Trabajo Fin de Grado

Grado en Estadística Empresarial

Gestión de Ingresos.

Autor:
Ernesto Moltó Quiles.

Tutor:
Mercedes Landete Ruiz.
Juan Francisco Monge Ivars.

Facultad de Ciencias Sociales y Jurídicas de Elche
Universidad Miguel Hernández
Curso 2023/2024

Índice

INTRODUCCIÓN

Este trabajo, se centra en la optimización financiera de recursos limitados, que desempeña un papel esencial en diversas industrias. En nuestro trabajo nos enfocaremos en la aviación, donde la disponibilidad de asientos en los vuelos se convierte en un recurso estratégico. A medida que enfrentamos la gestión de ingresos en este entorno, tenemos que coordinar las decisiones para satisfacer la demanda. A esto se le conoce como gestión de ingresos de red.

Para realizar este objetivo, utilizamos la formulación de un modelo matemático que representa la asignación optima de los recursos, en este caso, los asientos de vuelos, para maximizar los ingresos totales. La programación y la optimización matemática se convierten en herramientas fundamentales para resolver este problema.

El problema en específico implica la planificación de un itinerario y la asignación de una posición dentro de los desplazamientos u ocupaciones que realizan este itinerario, en función de la demanda y las capacidades de los desplazamientos. Por eso, nuestra programación en Python aborda este escenario, tomando decisiones estratégicas sobre si aceptamos o no las solicitudes de compra para lograr el máximo rendimiento financiero posible.

Seguidamente a través de Python, también podemos explorar sobre la viabilidad de aumentar 50 asientos en periodos y en máximo 4 vuelos específicos para evaluar el impacto que tiene sobre los ingresos totales. También añadiremos un análisis comparativo, para que podamos ver el comportamiento de los ingresos totales, si aumentamos desde el principio en 50 asientos a los 4 vuelos que esperemos que tengan una mayor demanda, o si no aumentamos los recursos de nuestros vuelos.

A través de esta integración de modelos matemáticos y programación, este trabajo buscará proporcionar soluciones efectivas para la gestión de ingresos en entornos de red. Al mejorar la toma de decisiones en la industria de la aviación, esperamos contribuir de manera significativa al éxito financiero y operativo de las empresas de este sector.

GESTIÓN DE INGRESOS

La gestión de ingresos, son problemas en los que su objetivo es maximizar los ingresos de la venta de cantidades limitadas de un conjunto de recursos mediante decisiones de gestión de la demanda. Los recursos en la gestión de ingresos suelen ser productos o servicios como una habitación de hotel para una fecha determinada. También nos podremos encontrar, en la gestión de ingresos que los recursos se vendan en conjunto. Es decir, las habitaciones en lugar de reservarlas por noche puedes seleccionar un periodo, o en el caso de los vuelos, se vende un

billete, para un destino, aunque no sea un vuelo directo y tengamos que hacer alguna escala. Por lo tanto, las decisiones de gestión de la demanda de estos recursos deben coordinarse, lo que generalmente conocemos como gestión de ingresos de red (Talluri y van Ryzin 2004).
En el enfoque actual utilizamos las políticas de precios de oferta (Talluri y van Ryzin 1998), en las que se genera un precio de oferta para cada paquete y se acepta una solicitud para comprar el paquete si y solo si los ingresos asociados superan el precio de la oferta. Las políticas de precios de oferta no son óptimas en general, pero son muy populares, son intuitivas y fáciles de implementar. Los precios de oferta pueden generarse utilizando enfoques "aditivos" o "no aditivos" (Bertsimas y Popescu 2003).
En este estudio propondremos un modelo de Programación Lineal Determinista (PLD) típico. En estos modelos asumimos que las demandas y otros parámetros son conocidos con certeza y no incluye consideraciones para la incertidumbre. Aunque el modelo también lo podríamos utilizar para escenarios estocásticos o alguna forma de incertidumbre, entonces también podríamos considerarlo como un modelo de Programación Lineal Estocástica (PLE).
Dentro del modelo, utilizaremos variables de decisión, funciones objetivo y restricciones lineales. Además, también mencionaremos el uso de precios duales y la optimización de ingresos, lo cual es común en problemas de gestión de ingresos.

OBJETIVO.

Como bien hemos introducido anteriormente, nuestro objetivo fundamental será satisfacer la demanda del mercado de manera eficiente, generando ingresos máximos para nuestra empresa. Donde la asignación de los asientos se convierte en una pieza clave para alcanzar el rendimiento financiero óptimo, ya que, si vendes un asiento, este estará ocupado y por lo tanto no podrás obtener mayor beneficio.
Para cumplir este objetivo, desarrollaremos 3 distintos bloques, para poder llegar a este objetivo.
El enfoque metodológico, donde crearemos un modelo matemático que refleje con precisión la asignación de asientos en vuelos, optimizando la gestión de ingresos en un entorno de red complejo y utilizando la programación en Python para implementar soluciones prácticas y eficientes basadas en los modelos matemáticos desarrollados, permitiendo una aplicación ágil.
También buscaremos la adaptabilidad y un correcto rendimiento, integrando estrategias de adaptación a la incertidumbre en la demanda, garantizando que el sistema pueda ajustarse a cambios rápidos y proporcionar decisiones sólidas en tiempo real, además de realizar

análisis de sensibilidad y una evaluación continua del rendimiento del modelo, identificando oportunidades de mejora y ajustando estrategias según la evolución del entorno operativo.
Para finalizar buscaremos que tenga un impacto practico, donde indicaremos recomendaciones prácticas y accionables basadas en los resultados obtenidos, ofreciendo directrices concretas para mejorar la toma de decisiones en la gestión de ingresos.
Con estos objetivos, nuestro estudio se posiciona no solo como un ejercicio teórico, sino también como una herramienta estratégica para alcanzar el equilibrio óptimo entre la satisfacción de la demanda del mercado y la maximización de los ingresos de la empresa.

