamic programming principles to make an optimal decision about overbooking flights with intention of maximizing profit. We built a function to examine the expected discounted profit when coach is overbooked by 5 tickets and found it to be **$42242.86**. We then repeated the process by iterating our function from 6-15 seats and found that the optimal overbooking policy is 9 seats, which shows the largest profit at **$42,496.11**.

We then included a situation where we do not sell any coach tickets and designed a new policy to accommodate this condition in our programming with a cap on the number of coach tickets at 120. We repeated the function above with this third condition and computed the profit (with 20 over bookings) to be **$42502.67**. This is higher than profit from the first policy.

Following this, we simulated the two policies 1000 times to compare them using evaluation metrics such as percentage of overbooked passengers, percentage of passengers removed from the plane, average overbooking costs and volatility of profits. Based on these metrics, we conclude that policy 2 has a higher average profit, with a greater volatility in profit values. This could be considered for a higher risk threshold to provide higher returns over a longer period of time.