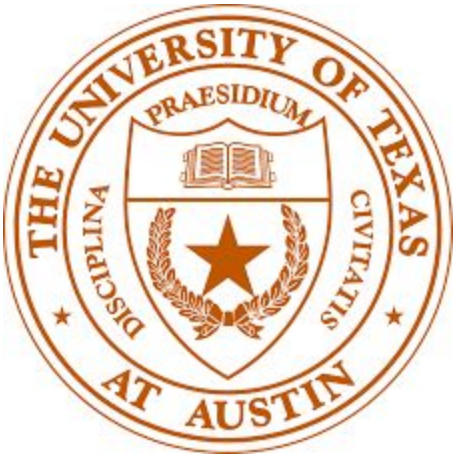
Airline Price Optimization

MS in Business Analytics - 2020
Pricing and Revenue Management Term Project Report



Team:

Ravikiran Bobba - RB44274,
Mohammed Saqib Asghar - MA58893
Satya Pachigolla - SP46958
Sachin Balakrishnan - SB56372
Arjun Rao - AKR732

1: Introduction:

The airline industry always stays under constant pressure to increase revenue. Pricing is one area of focus that offers vast potential to airlines looking to grow revenue and stand out in today's crowded marketplace. Ultimately, if airlines engage in their pricing strategies to help them tell their story, they can increase customer satisfaction, discover unique points of differentiation, and achieve total revenue optimization.

Here in this project, we as consultants to Aviato.com have documented our attempt of developing an algorithm capable of optimizing the sale of flights for airline companies to maximize their profit, and selling this algorithm to clients. Spirit Airlines was Aviato.com's first major client. While the product started off okay, recently the algorithm has not been performing well, and Spirit Airlines has demanded that Aviato.com improves its performance, or else the airline will no longer require its services. To solve this Avaito.com, has enlisted the help of a team of business analysts from UT Austin.

2: Project Overview:

The airline industry is a competitive market which causes different airlines to take different approaches to conquer their corner of the market. Spirit Airlines operates as a budget airline, where they sell the tickets for a low price and try to add on additional expenses to the customers. They believe that the customer should only pay for the services they use. For example, any services other than the seat such as other services, like checked bags, carry-on bags, food and drinks, and boarding passes, are charged separately. One aspect of Spirit's business model is they don't allow for refunds of plane tickets.

Optimization of Pricing dictates the revenue especially for the airlines that operate in the budget air segment. There are several Time-series models developed to forecast the demand, based on multiple parameters like date, day of the week, holidays, previous year demand, etc. These models help the airlines plan for pricing based on the demand. However this forecasting

is possible for Major players who have enough data from the prior years. The goal of this project is to create an optimized pricing plan, by using simulation of demand scenarios. The project does not focus on prediction of demand which is a data science project, but it focuses on the optimization for a random distribution of demand_levels.

3: Problem Statement and Approach:

3a: Problem Statement:

The problem statement was from a Kaggle competition which aims to identify the best possible pricing function which optimizes the revenue of the company for a demand_level which varies uniformly between a certain set of thresholds. The aim of the project is to create a pricing_function to maximize the total revenue collected for all flights in a simulated environment.

For each flight, pricing_function will be run once per (simulated) day to set that day's ticket price. The seats that weren't sold today will be available to sell tomorrow, unless the flight leaves that day. pricing_function is run for one flight at a time, and it takes the following inputs:

- Number of days until the flight
- Number of seats they have left to sell
- A variable called demand_level that determines how many tickets you can sell at any given price.

The quantity you sell at any price is: **quantity_sold = demand_level - price**

Ticket quantities are capped at the number of seats available, and the function will output the ticket price. The variable "demand_level" follows a uniform distribution between 100-200 every

day. Once the daily demand level is known, we set a price for the ticket using the “pricing function.”

3b: Project Evaluation:

A certain set of scenarios are simulated for running multiple demand_levels which were randomly generated and the average revenue was calculated for each of the pricing functions. This ensures the testing of versatility of the model under varied ranges of tickets_available and days_before_departure. The Python functions used to evaluate are attached in the appendix section and code was shared.

