Optimal Airline Overbooking Seat Number Simulation

Sam Brady

May 3, 2021

Abstract

For this project I built a simulation based off of real flight data that aimed to answer the question: what is the optimal number of tickets for airlines to overbook in a day? I created three distinct overbooking strategies and compared them. The first strategy is overbooking a constant number of seats throughout the day. The second is overbooking a constant amount of seats until mid-day (noon), after which overbooking ceases. Finally, the third strategy is a decreasing number of overbooked seats thought the day, at various amounts.

Lately I have been searching for interesting use cases for simulation based approaches to analytics. After reading a report on how American Airlines uses simulation to predict and budget for the number of spare engines and parts [6], I thought about how I could apply the same analytical approach to my preferred airline, Hawaiian Airlines. According to a report from the NY Times [1], overbooking seats is a huge challenge for airline analysts. I also came across a Monte Carlo method to answering the question of the optimal number of overbooked tickets [7] per flight; their approach to this problem was a great foundation for me to build into a more complex simulation in which I tried to account for all of the variability and nuances of airline data.

The global pandemic ended up giving this project an interesting twist. When I went to find flight data, I quickly realized that many of the airlines have changed their routes to accommodate for the decrease in bookings. Looking into Hawaiian Airlines routes I found (at least for the day that I was looking) that they had an interesting solution to this problem, where they were rerouting international or outer island mainland bound flights to only fly out of Honolulu once per day. Of course this makes great sense financially and made my simulation all the more relevant as I'm sure the analysts typically rely on historical data or software to predict the optimal overbooking number. I imagine they had to try their best to figure out a new overbooking number or strategy due to this unprecedented pandemic.

Background and Description

The airline industry is ultra competitive. Many major airlines struggle to turn a profit, which means that they must operate on razor sharp margins. On top of this, the number of seats in a given flight is a very fixed number. A surprisingly large number of passengers sometimes opt to cancel their flights for unknown reasons, often without warning. Many customers, especially business class passengers, have free cancellation policies which can lead to the airlines

losing money. To make up for this loss it is fairly common for airlines to overbook the number of seats sold, thereby assuring a full flight and optimizing revenue. This of course leads to some passengers being bumped unwillingly, although airlines typically ask for volunteers to give up their seats in exchange for flight vouchers and other perks. Choosing the right number of seats to overbook is a balancing act of maximizing seat profit while minimizing the voucher costs.

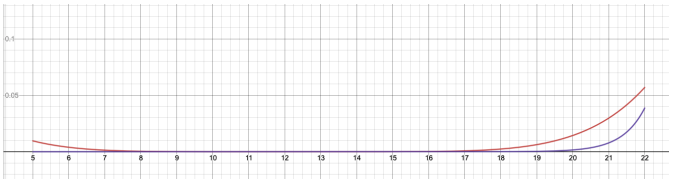
While this simulation isn't mean to be perfect it certainly improves upon the model discussed earlier, in terms of accounting for the variability in flight data. This is just a project simulation that relied on generated data to gain insights. Airline analysts would have a great deal of historical data to work off of, however I had to make many assumptions in this project. I will do my best to explain them as they arise, as well as go over other areas for improvement.

To get a sense of the number of flights an airline like Hawaiian had per day, I went to a flight tracker website [4] and took the outgoing flights for one day from all of the major airports on each island in Hawai'i. For the sake of simplicity I took only the flights to and from these major island airports. I also found a distribution of the Airlines' delay times which I incorporated into the simulation as well [2].

All of the code was written in R and various functions can be found in the appendix of this report.

Filling the Seats

The first part of this simulation and the main foundation was the generation of seats bookings. I wrote a function to fill any number of seats depending on the plane size and class of seats. Hawaiian Airlines' fleet has three classes of seats: main cabin, extra comfort, and first class. For this simulation only focused on main cabin and extra comfort. This function also accounts for a number of unsold seats for flights that leave at undesirable times. Analysts would typically have lots of historical data to model with, but here I had to make many assumptions. The first assumption I made is that seats are usually booked in groups. Rather than generating the seats one at a time I relied on group sizes, which I also varied throughout the day. I theorized that there are smaller group sizes in the morning, say for business travelers, and larger groups mid day, for families, etc. I felt that group sizes would be an accurate way to approach filling the seats, but it is important to note that I generated the probabilities for these group sizes based off of my best guess, not actual data. The probabilities of my group size were, consequently, my second assumption. The third assumption was the empty seat rate, which I had fluctuate according to the time of day. I thought it would be relevant to add these empty seats in as otherwise it assumes that all flights are full. That said, in the context of the pandemic, flights may actually be full far more often than usual as there are a much more limited number of flights. Either way I came up with some functions to fluctuate the empty seat rate for each class of seats. Here are the graphs of the functions modeling the probabilities of empty seats. I used these fluctuating probabilities to generate a random number of empty seats according to a binomial distribution using the rbinom() function.



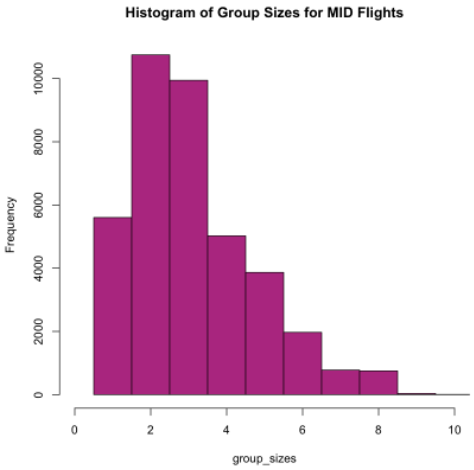
I split the day into four parts (morning, mid-day, afternoon , and evening) to come up with a discrete probability function generating the group size for each time interval. I used the sample() function to randomly sample a group size from 1 to 10 given the probabilities. There could, of course, be bigger groups, such as sports groups or school trips, but this is just a rough generalization. The approach for the function is to generate one group size at a time until the flight is full, and then remove groups. Below is the discrete pmf for the mid-day time interval, a sample of group sizes for a random mid-day flight, and a histogram for the amount of each group size for a mid-day flight.

```
[1072… sample_groups <- fill_seats(120, 12, main_empty_rate(12))
sample_groups
sample_sum <- sum(sample_groups)
sample_sum
```

```
3·4·3·2·5·1·3·1·3·2·4·3·2·5·3·6·1·2·2·3·2·2·2·3·2·3·2·2·3·2·6·1·8·5·6·2·3·1·2·5
```

```
120
```