RED.

Nuestro objetivo principal es buscar asignar la máxima cantidad de asientos posibles en los desplazamientos u ocupaciones, para así maximizar nuestro beneficio, teniendo en cuenta la demanda y la capacidad. En este estudio, se analizan dos clases de asientos: busines y estándar. Estas clases presentan diferencias significativas tanto en demanda como en oferta, lo que añade una capa adicional de complejidad y estrategia a nuestra optimización.
A continuación, veremos los distintos aeropuertos de salida y destino, y los vuelos que podemos realizar entre ellos, que son las flechas:

1- Imagen 1, Fuente:Elaboración propia

Por lo tanto, consideraremos 18 itinerarios posibles, los que se muestran a continuación. (18 puntos)

1) Alicante-Santiago de Compostela
2) Alicate- New York
3) Alicante-Punta Arenas
4) Santiago de Compostela-Alicante
5) Santiago de Compostela-New York
6) Santiago de Compostela-Punta Arenas
7) New York- Alicante
8) New York- Santiago de Compostela
9) New York- Punta Arenas
10) Punta Arenas- Alicante.
11) Punta Arenas- Santiago de Compostela.
12) Punta Arenas- New York.
13) Madrid-Miami
14) Madrid-Santiago de Chile
15) Miami-Santiago de Chile
16) Miami-Madrid
17) Santiago de Chile-Madrid
18) Santiago de Chile-Miami

Y, por otro lado, solo podemos realizar un total de 14 vuelos, los vuelos son las flechas que aparecen en nuestra red. (14 puntos)

1) Alicante-Madrid
2) Santiago de Compostela-Madrid
3) New York-Miami
4) Punta Arenas-Santiago de Chile
5) Madrid-Alicante
6) Madrid-Santiago de Compostela
7) Madrid-Miami
8) Madrid-Santiago de Chile
9) Miami-Madrid
10) Miami-New York
11) Miami-Santiago de Chile
12) Santiago de Chile-Punta Arenas
13) Santiago de Chile-Madrid
14) Santiago de Chile-Miami.

MODELO INICIAL.

En este trabajo, nos centraremos en la optimización financiera de recursos limitados. Para iniciar la investigación, he contado con el apoyo y la guía de mis tutores, quienes me han proporcionado un modelo base establecido.

Posteriormente, hemos desarrollado y ampliado este modelo base con el objetivo de mejorar su eficacia y adaptación a las necesidades específicas de nuestro estudio. [2]

Conjuntos.

- L, conjunto de vuelos.
- I, conjunto de itinerarios.
- J, conjunto de clases (business y economic).
- T, conjunto de instantes de tiempo en el horizonte de planificación.
- I_l, conjunto de itinerarios que usa el vuelo $l, l \in L$.

Parámetros.

- f_{ij} , tarifa de la clase j para el itinerario $i, i \in I, j \in J$.
- C_l , capacidad del vuelo $l, l \in L$.
- d_{ij}^t, demanda de la clase j para el itinerario i en el instante $t, i \in I, j \in J, t \in T$.

Variables.

- b_{ij}^t, Indica el número de clientes que aceptaremos en cada una de las clases j en el periodo $t, i \in I, j \in J, t \in T$.
- B_{ij}^t, variable acumulativa de decisión que indica el número acumulado de clientes que aceptamos en la clase clase j hasta periodo $t, i \in I, j \in J$.

Modelo.

$$max \sum_{i \in I} \sum_{j \in J} \sum_{t \in T} f_{ij}^t b_{ij}^t$$

s.t

$$B_{ij}^t = B_{ij}^{t-1} + b_{ij}^t \ , \ \forall t \in T, \forall i \in I, \forall j \in J$$

$$\sum_{i \in I_l} \sum_{j \in J} B_{ij}^t \leq C_l \ , \ \forall t \in T, \forall l \in L$$

$$0 \leq b_{ij}^t \leq d_{ij}^t, \ \forall t \in T, \forall i \in I, \forall j \in J$$

MODELO FINAL.

Este modelo, fruto del modelo base proporcionado por mis tutores, ha dado a lugar a una adaptación significativa en la gestión de ingresos en la industria de aviación.

Como bien hemos mencionado anteriormente, nuestro objetivo es maximizar el beneficio de los itinerarios, teniendo en cuenta la asignación

óptima de clientes a vuelos en diferentes periodos. Para ello hemos tenido en cuenta las siguietes restricciones.

1. El número de clientes aceptados en un vuelo debe ser menor o igual a la demanda correspondiente y superior o igual a zero, ya que no podemos eliminar un vuelo que ya ha sido asignado.
2. Introduciremos una variable acumulativa que suma el valor del periodo anterior para cada itinerario, clase y periodo. La suma de estas variables acumulativas para itinerarios y clases que comparten el mismo vuelo no puede exceder la capacidad del avión asignado al vuelo.
3. Implementamos una restricción para el aumento de la capacidad del vuelo, introduciendo una varaible binaria que indica en qué vuelo se realizará un aumento de capacidad. Esta decisión se toma de manera óptima durante la optimización del modelo.

Conjuntos.

- L, conjunto de vuelos.
- I, conjunto de itinerarios.
- J, conjunto de clases (business y economic).
- T, conjunto de instantes de tiempo en el horizonte de planificación.
- I_l, conjunto de itinerarios que usa el vuelo $l, l \in L$.

Parámetros.