3c: Approach:

As discussed above, the pricing function takes in three parameters:

- Days left.
- The Tickets remaining.
- The Demand level.

```
def pricing_function(days_left, tickets_left, demand_level):
    """Sample pricing function"""
    price = demand_level - 10
    return price
```

Then we ran different pricing functions in a simulator to test how well it performs on a range of flight situations.

```
import sys
sys.path.append('../input')
from flight_revenue_simulator import simulate_revenue, score_me
```

We started by creating a baseline model as a benchmark to compare the revenue. Then we progressed towards relatively complex models by adding the days and the demand as

parameters, In the final trials we even explored using dynamic programming to optimize the pricing for a given scenario. The detailed description of different models used are discussed in the modelling section

4: Modeling:

4a: Model1:

```
def pricing_function(days_left, tickets_left, demand_level):

    price = demand_level - 10
    return price
```

- By setting the price as 'demand_level - 10', we are making sure that we are selling 10 tickets every day irrespective of what the demand is.
- This gave us an expected average revenue of **$6073.**
- In a case where total number of days = 7 and total tickets are 50, since we are selling 10 tickets every day, we'll run out of spots 3 days before the flight.

4b: Model2:

```
def pricing_function(days_left, tickets_left, demand_level):

    price = demand_level - tickets_left/days_left
    return price
```

- Now, instead of selling 10 tickets every day, we divided the total number of tickets by the total number of days and suggested selling the same number of tickets for all the days.

- This makes sure that we have tickets to sell for every day. This model gave us an expected average revenue of **$6354.**

4c: Model3:

```
def pricing_function(days_left, tickets_left, demand_level):

    price = demand_level - ((tickets_left/days_left)* (demand_level-100)/50)
    return price
```

- In Model2, we will be filling the same number of seats every day irrespective of the demand. But ideally, we'll want to fill in more seats on days the demand level is high because we'll get more revenue from these seats.
- In Model3, we are adding an extra multiplier which increases as the demand level increases. This model increased our expected average revenue to **$6896**

4d: Model4:

```
def pricing_function(days_left, tickets_left, demand_level):

    if days_left==1:  # selling all the remaining tickets
       price = demand_level-tickets_left

    elif demand_level >= 185 and tickets_left >= demand_level/3:    #if the demand is high, we are selling half the tickets
        price = min(demand_level-(tickets_left/days_left), demand_level - (demand_level/3))

    elif demand_level >= 150: # if the demand is in the top 50%, we are selling equal number of tickets every day
        price = demand_level-(tickets_left/days_left)

    else: # if the demand is low, we are just selling one ticket that day
        price = demand_level-1

    return price
```

- Now further modifying Model3, we are adding in more tiers based on the demand level.

- Using if-else statements, we are adding conditions which allows the tickets sold to be high.

  if the demand level is high (if demand level > 185 - we are selling 1/3rd of remaining tickets,

  and if the demand level is between 150 and 185, we are selling that day's share of tickets) and if the demand is low (demand level is less than 150) we are selling only one ticket that day.
- We are also making sure that we sell all the leftover tickets on the last day before the flight. We expected good results from this tiered strategy but this gave us an expected revenue of **$5943**

4e: Model5:

```
def pricing_function(days_left, tickets_left, demand_level):

    if days_left > 14:
        if demand_level >= 180:
            price = demand_level - ((demand_level - 100)/10)
            return price
        else:
            price = demand_level
            return price
    if days_left >= 2 and days_left <=14:
        if demand_level >= 150:
            price = demand_level - ((demand_level - 100)/ 10)
            return price
        else:
            price = demand_level
            return price
    if days_left == 1:
        price = demand_level - tickets_left
        return price
```

- We further tweaked the code in Model4 by adding in more conditions. Here we added in rules which considers the number of days left as well.
- If there are more than 2 weeks until the flight, we are only selling tickets if the demand is high (>= 180) and if the demand is low, we are not selling any tickets.
- Similarly, if the number of days until the flight are between 2 and 14 days, we are selling tickets if the demand level crosses 150. And on the final day, we are selling all leftover tickets.
- This strategy increased the expected revenue to **$7380.**