$$p_X(x) = \begin{cases} 0.15 & x = 1 \\ 0.28 & x = 2 \\ 0.25 & x = 3 \\ 0.13 & x = 4 \\ 0.1 & x = 5 \\ 0.05 & x = 6 \\ 0.02 & x = 7 \\ 0.019 & x = 8 \\ 0.00075 & x = 9 \\ 0.00025 & x = 10 \end{cases}$$



Monte Carlo Method for Groups Cancelling

After seats have filled and group sizes were generated, the next step was to simulate groups canceling their flights. This is the main driver behind airlines overbooking their seats. To simulate this I went with a Monte Carlo approach and set up another discrete pmf to model the different probabilities that various group sizes would cancel. I made the assumption that smaller group sizes would generally have a higher rate of cancellation, due to the fact that they are traveling alone and could make decisions like that more easily. Single seat travelers may also be traveling on business which may or may not hold certain advantages like a free cancellation policy, etc. Again I created a different pmf for each seat class, main and extra comfort. For the Monte Carlo approach a uniform random number is generated and if it exceeds the given probability for that particular group size then the cancellation is accepted [13]. This process is repeated for every group on the plane. Below is a sample of about how many cancellations would

occur in a random mid-day flight, followed by a histogram showing the estimated distribution of the number of seats that were canceled in 1000 simulations.

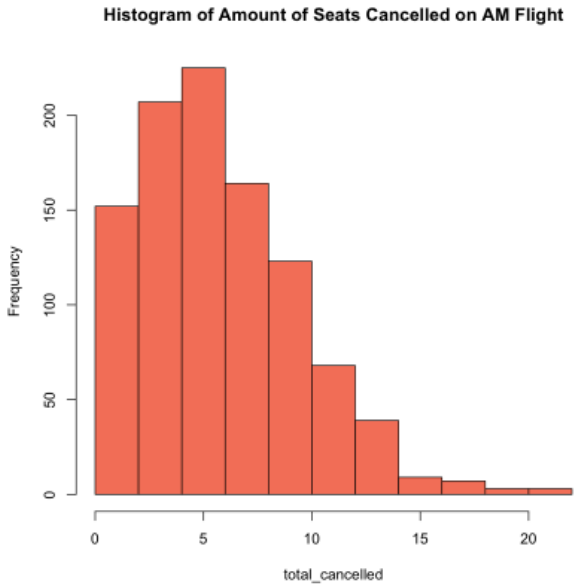
```
[1019…] sample_groups <- fill_seats(120, 12, main_empty_rate(12))
sample_groups
sum(sample_groups)
groups_showed <- groups_cancel(sample_groups, vals, main_cond_probs)
groups_showed
sum(groups_showed)
```

```
2 2 5 6 2 1 2 6 2 5 2 2 2 4 3 5 3 3 3 2 1 2 7 5 6 6 2 3 4 7 4 2 2 7
```

```
120
```

```
2 2 5 6 2 1 2 6 2 5 2 2 2 4 3 5 3 3 3 2 1 2 7 5 6 6 2 3 7 4 2 2 7
```

```
116
```



The Fleet and Revenue

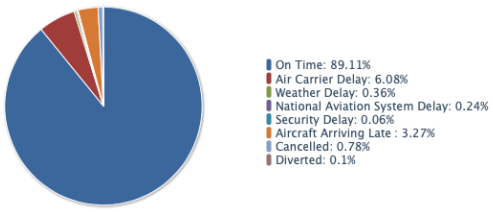
Once I had the number of seats that were to be filled pre-boarding I wrote some functions to model the revenue from each ticket sold. Again, the price of these one way fares was a big assumption and analysts would typically have a substantial amount of historical flight data for each plane, flight, and route. However here I used a normal distribution with a mean and standard deviation for each plane to calculate the cost of each ticket and then multiplied by the size of each group. The rnorm() function allowed me to generate these costs easily. While the costs are fictional, I feel that grouping seat costs together is accurate as most groups buy their tickets together. From the Hawaiian Airlines website [5] I was able to find all of the models in their fleet and then estimated the costs of these seats to what I thought was appropriate given their typical routes (certain planes fly overseas more often than other planes which fly more frequently between islands). Below is a histogram showing the distribution of revenue for a random flight.