- f_{ij} , tarifa de la clase j para el itinerario $i, i \in I, j \in J$.
- C_l , capacidad del vuelo $l, l \in L$.
- d_{ij}^t, demanda de la clase j para el itinerario i en el instante $t, i \in I, j \in J, t \in T$.

Variables.

- b_{ij}^t, Indica el número de clientes que aceptaremos en cada una de las clases j en el periodo $t, i \in I, j \in J, t \in T$.
- B_{ij}^t, variable acumulativa de decisión que indica el número acumulado de clientes que aceptamos en la clase clase j hasta periodo $t, i \in I, j \in J$.
- y_l variable binaria que indica que en el vuelo l se añaden 50 asientos , $l \in L$.

Modelo.

$$max \sum_{i \in I} \sum_{j \in J} \sum_{t \in T} f_{ij}^t b_{ij}^t$$

s.t

$$B_{ij}^t = B_{ij}^{t-1} + b_{ij}^t \ , \ \forall t \in T, \forall i \in I, \forall j \in J$$

$$\sum_{i \in I_l} \sum_{j \in J} B_{ij}^t \leq C_l\,,\ \forall t \in T, \forall l \in L$$
$$0 \leq b_{ij}^t \leq d_{ij}^t,\ \forall t \in T, \forall i \in I, \forall j \in J$$
$$\sum_{l \in L} y_l\ =\ 4\,, y_l\ \in [0,1]$$

LIBRERIAS UTILIZADAS.

- CPLEX: es una biblioteca de optimización desarrollada por IBM que ofrece un conjunto de herramientas para resolver problemas de optimización. Sus capacidades incluyen técnicas de resolución eficientes, manejo de restricciones complejas y capacidad de resolver problemas de gran escala. En el contexto de nuestro trabajo de gestión de ingresos, la integración de CPLEX nos permite abordar problemas específicos con mayor precisión y eficacia, lo que se traduce en soluciones óptimas y eficientes para asignar recursos limitados, como son los asientos de los vuelos en nuestro trabajo.
- DOCPLEX: es una interfaz de programación de Python que se integra con CPLEX y proporciona una forma intuitiva de formular y resolver modelos de optimización lineal y entera. Permite expresar modelos de optimización de manera clara y concisa mediante la definición de variables, restricciones y funciones objetivo. Con DOCPLEX, podemos traducir nuestros problemas de gestión de ingresos en modelos matemáticos precisos y resolverlos de manera eficiente utilizando las capacidades de CPLEX.
- NumPy: Es una biblioteca fundamental para la computación numérica en Python. Proporciona una amplia gama de funciones y herramientas para trabajar con matrices multidimensionales y realizar operaciones numéricas eficientes. En el contexto de nuestro trabajo, NumPy será útil para la generación de datos numéricos, como la creación de variables aleatorias con valores específicos para la media y la desviación estándar. Esto nos permite simular la demanda y los precios de los vuelos con mayor precisión, lo que contribuye a la robustez y la validez de nuestros análisis.

Con estas bibliotecas, tenemos acceso a herramientas poderosas y eficientes que nos permiten abordar los desafíos de optimización en nuestro trabajo de gestión de ingresos de manera efectiva y precisa.

PROGRAMACIÓN EN PYTHON.

Para realizar está programación hemos añadido los itinerarios, los vuelos y los periodos que tendrán nuestros clientes para comprar sus vuelos. Tambien hemos estimado una demanda y un beneficio, ya que no tenemos la información real para estos vuelos. Está demanda será simulada, a partir de una distribución normal para modelar la demanda esperada, con parámetros especificos según el tipo de itinerario y clase. De manera similar, generamos precios de venta aleatorios con una distribución normal, utilizando una media y una desviación estándar predefinidas.

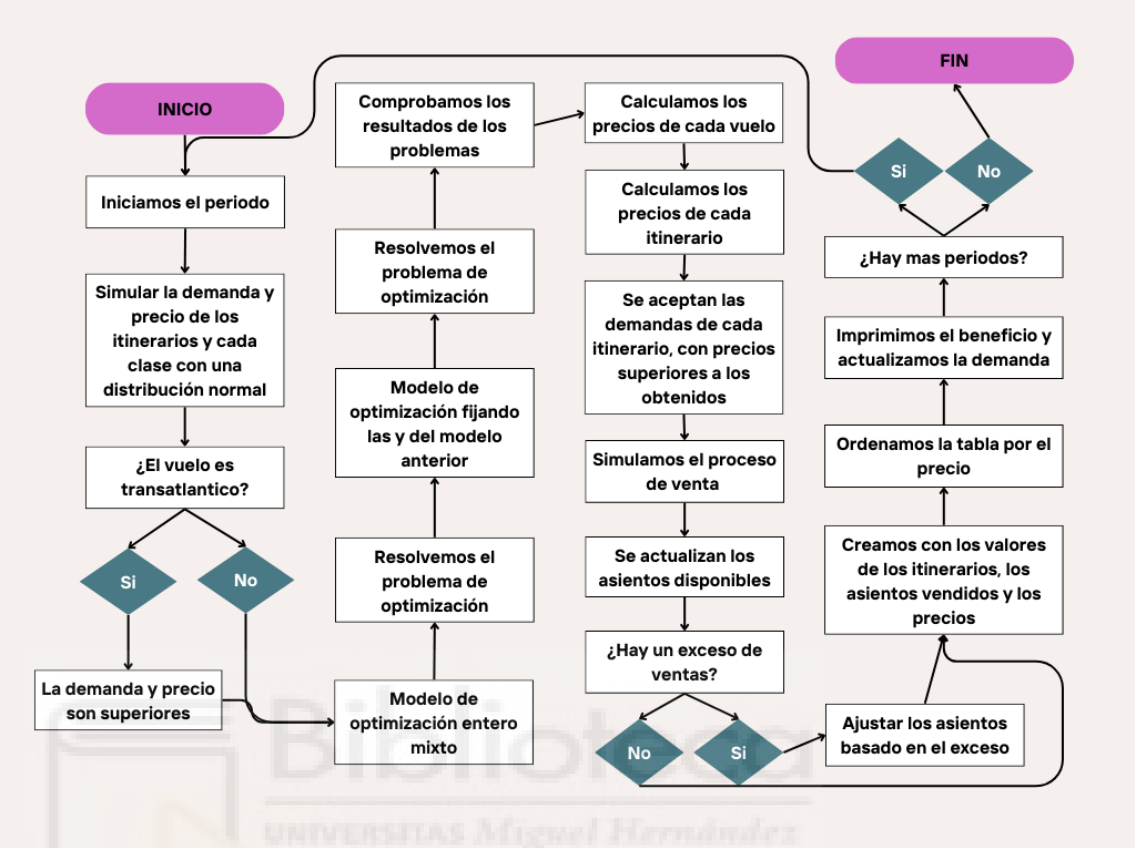
Modelo.

