Final Report

Component I: Supply Chain Analytics.

The goal of this analysis was to find the optimal number of tickets to book for the flight from San Diego to Boston that will result in the highest profit. Given the data of 500 flights with overbooked scenarios of 5, 15, 25, 35, and 45 seats, we first wanted to examine which scenario historically was the most profitable. For each overbooking level, we created a data table that included the number of no shows, their observed probabilities, the number of passengers who showed up, those who were boarded, those who got bumped, the average cost per bumped passenger, fixed operating cost, and the actual profit. Given that compensation wasn't constant, taking the average compensation for each no show seemed the appropriate way to calculate average profit for each level of overbookings. No shows, their probabilities, and average compensation were calculated using pivot tables from the provided data. The fixed cost was constant at $70,000, and the plane's capacity was 250 seats. To determine the average profit for each overbooking level, we first calculated the profit for every possible no show scenario within that level using the following formula:

$600*#Show up - #Bumped * Average compensation per passenger - #Bumped * $600 - $70,000

Since the airline refunds tickets to no show passengers, revenue was calculated based only on those who actually showed up, not on the total tickets purchased. Additionally, for each bumped passenger, Southwest not only had to provide compensation but also cover the cost of an alternative flight, which wiped out the $600 profit per seat. As a result, we subtracted $600 for each bumped passenger, along with the compensation.

After calculating the profit for each no-show scenario within a given overbooking level, we used Excel's "SUMPRODUCT" function to compute the average profit by weighting each outcome by its probability. As a result, booking 265 seats turned out to be the most profitable, so we focused on the range of 256-274 to find the actual optimal booking scenario.

For the second half of the analysis, we used a simulation. To prepare, we first used our existing data table, filtered out all cases where no bumping occurred, leaving only the cases with compensation. This allowed us to examine the distribution of compensation per passenger for each bumped case. We observed that as the number of bumps increased, the average compensation per passenger also grew. To take this into account in our simulation, we stored the compensation values for each bump level in a dictionary of lists. Our maximum value of bumped passengers was 24, since we are looking at the data until 274.

Furthermore, to determine the probability of no shows, we fitted a Multinomial Logit Model (MNL), which is a type of regression model, on our data. It used the historical data and created individual probabilities for each no show for every booking level. Although the no show values in our dataset range from 4 to 27, the MNL model automatically remaps them to a range of 0 to 21, such that 4 becomes 0, 5 becomes 1, etc. Since there are 22 distinct no-show values, they are represented as 0 through 21. Thus, to make sure our calculations are correct, we remapped the numbers back to the original. When examining the predicted probabilities, we observed that as the number of bookings increases, the probability of lower no shows decreases, while the probability of higher no shows increases, which aligns with the case given.

In the final part of our simulation, we calculated the average profit for each booking level (256–274) by combining the predicted no show probabilities from the regression model with the compensation values drawn from the previously collected distribution. The average profit calculations follow the same structure as in Excel, but with newly incorporated simulated data.

Limitations:

It's important to note that since the MNL model was trained on the entire dataset, it assigns a probability to all observed no shows, including 27 for every booking level, even if some of those outcomes are unlikely in certain cases (ex: 256). The model is able to introduce heterogeneity by adjusting the probabilities across different no shows for each booking level, even though the set of possible no shows stays the same.