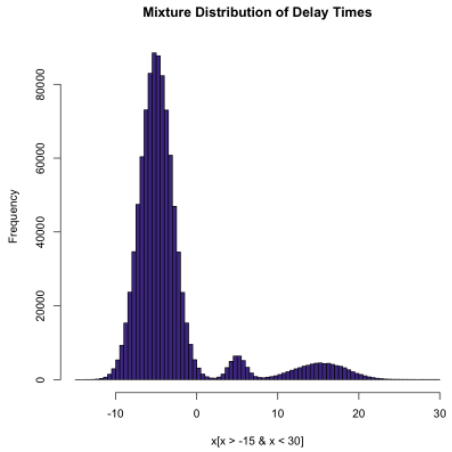
Delay Mixture Distribution

One of the main components in modeling the flights throughout the day is each flights delay time. Now there can be any number of reasons for a delay, each with their own average delay times. Together as a mixture distribution these delays make up the overall flights delay time distribution [12]. It is easy to simulate the mixture through the composition method, where the overall complex mixture random variable can be decomposed into smaller random variables which are more easily sampled from. One of my favorite things about flying Hawaiian is that they have a very low delay rate. By looking at the historical data from the Bureau of Transportation Statistics [2] I was able to create a mixture of Hawaiian Airline's delay times and will use this distribution to sample from for each flight. To keep it simple here I used all Normal distributions to model the delay times, each with their own mean and standard deviations in minutes. To create and sample from a mixture distribution I used the UnivarMixingDistribution() function from the distr package [9]. You can see in the histogram below that most of their flights leave on time or even a few minutes early! Note the multimodal peaks in the distribution from each of the individual distributions.

$$F(X) = (0.8911)N(-5, 2) + (0.0608)N(15, 3) + (0.0036)N(10, 6) + (0.0024)N(12, 8)$$

$$+(0.0006)N(25, 5) + (0.0327)N(5, 1) + (0.0078)N(18, 2) + (0.001)N(45, 10)$$





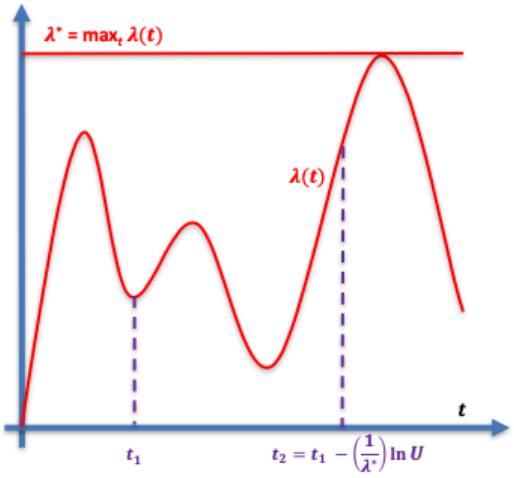
FIFO Boarding, Queue Process, and Upgrades

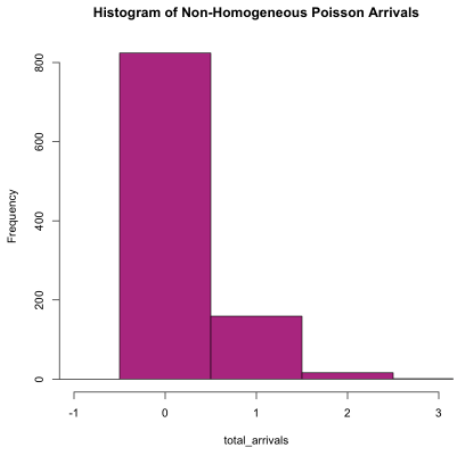
Another key component for this simulation was the queueing process for boarding the aircraft. I decided that a FIFO (first in first out) process would model most accurately what takes place at an airline's gate [11]. The queueing process needs to take into account the passengers who were bumped from their last flight due to overbooking and place them into the cabin first. I also was able to incorporate upgrades for passengers if there were any empty extra comfort seats available. So the first thing the simulation tries to do is take the passengers who were bumped and place them into an upgrade, if it cannot do that, it placed them into the main cabin before other passengers. Again, entire groups are considered rather than single passengers. The vsetdiff() function from the vecsets library came in handy many times while coding the simulation [10]. This function returns the differences in elements of two vectors. So for instance I could easily find the groups of passengers that were or weren't selected for an upgrade by comparing the vectors of groups before and after. I had to write a few helper functions to update the queue

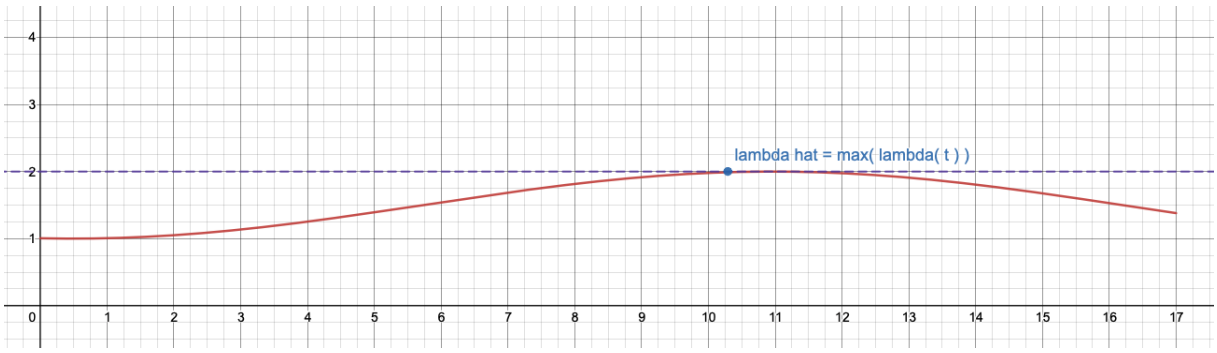
of the next available flight to the same destination given the flight itinerary for the day, also making sure not to overbook on the last flight of the day no matter the overbooking strategy being simulated.