Para obtener la información correcta, hemos implementado la solución de dos modelos. Aunque estos modelos sean el mismo difieren en el tipo de variables que utilizan. En el modelo 1, las variables son enteras, lo que significa que no pueden tomar valores decimales, mientras que en el modelo 2, las variables son continuas y por lo tanto estás si pueden ser decimales.

La diferencia más importante es la información que obtenemos de cada modelo. En el modelo 1, además de la solución optima para la asignación de clientes a vuelos, obtendremos los valores de las variables binarias que indican qué vuelos deben aumentar su capacidad. Y estas variables las utilizamos en el modelo 2.

Al resolver ambos modelos, obtendremos una visión completa de la situación, utilizando los resultados del modelo 1 para alimentar y enriquecer la solución del modelo 2. Está estrategia nos permite tomar decisiones más informadas y precisas.

Pseudocodigo.



2- Diagrama 1, Fuente: Elaboración propia.

El diagrama del flujo ilustra el proceso de optimización, abarcando desde la simulación de la demanda y precios hasta la resolución de modelos de optimización y la actualización de capacidades y beneficios. Los precios de los itinerarios se calculan con los valores duales, sumando estos valores de las restricciones de capacidad de los vuelos que componen cada itinerario, se refleja el costo marginal de añadir un pasajero adicional. Esto ayuda en la optimización de ingresos al indicar el precio de venta adecuado. [3]

Comparaciones de los modelos.

La idea detrás de esta estrategia es evaluar tres situaciones diferentes y comparar cómo afecta cada enfoque a los beneficios totales. Al mantener las mismas condiciones iniciales en los tres escearios, como las capacidades iniciales de los vuelos y la demanda estimada, podemos realizar una comparación justa y obtener resultados representativos. Esta metodología nos permitirá analizar adecuadamente la eficacia de cada estrategia y determinar cuál ofrece la mejor rentabilidad. Para poder realizar estas comprobaciones habrá que realizarle tres modificaciones en el código para observar las soluciones deseadas en las tres comparaciones que hay que realizar.

1- Estableceremos a cero la restriccion que determina qué vuelo aumentar, manteniendo las capacidades de los vuelos con los mismos valores.

```
# Clases
Clases = [1, 2]

# Itinerarios   # Ten en cuenta que empiezan en 0
T = range(len(periodo))
I = range(len(itinerarios))
K = range(3)  # Sabemos que es 3 porque es 3 el número de pasos que hacemos
Q = range(len(Clases))
J = range(len(vuelos))

C = [186,186,186,186,186,186,186,186,190,190,190,190,287,287]

capacidad_maxima=0 # El número de vuelos que aumentamos
```

3- Imagen 1, Fuente:Elaboración propia

2- Mantendremos en cero la decisión de aumentar los vuelos y aumentaremos manualmente las capacidades de los vuelos que elijamos.

```
# Clases
Clases = [1, 2]

# Itinerarios   # Ten en cuenta que empiezan en 0
T = range(len(periodo))
I = range(len(itinerarios))
K = range(3)  # Sabemos que es 3 porque es 3 el número de pasos que hacemos
Q = range(len(Clases))
J = range(len(vuelos))

C = [186,186,186,186,186,186,186,186,190+50,190+50,190+50,190+50,287,287]

capacidad_maxima=0 # El número de vuelos que aumentamos
```

4- Imagen 2, Fuente:Elaboración propia

3- En la ultima comprobación mantendremos los valores en 0, y indicaremos el número de vuelos en los que queremos aumentar la capacidad en 50 asientos.

```
# Clases
Clases = [1, 2]

# Itinerarios   # Ten en cuenta que empiezan en 0
T = range(len(periodo))
I = range(len(itinerarios))
K = range(3)  # Sabemos que es 3 porque es 3 el número de pasos que hacemos
Q = range(len(Clases))
J = range(len(vuelos))

C = [186,186,186,186,186,186,186,186,190,190,190,190,287,287]

capacidad_maxima=4 # El número de vuelos que aumentamos
```

5- Imagen 3, Fuente:Elaboración propia

Después de implementar estas modificaciones y obtener los resultados correspondientes, podremos analizar como varian los

beneficios en cada escenario y qué enfoque se traduce en una mejor rentabilidad.

Precios aceptados.

Para obtener los precios que aceptaremos para la venta, realizamos una simulación de demanda y precios de venta similares a la del modelo. Para cada itinerario, determinamos la demanda y precios, de la misma manera, que los hemos estimado para el modelo, pero añadiedo la variable tiempo, para que vayan aumentando, según vamos acercandonos al periodo final.