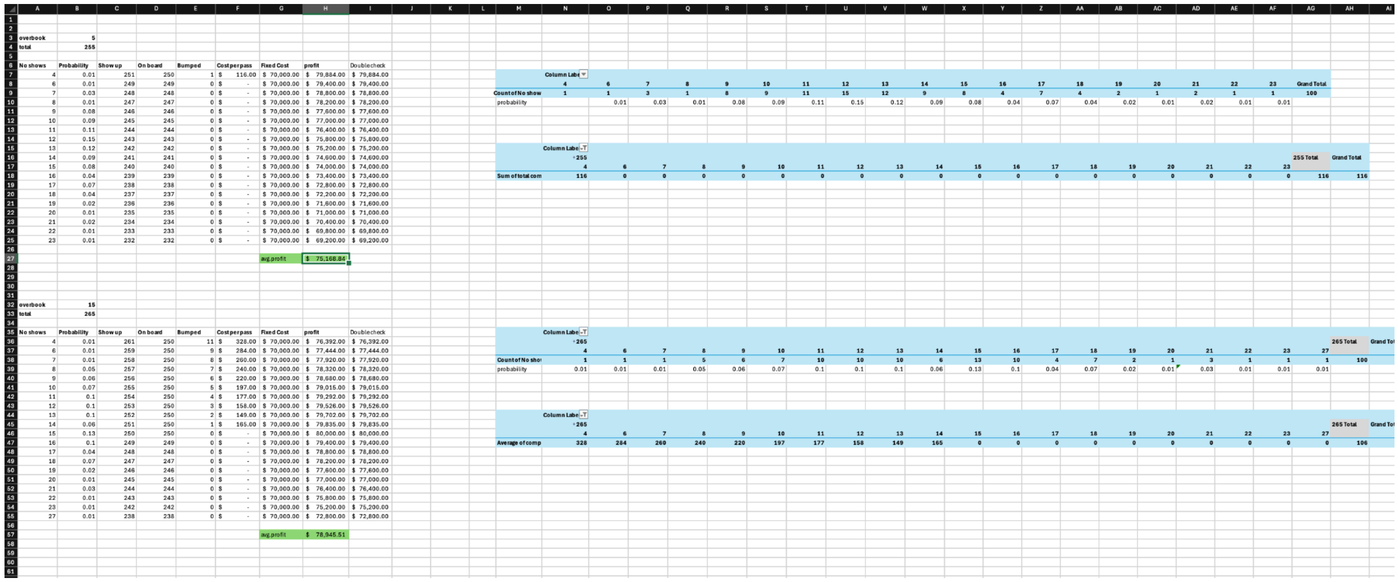
Results:

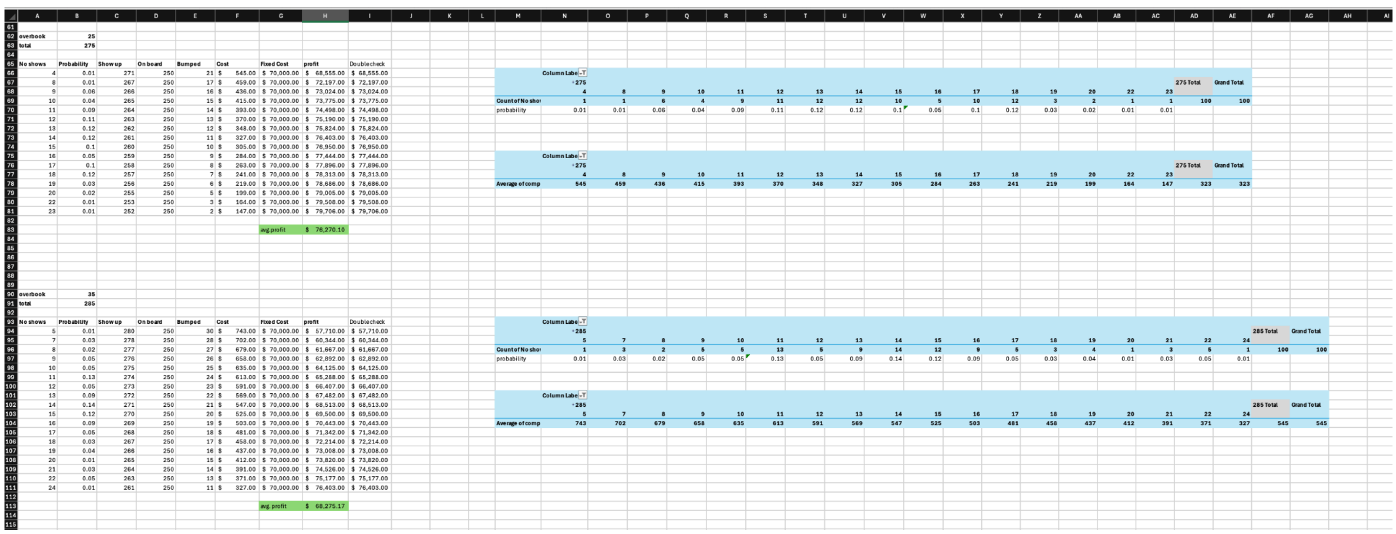
Given our analysis of the data provided, 265 was the optimal number of tickets to book for the flight from San Diego to Boston with an average profit of $79,015.32.