Non-Homogeneous Poisson Process and The Thinning Algorithm

One situation that could potentially arise in any given day at an airport is the arrival of a US Federal Marshall needing to board a flight, which could then potentially bump another passenger out of their seat if the plane was already full. I thought it would be fun to incorporate this into the simulation, but to be quite honest I don't know much about these types of situations. I decided that these arrivals could follow a Poisson process and therefore obey the first two assumptions that go along with this process, namely; that they occur one at a time and that they occur indecently in different time intervals [3]. Rather than keep a steady average through out the day it made sense to me that the average would fluctuate with the hour of the day, following a function shown below. With the appropriate rate, most of the time no Marshalls would show, sometimes there would be one and sometimes there would be two (but working on separate cases, and independent). Therefore this would need to be modeled by a non-homogeneous Poisson process [14]. To sample from a non-homogeneous Poisson process is not a trivial task. To generate this distribution I would use the Thinning Algorithm which is a form of the Acceptance-Rejection method [8][15]. Below is the rate function lambda of t, a diagram of how the thinning process works, some sample arrivals, and a histogram of arrivals.







```
[701]: lambda <- function(t) {  (0.5*sin(0.3*t-(8))+0.6) }

[702]: nonhomogeneous_poisson(lambda, 1, 120, 17)
```

0·0·0·1·1·1·0·0·0·0·0·0·0·0·0·0·0·0·1·0·0·1·0·0·0·0·0·0·0·0·0·0·0·0·1·1·0·1·1·2·0·1·0·0·
1·0·0·0·0·0·1·0·0·0·0·0·0·0·0·1·0·0·0·0·0·0·0·0·0·0·0·0·0·1

Voucher Costs, Hotel Rooms, and the Update Function

Before bumping a passenger an airline must first ask for volunteers, which they will then negotiate a flight voucher for. According to an article from trip savvy [17], the voucher for a volunteered bumped passenger is typically the amount of the one way ticket cost. If a passenger is unwillingly bumped then they are typically awarded a voucher worth 200% of their ticket cost, unless it is more than 4 hours or overnight, which they are then rewarded a voucher worth 400% of their ticket cost plus hotel fees if applicable. This simulation accounts for all of these things but to keep it simple I estimated an 85% volunteer rate and ignored all other fees including things like baggage and transportation. Even though the simulations didn't overbook any seats on the last flight to a given destination there was often still a large queue at the end of the day which resulted in extra hotel fees. This is really where it gets costly for an airline. I used the rnorm() function again to sample from a normal distribution to find the cost of these hotel rooms a sampled a random number for the number of rooms depending on the size of the group. My thinking is that some people in a group would be ok sleeping in the same room and some would require their own room.

Additionally I wrote an update function which would append values to the flight list data frame. This was both useful for summary statistics and to help me trouble shoot the code and make sure it was doing what it was supposed to be doing (there's nothing worse than being stuck in an airport all day). Here is a sample data frame from Lihue (LIH) airport when the overbooking number is 10 through the day.

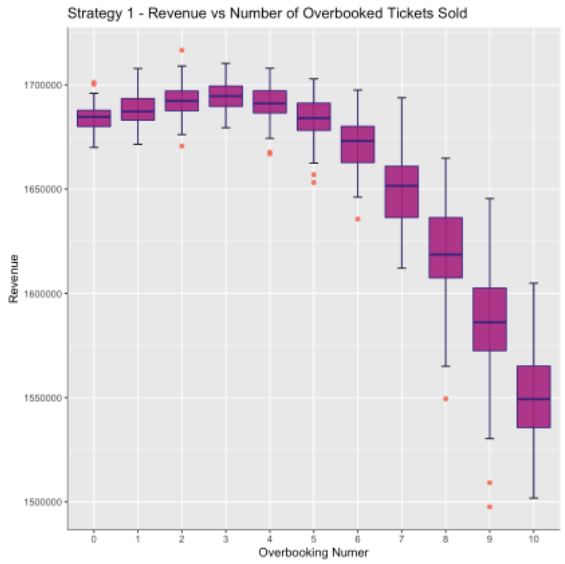
```
[1027… head(airport)
```

A data.frame: 5 × 20

	FLIGHT_NUM	DEPT	DEST	TIME	HOUR	PLANE	LAST	QUEUE	M_SEATS	OVERBOOKED	BOOKED	SHOWED	M_UNF	E_UNF	UPGRADED	MAIN_NO_UP	OVERFLOW	BOARD	DELAY	TAKEOFF
	<chr>	<chr>	<chr>	<chr>	<dbl>	<chr>	<lgl>	<list>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<list>	<dbl>	<dbl>	<dbl>
1	HAL154	LIH	HNL	3/31/2021 7:09AM HST	7.15	B712	FALSE	0	120	10	130	116	4	0	0	116		116	-0.08	7.07
2	HAL224	LIH	HNL	3/31/2021 10:50AM HST	10.83	B712	FALSE	0	120	10	130	128	-8	0	0	128	2, 3, 3	120	-0.08	10.75
3	HAL284	LIH	HNL	3/31/2021 1:50PM HST	13.83	B712	FALSE	2, 3, 3	120	10	130	120	0	0	0	120	5, 3	120	-0.09	13.74
4	HAL324	LIH	HNL	3/31/2021 4:15PM HST	16.25	B712	FALSE	5, 3	120	10	130	120	0	0	0	120	1, 1, 1, 2, 3	120	-0.07	16.18
5	HAL514	LIH	HNL	3/31/2021 7:10PM HST	19.17	B712	TRUE	1, 1, 1, 2, 3	120	0	120	115	5	0	0	115	3	120	-0.10	19.07

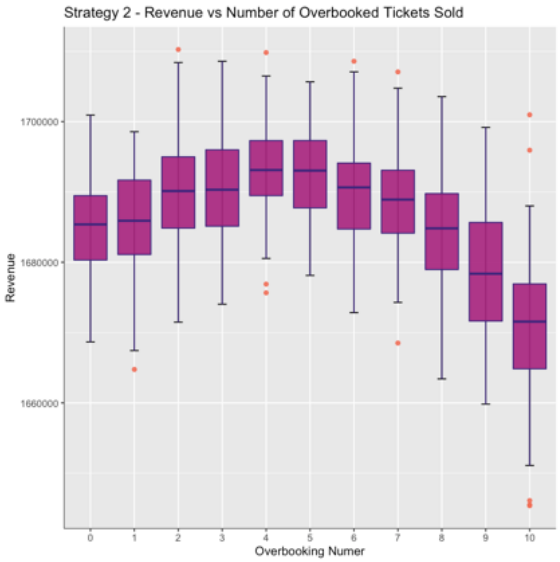
Strategy 1) Overbooking a Constant Number Throughout the Day