A continuación, evaluamos si los precios de venta simulados son superiores a los precios minimos obtenidos del modelo de optimización. Si es así, registramos el númreo de itinerarios vendidos y los ingresos generados en cada clase. Si los precios simulados son demasiado bajos, la programación nos indicara que el precio de venta es insuficiente para garantizar la venta.

Este proceso nos permitira explorar diferentes esenarios y evaluar la sensibilidad de nuestros ingresos a cambios en la demanda y los preicos de venta, ayudandonos a tomar decisiones estrategicas informadas en la gestión de ingresos en itinerarios de vuelo.

Despues de aceptar los precios, se iniciara una lista para almacenar la informacion del itinerario, cantidad, precio y clase de los itinerarios vendidos que han superado el precio minimo. Para determinar el exceso de capacidad y ajuste de las ventas, recorreremos los vuelos e identificaremos los itinerarios asociados a cada vuelo. Y veremos si en este vuelo tenemos un exceso de ventas y disminuir las ventas en el vuelo, por lo tanto, para esto ordenaremos la lista por precios y se ajustaran las ventas para así eliminar el exceso.

Para poder obtener la información correcta en el siguiente periodo, realizaremos una actualizacion de la capacidad de vuelo, donde restamos la cantidad vendida en cada itinerario correspondiente al vuelo, y tambien verificaremos si es necesiario o no aumentar la capacidad del vuelo y disminuiremos las posibilidades de aumentar un vuelo en el siguiente periodo. Por ultimo, calcularemos el beneficio que obtenemos en cada vuelo de manera que multiplicamos el precio y cantidad de ocupaciones que hemos vendido el itinerario.

Este código es fundamental para el análisis y la toma de decisiones en la gestion de ingresos en itinerarios de vuelo, ya que proportciona información sobre las ventas realizadas, la capacidad utilizada y el beneficio generado en cada periodo.

Resultados.

Como bien hemos introducido al inicio del trabajo, exploraremos tres posibles escenarios para evaluar la importancia de nuestro modelo. En el primer escenario, el modelo decide en qué vuelos necesitamos aumentar la capacidad. En el segundo, aumentaremos la capacidad de algunos vuelos desde el principio. En el último escenario, no se realizará ningún aumento en la capacidad de los vuelos. Estas tres opciones nos permitirán comprender mejor el impacto y la eficacia de nuestro modelo en diferentes contextos.

Selección por modelo.

Con el aumento recomendado por el modelo, hemos obtenido un beneficio de 681.364€. Según el modelo, se sugiere aumentar la capacidad de los vuelos Miami-Madrid en el período 4, Miami-Madrid de nuevo y Miami-Santiago de Chile en el período 5, y Santiago de Chile-Miami en el período 8.

Selección personal.

Optamos por aumentar la capacidad de los vuelos que realizan trayectos transatlánticos, como Madrid-Miami, Miami-Madrid, Madrid-Santiago de Chile y Santiago de Chile-Madrid. Tras realizar estos aumentos, he obtenido un beneficio total de 624.755€.

Sin aumento.

Si no aumentamos la capacidad de los vuelos, según nuestra demanda y oferta estimadas, el beneficio sería de 568.152€.

El modelo nos ayuda a maximizar nuestros beneficios y, por ende, nuestra rentabilidad. Según se demuestra en el Anexo Vuelos por periodos [1].