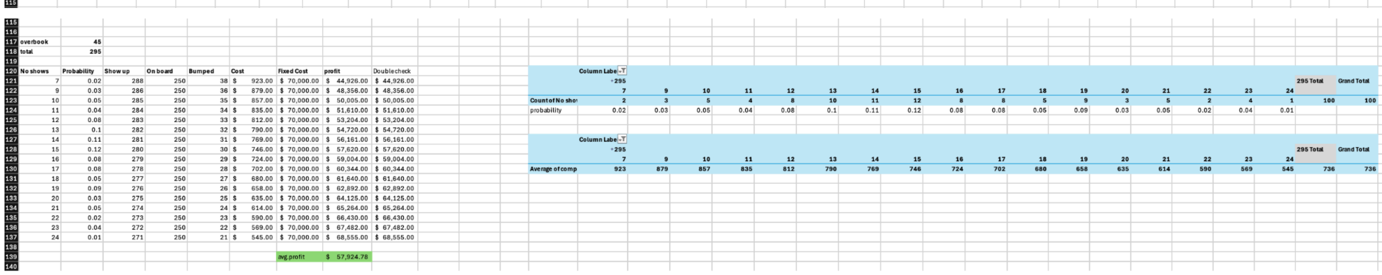
Appendix A - Component I: Supply Chain Analytics

<u>Excel Part:</u>

Calculation of average profit for each booking level. The profit formula is explained in the main write-up.







Simulation Part:

Import the Excel file and keep only the flights with bumped passengers to examine the compensation distribution

```python
import pandas as pd
import numpy as np
import random
import seaborn as sns
import matplotlib.pyplot as plt
import statsmodels.api as sm

# Load the Excel file
df = pd.read_excel("/Users/aruzhansatybay/Downloads/final project overbooking data.xlsx", engine='openpyxl', skiprows=1)

# Show first few rows
df.head()
```

[130] Python

	flight	Booked	No shows	Unnamed: 3	bumped	total compensation	Unnamed: 6	compensation per bumped passenger
0	1	255	12	NaN	0	0	NaN	0.0
1	2	255	17	NaN	0	0	NaN	0.0
2	3	255	12	NaN	0	0	NaN	0.0
3	4	255	17	NaN	0	0	NaN	0.0
4	5	255	17	NaN	0	0	NaN	0.0

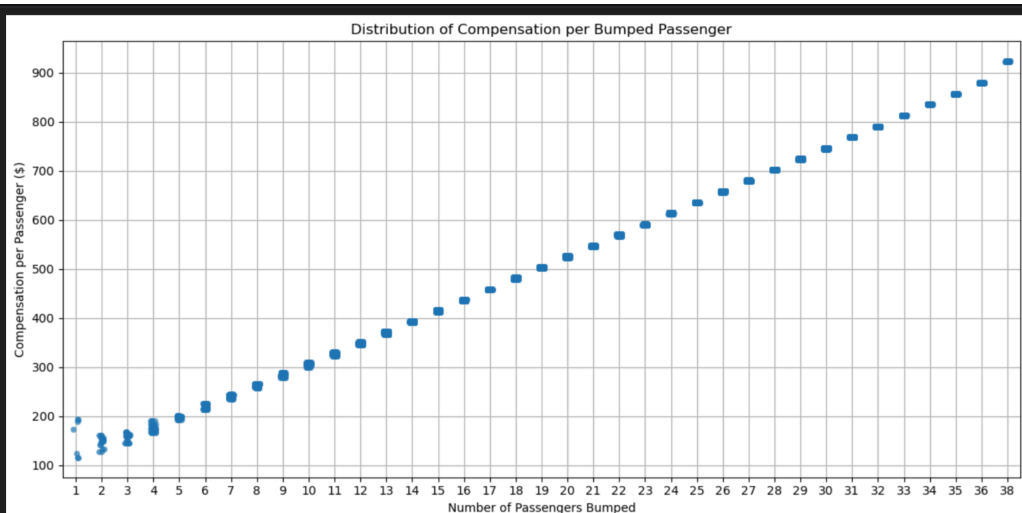
```python
df_bumped = df[df['bumped'] > 0].copy()
df_bumped.head()
```