Finally it was time to run the simulations. Again there were three strategies being tested to find the optimal overbooking number, and each strategy I ran for 100 simulations of each overbooking number per strategy and gathered the revenue for each. The first strategy was a constant number of overbooked seats throughout the day. So for this I ran 100 simulations each of overbooking 0 seats, 1 seat, 2 seats… up until 10 seats then plotted the revenue spread in a box and whisker chart. This was the easiest way for me to gain an understanding of the spread and scope of each overbooking number. Observing the data it is easy to see that the first few values produce decent revenue and then it quickly drops off as the number of overbooked seats increases.



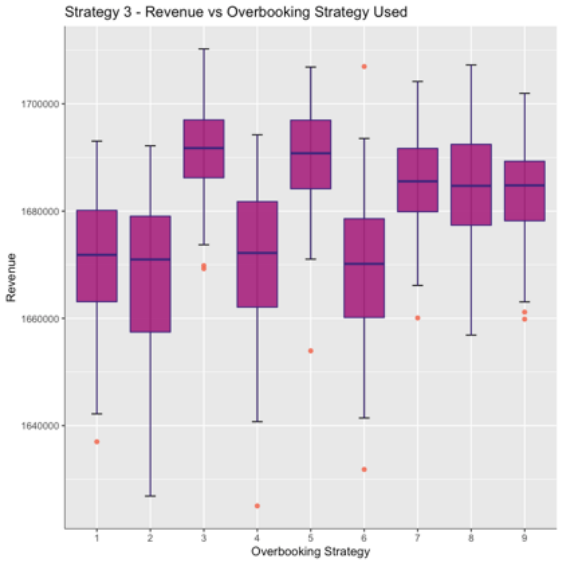
Strategy 2) Overbooking a Constant Number Until Mid-Day

The next overbooking strategy I tested was only overbooking for the morning flights, or flights before noon. In theory this could give the queues that built up enough time to diminish. I tested this strategy for 100 simulations of each overbooking number again from 0 to 10. You can see that there is a definite improvement from the first strategy and although 0 seats still comes out on top in terms of revenue. Other smaller overbooking seat numbers have shown an increase in revenue as well.



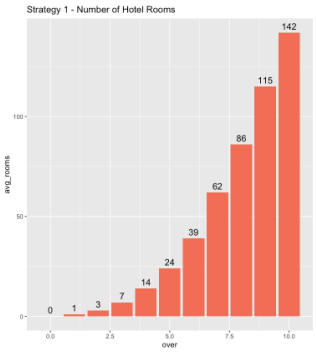
Strategy 3) Various Decreasing Number Strategies

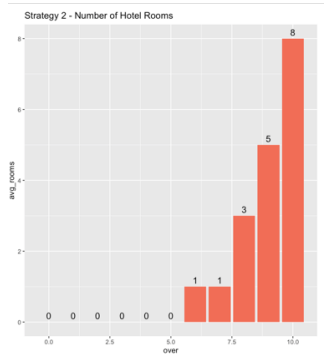
The final strategy I tested, which I imagine is closer to how the airlines actually try to overbook, is a decreasing number strategy. Where morning flights have a higher number of overbooked seats and the number of seats that get overbooked decreases as the day goes on. As I had already split the day into four intervals, I used these intervals again and tested out various random strategies to try to find the optimal one.



Number of Hotel Rooms and Revenue

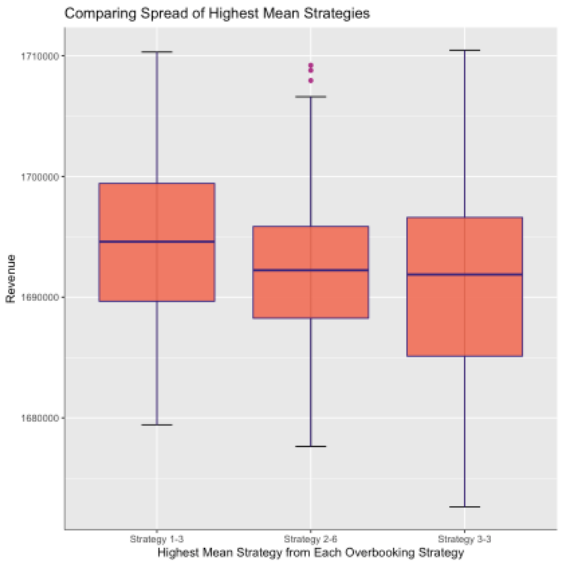
One thing that surprised me in this simulation is that while the increased overbooking seat number had a very strong relationship with the number of hotel rooms being issued, the relationship wasn't quite as clear on the overall revenue. For instance an increase in the overbooking seat number always resulted in an increase in the number of hotel rooms, however the highest mean revenue actually came from an overbooking seat number that didn't have the lowest number of hotel rooms. This shows that it is indeed more profitable for them to issue a small number of hotel rooms, over none at all. However before making any decisions, customer satisfaction is another factor that may want to be considered here. Below are the charts for the average number of hotel room issued in each simulation for Strategy 1 (constant overbooking number) and Strategy 2 (overbooking a constant number until mid-day). For these 100 simulations having 7 hotel rooms actually had a higher mean revenue than having none.