Anexo.

```
import pandas as pd
import numpy as np
from docplex.mp.model import Model

# Salida y destino
itinerarios = [[1, 2], [1, 3], [1, 4], [2, 1], [2, 3],
               [2, 4], [3, 1], [3, 2], [3, 4], [4, 1],
               [4, 2], [4, 3],
               [5, 6], [6, 5], [5, 7], [7, 5], [6, 7], [7, 6]]

periodo = [1, 2, 3, 4, 5, 6, 7, 8]
# Pasos que hay que hacer para llegar al destino
vuelos = [[1, 5], [5, 1], [2, 5], [5, 2], [3, 6],
```

```
        [6, 3], [4, 7], [7, 4],
        [5, 6], [6, 5], [5, 7], [7, 5], [6, 7], [7, 6]]

# Para hacer el itinerario 1, tienes que hacer el vuelo 1 y el 4
# será -1 cuando no tengamos que hacer ningún otro vuelo
J_i = [[1, 4, -1], [1, 9, 6], [1, 11, 8], [3, 2, -1], [3, 9, 6],
      [3, 11, 8], [5, 10, 2], [5, 10, 4], [5, 13, 8], [7, 12, 2],
      [7, 12, 4], [7, 14, 6], [9, -1, -1], [10, -1, -1], [11, -1, -1], [12, -1, -1],
[13, -1, -1], [14, -1, -1]]  # continuar
# Clases
Clases = [1, 2]

# Itinerarios   # Ten en cuenta que empiezan en 0
T = range(len(periodo))
I = range(len(itinerarios))
K = range(3)  # Sabemos que es 3 porque es 3 el número de vuelos
que hacemos en cada itinerario
Q = range(len(Clases))
J = range(len(vuelos))

C = [186,186,186,186,186,186,186,186,190,190,190,190,287,287]
aumento_capacidad = 50
capacidad_maxima=0 # El número de vuelos que aumentamos
np.random.seed(55555)
D = np.zeros((len(I), len(T), len(Q)), dtype=int)
r = np.zeros((len(I), len(T), len(Q)))
d_value = 0

for i in I:
    for t in T:
        for q in Q:
            if i<8 and i>12:       # Ajusta las medias y desviaciones estándar
según las diferencias entre las clases
                if q == 0:
                    media = 12
                    desviacion_estandar = 4
                    mean = 60
                    desv_est = 25

                else:
                    media = 8
                    desviacion_estandar = 2
                    mean = 110
                    desv_est = 50
            else:
```

```
            if q == 0:
                media = 32
                desviacion_estandar = 12
                mean = 180
                desv_est = 60
            else:
                media = 20
                desviacion_estandar = 6
                mean = 340
                desv_est = 100

                    # Generar números aleatorios de distribuciones normales
            D[i][t][q] = np.random.normal(media, desviacion_estandar)
            if D[i][t][q]<0:
                D[i][t][q]=0
                # Almacena los valores en la matriz D
                        # Generar números aleatorios de distribuciones normales
            r[i][t][q] = np.random.normal(mean, desv_est)
            if r[i][t][q]<0:
                r[i][t][q]=0
tabla_final=[]
for t in T:
    print("Periodo {}".format(t+1))
    mdl = Model(name="vuelos")
    tiempo=t

            # Variable continua que indica el nº de clientes que aceptamos en cada iteración
    x_var = [[[mdl.integer_var(lb=0, ub=D[i][t][q], name="x_i%d_t%d_q%d" % (i + 1, t + 1, q + 1)) for q in Q] for t in T] for i in I]

    B = [[[mdl.integer_var(lb=0, name="B_i%d_t%d_q%d" % (i + 1, t + 1, q + 1)) for q in Q] for t in T] for i in I]
    y = [mdl.binary_var(name="y_j%d" % (j + 1)) for j in J]

    mdl.maximize(mdl.sum(r[i][t][q] * x_var[i][t][q] for i in I for t in T for q in Q))

            # Creamos una restricción por cada paso que hacemos en cada paso
    restricciones = [mdl.linear_expr() for j in J]
```

```
    restricciones2 = [[[mdl.linear_expr() for q in Q] for t in T] for i in I]

            # Restricciones para definir el valor de acumulativas
    for i in I:
                for t in T:
                    p = t - 1
                    for q in Q:
                        if t > 0:
                            restricciones2[i] = mdl.add_constraint(B[i][t][q] ==
B[i][p][q] + x_var[i][t][q])
                        else:
                            restricciones2[i] = mdl.add_constraint(B[i][t][q] ==
x_var[i][t][q])

    for j in J:
                itinerarios_por_vuelo = [[[mdl.linear_expr() for k in K] for t
in T] for i in I]
                for i in I:
                    for k in K:
                        if (J_i[i][k] == j + 1):
                            itinerarios_por_vuelo[i][t][k] = mdl.sum(B[i][t][q] for
q in Q)

                restricciones[j]                                    =
mdl.add_constraint(mdl.sum(itinerarios_por_vuelo[i][t][k] for k in K for i
in I) <= C[j] + 50 * y[j],
                                                "vuelo_%d" % (j + 1))

    mdl.add_constraint(mdl.sum(y[j] for j in J) == capacidad_maxima)
    k = mdl.solve(log_output=False)

    for j in J:
                valor_y = mdl.solution.get_value("y_j{}".format(j + 1))

                if valor_y > 0:
                    C[j] = C[j] + aumento_capacidad
                    print("el valor de y_j{} es {}".format(j + 1, valor_y))
                else:
                    C[j] = C[j]


    mdl2 = Model(name="fijo")

            # Variable continua que indica el nº de clientes que
aceptamos en cada iteración
```

```
    x_var2 = [[[mdl2.continuous_var(lb=0, ub=D[i][t][q],
name="x2_i%d_t%d_q%d" % (i + 1, t + 1, q + 1)) for q in Q] for t in T]
for i in I]

    B2 = [[[mdl2.continuous_var(lb=0, name="B2_i%d_t%d_q%d" % (i
+ 1, t + 1, q + 1)) for q in Q] for t in T] for i in I]

    mdl2.maximize(mdl2.sum(r[i][t][q] * x_var2[i][t][q] for i in I for t in T for
q in Q))

            # Creamos una restricción por cada paso que hacemos en
cada paso
    restricciones_fija = [mdl2.linear_expr() for j in J]

    restricciones_fija2 = [[[mdl2.linear_expr() for q in Q] for t in T] for i in
I]

    for i in I:
            for t in T:
                p = t - 1
                for q in Q:
                    if t > 0:
                        restricciones_fija2[i]                                =
mdl2.add_constraint(B2[i][t][q] == B2[i][p][q] + x_var2[i][t][q])
                    else:
                        restricciones_fija2[i]                                =
mdl2.add_constraint(B2[i][t][q] == x_var2[i][t][q])

    for j in J:
            itinerarios_por_vuelo = [[[mdl2.linear_expr() for k in K] for
t in T] for i in I]
            for i in I:
                for k in K:
                    if (J_i[i][k] == j + 1):
                        itinerarios_por_vuelo[i][t][k]  =  mdl2.sum(B[i][t][q]
for q in Q)