4f: Model6:

```
demand_list = []
avrg_demand = demand_list

def avrg_calc(demand_level):
    avrg_demand.append(demand_level)
    return np.mean(avrg_demand)

def std_demand(demand_level):
    return np.std(avrg_demand)

def pricing_function(days_left, tickets_left, demand_level):
  average_demand = avrg_calc(demand_level)  #Average demand
  STD_demand = std_demand(demand_level)     #Standard deviation of demand

  if days_left > 14:
    if demand_level >= average_demand + (STD_demand):   #If demand is 1 standard deviation above the average
      price = demand_level - ((demand_level-100)/10)
      return price
    else:
      price = demand_level
      return price
  if days_left >= 2 and days_left <= 14:
    if demand_level >= average_demand + (0.5*STD_demand):  #If demand is 0.5 standard deviation above the average
      price = demand_level - ((demand_level-100)/10)
      return price
    else:
      price = demand_level
      return price
  if days_left == 1:   #Last day
    price = demand_level - tickets_left
    return price
```

- Tweaking the model further by only selling when the days remaining is greater than 14 and the demand level is greater than 1 standard deviation above the mean demand.
- Similarly, if the number of days until the flight are between 2 and 14 days, we are selling tickets if the demand level is greater than 0.5 standard deviation above the mean.
- Then we sell the remaining tickets on the last day.
- We were able to optimize the revenue to **$7403.**

4g: Model7:

```
def pricing_function(days_left, tickets_left, demand_level):

    if days_left == 1:
        price = demand_level - tickets_left
        #price = demand_level/2
    else:
        #we will sell tickets only if demand crosses a threshold
        #the optimum demand is calculated by checking where the derivative reaches maximum
        fraction = (tickets_left / (50 * days_left)) ** 0.5
        demand_treshold = int(200 - fraction*100)
        if demand_level >= demand_treshold:
            tickets_to_sell = int(tickets_left / (days_left * fraction))
            price = demand_level - tickets_to_sell
        else:
            #if the demand is not above the threshold, we are not selling any tickets
            price = demand_level
    return price
```

In Model5, after realising that being selective about selling tickets based on the demand level is a good strategy we decided to further improve it.

In Model7, we decided to sell tickets only on days where demand is above a certain threshold, thus we had to calculate what the ideal threshold is.

As a simple case, if we choose to sell tickets on 100% of days, the expected revenue can be written as follows:

$$revenue = (150 - \frac{tickets}{days}) * tickets$$

If we decide to sell tickets only on fraction of days, where demand is higher than a threshold, we get the following formula:

$$revenue = (200 - 50 * fraction - \frac{tickets}{days * fraction}) * tickets$$

(upper bound of demand = 200, lower bound of demand = 200 - 100 * fraction. Hence expected value is 200 - 50 * fraction). Then we need to maximise revenue with respect to fraction, where tickets and days are known constants. After taking a derivative and equating it to zero, we get the following:

$$fraction = \sqrt{\frac{tickets}{50 * days}}$$

This pricing strategy boosted our expected average revenue to **$7513**

4h: Model8:

Till Model 7, we tried to optimize the price mostly based on heuristics, differentiation (using calculus) to find the maximum value under the assumption that we sell an equal number of tickets on each day and segmentation. In Model 8 we tried to leverage the power of Dynamic programming to solve this optimization problem.

Dynamic programming works backward and tries to optimize the value of the objective. Here in our case the Bellman Equation is

V[day, tickets] = max(price*(tickets sold in the day) + V[days-1, tickets-tickets sold in the day])