Output Analysis: Comparing Revenue Means and Confidence Intervals

From looking at the box and whisker charts it is easy to see which overbooking numbers generally outperform others for different overbooking strategies. It isn't quite as clear however which strategy produces the highest revenue on average, as the box and whiskers only show the medians, quantiles, and spread. It may make sense in this context to compare their means. The trends so far have shown that overbooking slightly over 0 seats has produced the highest revenue for both overbooking all day and until mid-day. I wanted to compare the means of the best performing overbooking numbers from each strategy. To compare the means I used a Welch's t-test between all strategies [16]. This test assumes normality, which is fine as I ran 100 simulations for each strategy and the sample size is large enough according to the Central Limit Theorem. This test is also appropriate when the variance is unknown for both populations. After looking a the results I convinced myself that the first strategy would have the highest mean revenue when overbooking 3 seats for every flight all day, except on the last flight to each destination for the day. Finally I computed a mean of 1694629 with a 95% confidence interval for this revenue mean to be [1693394 - 1695866]. Below is the spread for each of the highest mean overbooking numbers per strategy.



Conclusion

Using simulation to capture the variance was beyond interesting. It is easy to see how this strategy can reveal insights that may be hidden from other analytical approaches. The one main ingredient that I felt was missing from this project is real historical seat data per flight. I feel that having this sort of data would obviously give more accurate results. Also perhaps looking into flights for the week rather than looking at only one day. There are also many other statistical measures that can be used to compare means and even medians to gain insights, but for this project I decided to keep it simple. The next obvious step for this project will be to look at the lengths of the queue throughout the day. As a decision maker not only will the overall total revenue be an important factor but customer satisfaction as well. And like I said, nobody likes to be stuck in an airport all day.

References

[1] Bailey, J. (2007, May 30). Bumped Fliers and No Plan B. The New York Times. The New York Times. https://www.nytimes.com/2007/05/30/business/30bump.html?pagewanted=all&_r=0. Accessed 27 April 2021.

[2] Bureau of Transportation Statistics. Airline On-Time Statistics and Delay Causes. https://www.transtats.bts.gov/OT_Delay/OT_DelayCause1.asp?20=E.Accessed1May2021.

[3] D.Goldsman, P. Goldsman, A First Course in Probability and Statistics, David Goldsman and Paul Goldsman, December 18, 2020

[4] FlightExplorer.com. Airline Flight Tracking - Flight Status Information. http://travel.flightexplorer.com/trackFlight.aspx.Accessed3May2021.

[5] Hawaiian Airlines. Our Fleet. https://www.hawaiianairlines.com/our-services/at-the-airport/our-fleet. Accessed 22 April 2021.

[6]Informs. Overcoming the Challenges of Aircraft Engine Maintenance and Repair. INFORMS. https://www.informs.org/Impact/O.R.-Analytics-Success-Stories/Overcoming-the-Challenges-of-Aircraft-Engine-Maintenance-and-Repair. Accessed 27 April 2021.

[7] Khare, M., Huynh, M., Sturluson, A., Simon, C. Monte Carlo simulation of airline overbooking. The Simon Ensemble. https://simonensemble.github.io/2018-07/airline-overbooking.html. Accessed 21 April 2021.

[8] Krishna, Radha. (2015, April 20). Simulation of Non-Homogenous Poisson Process. https://radhakrishna.typepad.com/simulating-nonhomogeneous-poisson-process.pdf. Accessed 27 April 2021.

[9] RDocumentation. distr-package: distr – Object Oriented Implementation of Distributions. https://www.rdocumentation.org/packages/distr/versions/2.8.0/topics/distr-package. Accessed 22 April 2021.

[10] RDocumentation. vsetdiff: Find all elements in first argument which are not in second argument. https://www.rdocumentation.org/packages/vecsets/versions/1.1/topics/vsetdiff. Accessed 24 April 2021.

[11] Wikipedia contributors. (2021, May 2). FIFO (computing and electronics). Wikipedia, The Free Encyclopedia.. https://en.wikipedia.org/wiki/FIFO_(computing_and_electronics). Accessed 2 May 2021.

[12] Wikipedia contributors. (2021, January 23). Mixture model. Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/wiki/Mixture_model. Accessed 30 April 2021.

[13] Wikipedia contributors. (2021, March 30). Monte Carlo method. Wikipedia, The Free Encyclopedia.https://en.wikipedia.org/wiki/Monte_Carlo_method. Accessed 23 April 2021.

[14] Wikipedia contributors. (2021, April 3). Poisson point process. Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/wiki/Poisson_point_process. Accessed 27 April 2021.

[15] Wikipedia contributors. (2021, April 29). Rejection sampling. Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/wiki/Rejection_sampling. Accessed 30 April 2021.

[16] Wikipedia contributors. (2021, April 22). Welch's t-test. Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/wiki/Welch\%27s_t-test. Accessed 2 May 2021.

[17] Wilson, B. (Updated 2019, September 26). The Truth About Being Bumped off an Airline Flight. TripSavvy. https://www.tripsavvy.com/being-bumped-off-an-airline-flight-54444. Accessed 24 April 2021.