            restricciones_fija[j]                                             =
mdl2.add_constraint(mdl2.sum(itinerarios_por_vuelo[i][t][k] for k in K
for i in I) <= C[j],
                                                "vuelo_%d" % (j + 1))

            # Resuelve el modelo 2
    s = mdl2.solve(log_output=False)
```

```
    # Tras la realización de los modelos, como son dos modelos uno
con valores enteros y otro si esos valores enteros, vamos a ver ahora
    # si coinciden los resultados y como podemos ver si son distintos
nos saltara una alerta
    for i in I:
                for t in T:
                    for q in Q:
                        valor_x                                          =
mdl.solution.get_value("x_i{}_t{}_q{}".format(i + 1, t + 1, q + 1))
                        valor_x2                                         =
mdl2.solution.get_value("x2_i{}_t{}_q{}".format(i + 1, t + 1, q + 1))
                        valor_B                                          =
mdl.solution.get_value("B_i{}_t{}_q{}".format(i + 1, t + 1, q + 1))
                        valor_B2                                         =
mdl2.solution.get_value("B2_i{}_t{}_q{}".format(i + 1, t + 1, q + 1))

                        print("i{}_t{}_q{} x={} x2={} B={} B2={}".format(i + 1, t
+ 1, q + 1, valor_x, valor_x2, valor_B, valor_B2))

                        if valor_x-valor_x2<1 and valor_B-valor_B2>1:
                            print("Problema en B")
                        elif valor_x-valor_x2>1 and valor_B-valor_B2<1:
                            print("Problema en x")
                        elif valor_x-valor_x2>1 and valor_B-valor_B2>1:
                            print("No coincide ni x ni B")
    # Imprime las variables duales del modelo
    print("Variables duales del modelo:")
    for j in J:
                print("La variable dual de la capacidad del vuelo {} es {}.
La capacidad restante es {}".format(j + 1,
                                                    restricciones_fija[

j].dual_value,

restricciones_fija[

j].slack_value))
    # Ahora calcularemos la minima cantidad por la que aceptaremos la
compra de uno de nuestros vuelos
    prices = []
    print("Bid prices:")
    for i in I:
                prices.append(0)
                for k in K:
                    if J_i[i][k] > 0:
```

```
                    prices[i] = prices[i] + restricciones_fija[J_i[i][k] -
1].dual_value
                print("Itinerario {}: precio {}".format(i + 1, prices[i]))

    p = np.zeros((len(I), len(Q)))
    V = np.zeros((len(I), len(Q)), dtype=int)

    # Demandas por vuelo, clase y periodo
    for i in I:
        if i<8 and i>12:
                for q in Q:
                    if q == 0:
                        media = 14*tiempo
                        desviacion_estandar_d = 5+tiempo
                        mean = 60
                        desviacion_estandar_r = 25
                    else:
                        media = 8*tiempo
                        desviacion_estandar_d = 3+tiempo
                        mean = 110
                        desviacion_estandar_r = 50
        else:
            for q in Q:
                    if q == 0:
                        media = 32*tiempo
                        desviacion_estandar_d = 12+tiempo
                        mean = 180
                        desviacion_estandar_r = 60
                    else:
                        media = 20*tiempo
                        desviacion_estandar_d = 8+tiempo
                        mean = 340
                        desviacion_estandar_r = 100

                # Generar un solo número aleatorio de distribución
normal
                    V[i][q] = np.random.normal(media,
desviacion_estandar_d)
                    p[i][q] = np.random.normal(mean,
desviacion_estandar_r)

                # Almacena los valores en la matriz D
                    print(" La demanda del itinerario {} en la clase {} es {}
y el precio de venta es {}".format(i + 1, q + 1, V[i][q], round(p[i][q], 2)))
```

```
    euros1=[0]*len(I)
    acumulativa1=[0]*len(I)
    euros2=[0]*len(I)
    acumulativa2=[0]*len(I)
    beneficio=[0]*len(I)
    # En este bucle buscamos los valores que con la simulación de la demanda y los precios de venta sean superiores a los que obtenemos
    # resolviendo el modelo
    for i in I:
            for q in Q:
                if C[j]==0:
                    euros1[i]=0
                    euros2[i]=0
                    acumulativa1[i]=0
                    acumulativa2[i]=0
                else:
                    if p[i][q]>=prices[i]:
                        if q==0:
                            euros1[i]=round(p[i][q],2)
                            if V[i][q]<0:
                                acumulativa1[i]=0
                            else:
                                acumulativa1[i]=V[i][q]
                            print("El número de itinerarios {} vendidos es de {} a {} en la clase 1".format(i+1,acumulativa1[i],euros1[i]))
                        else:
                            euros2[i]=round(p[i][q],2)
                            if V[i][q]<0:
                                acumulativa2[i]=0
                            else:
                                acumulativa2[i]=V[i][q]
                            print("El número de itinerarios {} vendidos es de {} a {} en la clase 2".format(i+1,acumulativa2[i],euros2[i]))
                    else:
                        print("El número de itinerarios {} el precio es muy pequeño en la clase {}".format(i+1,q+1))


    # Cuando no tengamos capacidad en el vuelo, a los itinerarios que utilicen este vuelo le daremos el valor 0, para que no puedan interferir
    # en nuestro problema
    for j in J:
            # Aquí realizamos un bucle, para saber los itinerarios que utilizan cada vuelo
            Itinerarios_vuelo = []
```

```
        for i in I:
            for k in K:
                if J_i[i][k] == j + 1:
                    Itinerarios_vuelo.append(i)
        # A través del bucle anterior, obtendremos la lista de los itinerarios, y a través de este acumulado
        # calcularemos los valores
        for i in Itinerarios_vuelo:
            if C[j]==0:
                euros1[i]=0
                euros2[i]=0
                acumulativa1[i]=0
                acumulativa2[i]=0

    tabla_i=[]
    # En este tabla, tendremos el itinerario, la cantidad, el precio, y el tipo de los vuelos que han superado el precio que nos ha indicado