[131] Python

	flight	Booked	No shows	Unnamed: 3	bumped	total compensation	Unnamed: 6	compensation per bumped passenger
11	12	255	4	NaN	1	116	NaN	116.000000
100	101	265	11	NaN	4	725	NaN	181.250000
101	102	265	6	NaN	9	2554	NaN	283.777778
103	104	265	11	NaN	4	711	NaN	177.750000
108	109	265	12	NaN	3	484	NaN	161.333333

```python
# Repeat each row 'bumped' times. One for each bumped passenger
expanded = df_bumped.loc[df_bumped.index.repeat(df_bumped['bumped'])].copy()

# Reset index and optionally track each passenger within a flight
expanded.reset_index(drop=True, inplace=True)
expanded['passenger_number'] = expanded.groupby('flight').cumcount() + 1

plt.figure(figsize=(12, 6))

sns.stripplot(
    data=expanded,
    x='bumped',  # x-axis: number of passengers bumped
    y='compensation per bumped passenger',  # y-axis: compensation amount
    jitter=True,
    alpha=0.7
)

plt.title("Distribution of Compensation per Bumped Passenger")
plt.xlabel("Number of Passengers Bumped")
plt.ylabel("Compensation per Passenger ($)")
plt.grid(True)
plt.tight_layout()
plt.show()
```

[132] Python



Distribution of compensation stored in the dictionary of lists for later access in the simulation

```python
# max amountof bumped can be 24 for the range 256-274
options_by_var = {
    1: [116, 124, 173, 190, 192, 194],
    2: [127.5, 133, 142.5, 147, 148, 151, 154, 155.5,156.5,160.5,161],
    3: [146, 150, 159.7, 161, 161.3, 163.7, 165.7, 167.3],
    4: [167, 167.25, 173, 173.5,174.8, 176.8, 177.75, 181.3, 184, 191.5],
    5: [192.4, 195, 196.6, 197.2, 197.6, 199, 200, 200.4, 201],
    6: [213.2, 214, 216.2, 216.8, 219.3, 222.5, 224.3, 225.8, 226.3],
    7: [234, 236.6, 236.9, 237, 238.1, 238.3, 239.3, 240.9, 241.6, 241.7, 241.9, 243.3, 244, 244.1, 245.9],
    8: [258, 258.1, 260, 260.4, 261.4,  264, 264.5, 265.8,  266.5,  266.6,  267],
    9: [277.8,  282.4,  283.8,  285, 288.4, 288.7],
    10: [300.4, 300.7, 303.4, 304.6, 304.7, 305.8, 306.5, 308, 308.9, 309.5],
    11: [322.9, 323.5, 323.9, 325.5, 326.9, 327.1, 327.7, 328.3, 328.4, 328.9, 329.1, 331,  331.2],
    12: [344.5, 345.9, 346.6, 346.7, 347.8, 348.1, 348.6, 348.9, 349.8, 351.7, 352.5],
    13: [367, 367.1, 367.5, 368.1, 368.5, 368.6, 368.9, 369.7, 371.2, 372.3, 372.4, 372.5, 373.3, 373.4],
    14: [390.1, 390.5, 390.7, 390.9, 391.3, 391.6, 392.9, 393.6, 393.6, 393.8, 394.2, 394.4],
    15: [411.9, 412.1, 414.2, 416.4, 417.5],
    16: [434.1, 434.4, 434.7, 435.6, 436.6, 436.8, 437.3, 437.4, 438.3, 438.4],
    17: [458.1, 458.3, 458.6, 459.2],
    18: [478.2, 479,    479.2,  483.1,  483.2],
    19: [500.4, 500.9, 501.4, 501.9, 502.4, 504.2, 504.5, 504.7, 504.8],
    20: [523.2, 523.3,  524.1,  524.6,  525.3,  525.5,  525.9,  526, 526.4, 526.7,  526.9,  527],
    21: [544.5, 544.7, 545.1, 545.7, 545.9, 546.3, 546.7, 546.8, 547.6, 548.3, 548.8],
    22: [566.8, 567.9, 568.5, 568.6, 569, 569.2, 569.7, 570, 570.2, 570.3, 570.8, 570.9],
    23: [589.1, 589.6, 590.3, 591,  591.1,  591.4, 592.9],
    24: [611.4, 611.6, 612.1, 612.7, 613, 613.2, 613.5, 614, 614.1, 614.2,  614.3,  614.6, 614.8]
}
```