Appendix

Empty Seat Rates

```
# empty seat rates

main_empty_rate <- function(t){(0.00162*(-0.4*t-(-4.9))^6)/100}
exc_empty_rate <- function(t){0.0005*exp(1.6*(t-(4.05)^2))/100}
```

Fill Seats

```
fill_seats <- function(max_seats, t, empty_rate){

    am = c(0.30, 0.26, 0.20, 0.12, 0.08, 0.02, 0.015, 0.005, 0.0000000001, 0.0000000001)
    mid = c(0.15, 0.28, 0.25, 0.13, 0.1, 0.05, 0.02, 0.019, 0.00075, 0.00025)
    pm = c(0.25, 0.25, 0.22, 0.11, 0.09, 0.05, 0.02, 0.009, 0.00075, 0.00025)
    eve = c(0.28, 0.26, 0.19, 0.14, 0.08, 0.03, 0.015, 0.005, 0.0000000001, 0.0000000001)

    if (t > 19){time <- eve} else if (t > 15){time <- pm} else if (t > 10){time <- mid} else {time <- am}

    # instantiate values and vectors
    group_total = 0
    groups <- c()

    # random number of empty unsold seats
    to_empty <- rbinom(1, max_seats, empty_rate)

    # run until we have at least filled all available seats - unsold
    while(group_total <= max_seats - to_empty){

        # generate one random group number at a time for approriate time interval
        group <- sample(1, x = 1:10, replace = TRUE, prob = time)
        # add number in the group to total amount and add group to groups vector
        group_total = group_total + group
        groups <- c(groups, group)
        }

    # ensure the flight is full and not overbooked, remove overbooked by index
    rm_idx <- which.max(groups == (group_total - max_seats))
    groups <- groups[-rm_idx]

    return (groups)

}
```

Cancel Groups Conditional Probabilities

```
vals <- c(seq(1, 6))
main_cond_probs <- c(0.8713, 0.9092, 0.9452, 0.9728, 0.9891, 0.9999)
exc_cond_probs <- c(0.9294, 0.9301, 0.9356, 0.9452, 0.9848, 0.9999)
```

```
# groups cancel

groups_cancel <- function(groups, vals, cond_probs){

    showed_up <- c()

    for (g in groups){

    U1 <- runif(1, 0, 1)

        if (g == vals[1]){
            if(U1 <= cond_probs[1]){
                showed_up <- c(showed_up, g)
            }
        }
        if (g == vals[2]){
            if(U1 <= cond_probs[2]){
                showed_up <- c(showed_up, g)
            }
        }
        if (g == vals[3]){
            if(U1 <= cond_probs[3]){
                showed_up <- c(showed_up, g)
            }
        }
        if (g == vals[4]){
            if(U1 <= cond_probs[4]){
                showed_up <- c(showed_up, g)
            }
        }
        if (g == vals[5]){
            if(U1 <= cond_probs[5]){
                showed_up <- c(showed_up, g)
            }
        }
        if (g >= vals[6]){
            if(U1 <= cond_probs[6]){
                showed_up <- c(showed_up, g)
            }
        }
    }
    return (showed_up)
}
```

Monte Carlo Groups Cancel

```
max_seats_main <- function(plane){

    if(plane == "A332"){
        return (192)
    }
    if(plane == "A21N"){
        return (129)
    }
    if(plane == "B712"){
        return (120)
    }
}
```

Airline Fleet Main Cabin Seats

```
main_rev <- function(main_booked, plane){

    total_main_rev <- 0.0

    if (plane == "A332"){
        for (group in main_booked){
            group_rev <- round(rnorm(1, mean=240, sd=40),2) * group
            total_main_rev = total_main_rev + group_rev
          }
        }
    if (plane == "A21N"){
        for (group in main_booked){
            group_rev <- round(rnorm(1, mean=180, sd=30),2) * group
            total_main_rev = total_main_rev + group_rev
          }
        }
    if (plane == "B712"){
        for (group in main_booked){
            group_rev <- round(rnorm(1, mean=120, sd=20),2) * group
            total_main_rev = total_main_rev + group_rev
          }
        }
    return (total_main_rev)
}
```

Revenue for Main Cabin Seats

```
nonhomogeneous_poisson <- function(lambda, lambda_star, num, stop_time){

    arrivals <- c()

    for (n in 1:num){

    T <- c()
    i <- 1
    T[i] <- 0
    t <- T[i]

    while (t <= stop_time){

        U <- runif(1)
        # potential arrival at rate lambda star (max)
        t <- t - (1/lambda_star) * log(U)

        V <- runif(1)
        # current rate at time t
        curr_rate <- lambda(t)
        # the thinning probability
        thinning_prob <- (curr_rate / lambda_star)

        if (V <= thinning_prob){
            i <- i + 1
            T[i] <- t
         }
    }
    arrivals[n] <- sum(T <= 2 ) - 1
    }
    return (arrivals)
}
```

Non-Homogeneous Poisson Process

```
delay_mixture <- function(){

    on_time_prob <- 0.8911
    carrier_prob <- 0.0608
    weather_prob <- 0.0036
    nas_prob <- 0.0024
    security_prob <- 0.0006
    late_arr_prob <- 0.0327
    canceled_prob <- 0.0078
    diverted_prob <- 0.001

    mixture <- UnivarMixingDistribution(Norm(mean=-5, sd=2), Norm(mean=15, sd=3),
                                        Norm(mean=10, sd=6), Norm(mean=12, sd=8),
                                        Norm(mean=25, sd=5), Norm(mean=5, sd=1),
                                        Norm(mean=18, sd=2), Norm(mean=45, sd=10),
                                    mixCoeff=c(on_time_prob, carrier_prob, weather_prob, nas_prob,
                                    security_prob, late_arr_prob, canceled_prob, diverted_prob))

    return (mixture)
    }
```