- **V is the value function for days before departure and tickets left**

Here in the simulation, the Value is to be maximized and Price can vary over the range from 0 to demand level and the value of the price, which maximizes the output for the given day, tickets_left will be returned as the output of the function. The average revenue of this model turned out to 6310$

```
def pricing_function (days_left, tickets_left, demand_level):


  t=int(tickets_left)
  d=int(days_left)
  dl=int(demand_level)

  dValues=np.linspace(start=0, stop=d, num=d+1)
  tValues=np.linspace(start=0, stop=t, num=t+1)
  dN=len(dValues)
  tN=len(tValues)

  V = np.zeros( (dN, tN) )
  U = copy.deepcopy(V)
  for d in (dValues):
    for t in (tValues):
    #for each time value the loop through the possible values of Tickets
    #cat(d,t)
      if(d==0):
        # Boundary condition
        V[int(d), int(t)]=0
      elif(t==0):
        # Boundary Condition
        V[int(d), int(t)]=0
      else:

       # Bellman Equation
        p = np.linspace(start=(dl-t), stop=dl, num=dl+1) # checking for all values of prices less than demand level
        valueChoices=np.linspace(start=(dl-t), stop=dl, num=dl+1) # Storing the value for each price
        for index in range(len(p)):
          valueChoices[index]= p[index]*max(min(t,dl-p[index]),0) +V[int(d-1),int(t-max(min(t,dl-p[index]),0))]
        #print(valueChoices)
        #print(d,t,max(valueChoices))
        V[int(d), int(t)]=max(valueChoices)
        #print(V)
        U[int(d), int(t)]=p[np.argmax(valueChoices)]
#print(V)

return(U[int(dN-1),int(tN-1)])
```

4: Comparison of Models:

Model	Remarks	Average Revenue
Model1	10 tickets per day	$6073
Model2	Equal number of tickets all days	$6354
Model3	Selling more on high demand days	$6896
Model4	Tier based on demand	$5943
Model5	Tier based on demand and days left	$7380
Model6	Using mean and std of demand	$7403
Model7	Maximising a quadratic equation	$7513
Model8	Dynamic Programming	$6310

5: Conclusion:

The project gave us an insight of how pricing of a product has a major impact on the revenue of the organization. Even the basic intelligent models based on simulations, heuristics, and calculus can result in very high lift in the observed revenue for the company. Even for companies and organizations where demand is stochastic like airlines, simulations can be used to iterate several possible scenarios and the best decision to optimize the revenue.

- From developing our algorithms, we determined that the best approach for Avaito.com to base their model on the logic in Model7. In this model, we decided to sell tickets only on days where the demand crosses a specific threshold, and we selected this threshold by solving a quadratic equation
- From Dynamic programming we think that the random generation of demand level, which was not known when the price of the previous level was evaluated, could be the reason for its inefficient result. If the demand levels were known in advance, it can be incorporated into the DP to get the optimal results.

6: Appendix:

This section contains the relevant links and Code used in the project.

6.1: Problem Statement:

The problem statement was from this Kaggle competition and further information on the overview and applications can be found here:

https://www.kaggle.com/dansbecker/airline-price-optimization-micro-challenge

6.2: Github Repository of Codes:

The entire codes that were used for all the models were updated on github, and can be found here:

https://github.com/RavikiranBobba/Airline-Price-Optimization

6.3: R Code used for Dynamic Programming:

```
pricing_function <- function(days_left, tickets_left, demand_level){

t=tickets_left
d=days_left
dl=demand_level

#possible values of "d" and "t"
dValues=seq(0,d)
tValues = seq(0,t)
dN=length(dValues)
tN=length(tValues)

#Value function matrix
V = matrix(NA,dN,tN)
rownames(V) = dValues # add rownames and colnames to the V matrix
colnames(V) = tValues

#Action matrix
U = V # copy V matrix with rownames and colnames, to U matrix

for (d in (dValues)){
  for (t in (tValues)){
```

```
 #for each time value the loop through the possible values of Tickets
        if(d==0){
        # Boundary condition
        V[paste(d), paste(t)]=0
        }else if(t==0){
        # Boundary Condition
        V[paste(d), paste(t)]=0
        }else
        {
        # Bellman Equation
        p = seq(0,dl) # checking for all values of prices less than demand level
        valueChoices=seq(0,dl)
        for (index in 1:length(p)){
      valueChoices[index]=
                p[index]*max(min(t,dl-p[index]),0)+V[paste(d-1),paste(t-max(min(t,dl-p[index]),0
        }
        V[paste(d),paste(t)]=max(valueChoices)
        U[paste(d),paste(t)]=p[which.max(valueChoices)]
        }
 }
}
return(U[d+1,t+1])
}
```