MNL Regression Model to predict probabilities from the original data:

```python
seats = 250
revenue_per_passenger = 600
fixed_cost = 70000


# Group the data to count how many times each (Booked, No shows) combination appears
grouped = df.groupby(['Booked', 'No shows']).size().reset_index(name='Count')


# Expand rows based on count for MNL fitting
expanded = grouped.loc[grouped.index.repeat(grouped['Count'])].reset_index(drop=True)


# Feature: Booked tickets
X = sm.add_constant(expanded[['Booked']])
y = expanded['No shows']


# Fit the Multinomial Logit Model
model = sm.MNLogit(y, X)
result = model.fit()


# --- Map internal labels back to actual no-show values ---
internal_labels = np.unique(result.model.endog)
original_labels = sorted(y.unique())
reverse_mapping = dict(zip(internal_labels, original_labels))

# Predict no-show distributions for booking levels
booking_range = range(256, 275)
predicted_distributions = {}


for booked in booking_range:
    X_new = pd.DataFrame({'const': [1], 'Booked': [booked]})
    probs = result.predict(X_new).values[0]

   # Map internal label → actual no-show value
    mapped_probs = {
        reverse_mapping[i]: p for i, p in enumerate(probs)
    }
    predicted_distributions[booked] = mapped_probs
```

Calculation of profit similar to Excel but now with simulated probabilities and compensations:

```python
for booked in booking_range:
    no_show_probs = predicted_distributions[booked]
    profits = []
    for actual_no_shows, prob in no_show_probs.items():
        show_ups = booked - actual_no_shows
        if show_ups <= seats:
            revenue = show_ups * revenue_per_passenger
            cost = fixed_cost
        else:
            flown = seats
            bumped = show_ups - seats
            revenue = flown * revenue_per_passenger
            if bumped in options_by_var:
                comp_options = options_by_var[bumped]
                compensation_per_passanger = np.random.choice(comp_options)
                total_compensation = bumped * compensation_per_passanger
            else:
                total_compensation = 0
            cost = fixed_cost + total_compensation

        profit = revenue - cost
        weighted_profit = prob * profit
        profits.append(weighted_profit)

    expected_profit = sum(profits)
    print(f"Booked: {booked}, Expected Profit: ${expected_profit:,.2f}")

   #If want to see simulated probabilites for each booking scenario
    #print(predicted_distributions[booked])

[134]                                                                   Python
```

The results show 265 is the most profitable booking scenario:

```
Optimization terminated successfully.
         Current function value: 2.678698
         Iterations 9
Booked: 256, Expected Profit: $75,708.97
Booked: 257, Expected Profit: $76,262.19
Booked: 258, Expected Profit: $76,804.58
Booked: 259, Expected Profit: $77,328.31
Booked: 260, Expected Profit: $77,794.50
Booked: 261, Expected Profit: $78,216.13
Booked: 262, Expected Profit: $78,550.43
Booked: 263, Expected Profit: $78,787.62
Booked: 264, Expected Profit: $78,956.64
Booked: 265, Expected Profit: $79,024.15
Booked: 266, Expected Profit: $79,014.12
Booked: 267, Expected Profit: $78,941.47
Booked: 268, Expected Profit: $78,770.97
Booked: 269, Expected Profit: $78,559.26
Booked: 270, Expected Profit: $78,270.48
Booked: 271, Expected Profit: $77,937.46
Booked: 272, Expected Profit: $77,561.88
Booked: 273, Expected Profit: $77,123.50
Booked: 274, Expected Profit: $76,654.40
```

To see how an individual booking level gets computed:

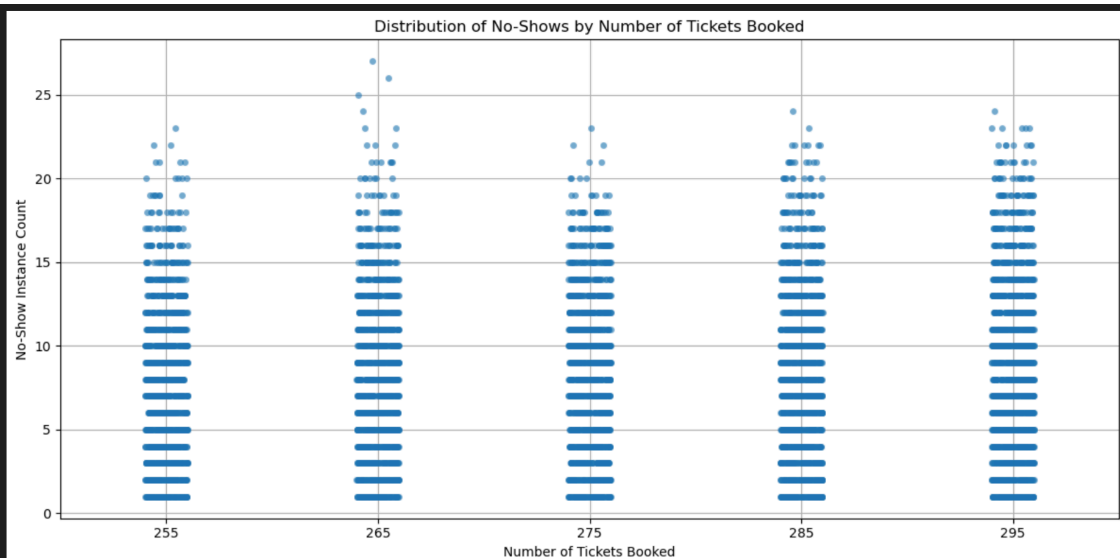
```python
booked = 271
no_show_probs = predicted_distributions[booked]

profits = []  # Must reset this here, not outside multiple runs

print(f"\n--- Detailed Breakdown for Booked = {booked} ---\n")
for actual_no_shows, prob in sorted(no_show_probs.items()):
    show_ups = booked - actual_no_shows
    if show_ups <= seats:
        revenue = show_ups * revenue_per_passenger
        cost = fixed_cost
        total_compensation = 0
    else:
        flown = seats
        bumped = show_ups - seats
        revenue = flown * revenue_per_passenger
        if bumped in options_by_var:
            comp_options = options_by_var[bumped]
            compensation_per_passanger = np.random.choice(comp_options)
            total_compensation = bumped * compensation_per_passanger
        else:
            total_compensation = 0
        cost = fixed_cost + total_compensation

    profit = revenue - cost
    weighted_profit = prob * profit
    profits.append(weighted_profit)

    print(f"No-shows: {actual_no_shows:2d} | Show-ups: {show_ups:3d} | "
          f"Bumped: {max(show_ups - seats, 0):2d} | Prob: {prob:.4f} | "
          f"Profit: ${profit:,.2f} | Weighted: ${weighted_profit:,.2f}")
expected_total_profit = sum(profits)
print(f"\n✅ Expected Total Profit for {booked} Bookings: ${expected_total_profit:,.2f}")
```

```
--- Detailed Breakdown for Booked = 271 ---

No-shows:  4 | Show-ups: 267 | Bumped: 17 | Prob: 0.0057 | Profit: $72,212.30 | Weighted: $414.34
No-shows:  5 | Show-ups: 266 | Bumped: 16 | Prob: 0.0012 | Profit: $73,049.60 | Weighted: $87.80
No-shows:  6 | Show-ups: 265 | Bumped: 15 | Prob: 0.0024 | Profit: $73,787.00 | Weighted: $179.72
No-shows:  7 | Show-ups: 264 | Bumped: 14 | Prob: 0.0186 | Profit: $74,478.40 | Weighted: $1,383.45
No-shows:  8 | Show-ups: 263 | Bumped: 13 | Prob: 0.0192 | Profit: $75,158.80 | Weighted: $1,445.26
No-shows:  9 | Show-ups: 262 | Bumped: 12 | Prob: 0.0602 | Profit: $75,826.40 | Weighted: $4,567.02
No-shows: 10 | Show-ups: 261 | Bumped: 11 | Prob: 0.0645 | Profit: $76,382.10 | Weighted: $4,923.31
No-shows: 11 | Show-ups: 260 | Bumped: 10 | Prob: 0.1004 | Profit: $76,911.00 | Weighted: $7,719.57
No-shows: 12 | Show-ups: 259 | Bumped:  9 | Prob: 0.1054 | Profit: $77,458.40 | Weighted: $8,164.08
No-shows: 13 | Show-ups: 258 | Bumped:  8 | Prob: 0.1113 | Profit: $77,884.00 | Weighted: $8,666.29
No-shows: 14 | Show-ups: 257 | Bumped:  7 | Prob: 0.1009 | Profit: $78,324.90 | Weighted: $7,905.69
No-shows: 15 | Show-ups: 256 | Bumped:  6 | Prob: 0.1101 | Profit: $78,654.20 | Weighted: $8,661.87
No-shows: 16 | Show-ups: 255 | Bumped:  5 | Prob: 0.0707 | Profit: $79,000.00 | Weighted: $5,584.64
No-shows: 17 | Show-ups: 254 | Bumped:  4 | Prob: 0.0688 | Profit: $79,264.00 | Weighted: $5,451.46
No-shows: 18 | Show-ups: 253 | Bumped:  3 | Prob: 0.0648 | Profit: $79,516.10 | Weighted: $5,149.83
No-shows: 19 | Show-ups: 252 | Bumped:  2 | Prob: 0.0288 | Profit: $79,689.00 | Weighted: $2,293.59
No-shows: 20 | Show-ups: 251 | Bumped:  1 | Prob: 0.0139 | Profit: $79,808.00 | Weighted: $1,109.18
No-shows: 21 | Show-ups: 250 | Bumped:  0 | Prob: 0.0230 | Profit: $80,000.00 | Weighted: $1,840.99
No-shows: 22 | Show-ups: 249 | Bumped:  0 | Prob: 0.0165 | Profit: $79,400.00 | Weighted: $1,306.86
No-shows: 23 | Show-ups: 248 | Bumped:  0 | Prob: 0.0107 | Profit: $78,800.00 | Weighted: $844.82
No-shows: 24 | Show-ups: 247 | Bumped:  0 | Prob: 0.0010 | Profit: $78,200.00 | Weighted: $79.31
No-shows: 27 | Show-ups: 244 | Bumped:  0 | Prob: 0.0019 | Profit: $76,400.00 | Weighted: $146.12

✅ Expected Total Profit for 271 Bookings: $77,925.22
```