Delay Mixture

```
simple_overbooking <- function(last, main_unfilled, overbooked){

    # only overbook if it's not the last flight that day
    if ((last != TRUE) & (main_unfilled == 0)){
        num_overbooked = overbooked
        }
    # keep track of overnight
    else{
        num_overbooked = 0

        }
    return (num_overbooked)
    }
```

```
midday_overbooking <- function(last, main_unfilled, hour, overbooked, cuttoff){

    # only overbook if it's not the last flight that day
    if ((last != TRUE) & (main_unfilled == 0) & (hour < cutoff)){
        num_overbooked = overbooked
        }
    # keep track of overnight
    else{
        num_overbooked = 0

        }
    return (num_overbooked)
    }
```

```
decreasing_overbooking <- function(last, main_unfilled, hour, overbooking_strategy){

    # only overbook if it's not the last flight that day
    if ((last != TRUE) & (main_unfilled == 0)){

        am_num <- overbooking_strategy[1]
        mid_num <- overbooking_strategy[1]
        pm_num <- overbooking_strategy[1]
        eve_num <- overbooking_strategy[1]

        if (hour > 19){num_overbooked = eve_num}
        else if (hour > 15){num_overbooked = pm_num}
        else if (hour > 10){num_overbooked = mid_num}
        else {num_overbooked = am_num}

        }

    else{
        num_overbooked = 0

        }
    return (num_overbooked)
```

3 Overbooking Strategies

```
board <- function(main_to_board, queue, fed_marshall, main_seats){

    boarding <- c(fed_marshall, queue)
    count <- sum(boarding)

        for(i in 1:length(main_to_board)){
            boarding <- c(boarding, main_to_board[i])
            count = count + main_to_board[i]
            if (count > main_seats){break}
            }

    rm_idx <- which.max(boarding == (sum(boarding) - main_seats))
    boarding <- boarding[-rm_idx]
    boarding <- boarding[boarding != 0]

    return (boarding)

}
```

Make Queue

```
make_queue <- function(df, dept, dest, hour, last, queue_groups){

    if (last != TRUE){

        if ((length(queue_groups)!= 0)){

    queue_groups <- list(queue_groups)

    next_flight <- head(df[which.max((df['DEPT'] == dept) & ( df['DEST'] == dest) & (df['HOUR'] > hour)), ], 1)

    next_flight_num <- unlist(next_flight['FLIGHT_NUM'])

    df$QUEUE[df['FLIGHT_NUM'] == next_flight_num] <-  queue_groups

        }
    }

    return (df)
}
```

Update

```
upgrade <- function(num_unfilled, queue){

    upgraded <- c()
    num_queue <- sum(queue)
    flag = TRUE

    if ((num_unfilled > 0) & (num_queue > 0)){

        while(flag == TRUE){

        big <- queue[which.max(queue)]
        small <- queue[which.min(queue)]

            if(num_unfilled %in% queue){
                upgraded <- c(upgraded, num_unfilled)
                queue <- queue[-which(queue == num_unfilled)[1]]
                num_unfilled <- 0
                }
            else if((big <= num_unfilled) && (length(big) > 0)){
                upgraded <- c(upgraded, big)
                num_unfilled <- num_unfilled - big
                queue <- queue[-which.max(queue)]
                }
            else if((small <= num_unfilled) && (length(small) > 0)){
                upgraded <- c(upgraded, small)
                num_unfilled <- num_unfilled - small
                queue <- queue[-which.min(queue)]
                }
            else{flag <- FALSE
                }
            }
    }
    else{return (0)
        }
    return (upgraded)
}
```

Voucher Cost

```
voucher_cost <- function(overflow, plane, last){

    cost <- c(0)
    if (last != TRUE){
        for (over_group in overflow){
            c <- sample(1, x = c(main_rev(over_group, plane), main_rev(over_group, plane) * (2.0)), replace = TRUE, prob = c(0.95,0.05))
            cost <- c(cost, c)
            }
        }
    else {
        for (over_group in overflow){
            c <- main_rev(over_group, plane) * (4.0)
            cost <- c(cost, c)
        }
    }

    return (sum(cost))
}
```

Number of Hotel Rooms

```
hotel_rooms <- function(last, q){

    num_rooms <- 0

    if (last == TRUE){

        hotel_groups <- q

        for (g in hotel_groups){

            if((is.numeric(g)) & (length(g)!= 0)){

            if ((g > 0) & (g <= 2)){
                rooms <- sample(1:2, 1)
                num_rooms <- num_rooms + rooms
            }
            else if ((g > 2) & (g <= 4)){
                rooms <- sample(1:4, 1)
                num_rooms <- num_rooms + rooms
            }
            else if ((g > 4) & (g <= 6)){
                rooms <- sample(1:3, 1)
                num_rooms <- num_rooms + rooms
            }
            else if ((g > 6) & (g <= 8)){
                rooms <- sample(1:4, 1)
                num_rooms <- num_rooms + rooms
            }
            else if ((g > 8)){
                rooms <- sample(1:10, 1)
                num_rooms <- num_rooms + rooms
            }
            }
            }

        }
    else{num_rooms <- 0}

    return (num_rooms)


    }
```

Update

```
update <- function(df){

    df$M_SEATS[df['FLIGHT_NUM']==flight_num] = main_seats
    df$OVERBOOKED[df['FLIGHT_NUM']==flight_num] = num_overbooked
    df$BOOKED[df['FLIGHT_NUM']==flight_num] = sum(main_booked)
    df$SHOWED[df['FLIGHT_NUM']==flight_num] = sum(main_showed)
    df$M_UNF[df['FLIGHT_NUM']==flight_num] = main_unfilled_preboard
    df$E_UNF[df['FLIGHT_NUM']==flight_num] = exc_unfilled_preboard
    df$UPGRADED[df['FLIGHT_NUM']==flight_num] = sum(upgraded_main)
    df$MAIN_NO_UP[df['FLIGHT_NUM']==flight_num] = sum(main_no_upgrade)
    df$OVERFLOW[df['FLIGHT_NUM']==flight_num] = list(c(overflow))
    df$BOARD[df['FLIGHT_NUM']==flight_num] = sum(boarding)
    df$DELAY[df['FLIGHT_NUM']==flight_num] = round(delay, 2)
    df$TAKEOFF[df['FLIGHT_NUM']==flight_num] = round(hour, 2)

    return (df)

}
```