Visualisation of No-Show Distribution by Booking Level

```python
# Create a new DataFrame where each row represents a single no-show
expanded_no_shows = df.loc[df.index.repeat(df['No shows'])].copy()
expanded_no_shows.reset_index(drop=True, inplace=True)
expanded_no_shows['no_show_number'] = expanded_no_shows.groupby('flight').cumcount() + 1

plt.figure(figsize=(12, 6))

sns.stripplot(
    data=expanded_no_shows,
    x='Booked',  # x-axis: number of tickets booked
    y='no_show_number',  # y-axis: each no-show instance (from 1 to n)
    jitter=True,
    alpha=0.6
)

plt.title("Distribution of No-Shows by Number of Tickets Booked")
plt.xlabel("Number of Tickets Booked")
plt.ylabel("No-Show Instance Count")
plt.grid(True)
plt.tight_layout()
plt.show()
```

[47] ✓ 0.1s Python



Visualisation of Probability Distribution of No_Shows for Every Booking Level

```python
# Group by 'Booked' and 'No shows' to get counts
grouped = df.groupby(['Booked', 'No shows']).size().reset_index(name='count')

# Calculate conditional probabilities: P(NoShows | Booked)
grouped['probability'] = grouped.groupby('Booked')['count'].transform(lambda x: x / x.sum())

# Filter to only certain booking levels (for cleaner plot)
booking_levels = [255]
filtered = grouped[grouped['Booked'].isin(booking_levels)]

# Plot
plt.figure(figsize=(12, 6))

for level in booking_levels:
    subset = filtered[filtered['Booked'] == level]
    plt.plot(subset['No shows'], subset['probability'], marker='o', label=f'Booked = {level}')

plt.title('Probability Distribution of No-Shows by Booking Level')
plt.xlabel('Number of No-Shows')
plt.ylabel('Probability')
plt.legend(title='Booked Tickets')
plt.grid(True)
plt.tight_layout()
plt.show()
```