    # el modelo
    for i in I:
        if euros1[i] > 0:
            fila1 = [i, acumulativa1[i], euros1[i], 1]
            tabla_i.append(fila1)
        if euros2[i] > 0:
            fila2 = [i, acumulativa2[i], euros2[i], 2]
            tabla_i.append(fila2)
    # En este bucle, volveremos a cargar todos los itinerarios, y si no hay capacidad nos indicará que itinerarios tenemos que rechazar.
    # Primero crearemos la una tabla, donde acumularemos los valores de la tabla creada, para despues restar este valor a la capacidad
    # del vuelo y de esta manera saber el exceso que tenemos en este vuelo.
    # Despues ordenaremos la tabla para saber cual de nuestros itinerarios tiene un precio menor, y pasará por todos los itinerarios,
    # hasta que el itinerario sea 0, y le daremos el valor de cada itinerario a la tabla_i, para así editarla tambien.
    exceso = [0] * len(J)
    for j in J:
        print("En el vuelo {} realizamos los itinerarios:".format(j + 1))
        Itinerarios_vuelo = []
        for i in I:
            for k in K:
                if J_i[i][k] == j + 1:
                    Itinerarios_vuelo.append(i)
        print(Itinerarios_vuelo)
```

```
            acumulativa=0
            exceso[j]=0
            tabla = []
            for i in Itinerarios_vuelo:
                for fila in tabla_i:
                    if i ==fila[0] and fila[3]==1:
                        fila1 = [fila[0], fila[1], fila[2],fila[3]]
                        tabla.append(fila1)
                        acumulativa += fila[1]

                    elif i ==fila[0] and fila[3]==2:
                        fila2 = [fila[0], fila[1], fila[2],fila[3]]
                        tabla.append(fila2)
                        acumulativa += fila[1]
            # Ordenar tabla por precio
            if C[j]==0:
                exceso[j]=0
            else:
                exceso[j]=acumulativa-C[j]
            tabla_ordenada = sorted(tabla, key=lambda x: x[2])
            # Restar el exceso
            ite=0
            acu=0
            tipo=0
            euros=0
            for fila in tabla_ordenada:
                if C[j]==0:
                    fila[1]=0
                    for fila_i in tabla_i:
                            if fila_i[0] == ite and fila_i[2] == euros and fila_i[3] ==
tipo:
                                fila_i[1] = 0
                else:
                    if exceso[j] > 0:
                        if fila[1] >= exceso[j]:
                            fila[1] -= exceso[j]
                            exceso[j] = 0
                            ite = fila[0]
                            acu = fila[1]
                            euros = fila[2]
                            tipo = fila[3]
                            for fila_i in tabla_i:
                                if fila_i[0] == ite and fila_i[2] == euros and fila_i[3]
== tipo:
                                    fila_i[1] = acu
```

```
                    else:
                        exceso[j] -= fila[1]
                        fila[1] = 0
                        ite = fila[0]
                        acu = fila[1]
                        euros = fila[2]
                        tipo = fila[3]
                        for fila_i in tabla_i:
                            if fila_i[0] == ite and fila_i[2] == euros and fila_i[3]
== tipo:
                                fila_i[1] = 0

    # Obtendremos la impresión de las ventas que hemos realizado, tras
la resta del exceso
    print("La tabla de ventas por periodo")
    for fila in tabla_i:
        print(fila)
    # Restar las ventas realizadas a la capacidad del vuelo, para así
poder tener la capacidad del siguiente vuelo, además de ver si
tenemos la
    # necesidad de aumentar en este periodo
    for j in J:
        Itinerarios_vuelo = []
        for i in I:
            for k in K:
                if J_i[i][k] == j + 1:
                    Itinerarios_vuelo.append(i)
        for fila in tabla_i:
            for i in Itinerarios_vuelo:
                if i ==fila[0]:
                    C[j] -= fila[1]
        valor_y = mdl.solution.get_value("y_j{}".format(j + 1))

        if valor_y>0:
            if C[j]<50:
                capacidad_maxima=capacidad_maxima-1
                C[j] = C[j]
                print("Hemos añadido 50 asientos en {} y solo podremos
aumentar {} vuelos".format(j+1,capacidad_maxima))
            else:
                C[j] = C[j] - aumento_capacidad
        else:
            C[j] = C[j]
        print("Las capacidades de este periodo en el vuelo {} es
{}".format(j+1,C[j]))
```

```
    # Iterar sobre los periodos y procesar la información de la tabla_i
    for fila in tabla_i:
        # Multiplicar el segundo y tercer elemento para obtener el nuevo
valor
        nuevo_valor = fila[1] * fila[2]
        # Buscar si ya existe una fila con el mismo itinerario en tabla_final
        encontrado = False
        for index, fila_existente in enumerate(tabla_final):
            if fila_existente[0] == fila[0]:
                # Actualizar la fila existente con la nueva información
                tabla_final[index][1] += fila[1]  # Acumular el segundo
elemento
                tabla_final[index][2] += nuevo_valor  # Acumular el nuevo
valor
                encontrado = True
                break
        if not encontrado:
            # Si no se encuentra, agregar la nueva fila a tabla_final
            tabla_final.append([fila[0], fila[1], round(nuevo_valor,2)])

    # Imprimir la tabla final
    print("Tabla Final:")
    benef=0
    for fila in tabla_final:
        print(fila)
        benef+=fila[2]
print("El beneficio es:{}".format(benef))
```

Conclusiones.

En este trabajo, he ampliado mis conocimientos en el manejo de modelos de optimización en Python utilizando DOcplex. He aprendido a crear y definir variables continuas y enteras para modelos de optimización, así como a utilizar expresiones lineales y restricciones para modelar problemas de optimización y maximizar una función objetivo. Además, he mejorado en la gestión de restricciones en los modelos, incluyendo la implementación de restricciones para definir valores acumulativos, la creación de restricciones de capacidad para vuelos y su asociación con variables binarias y, la validación y ajuste de las restricciones según los resultados del modelo.

Por otro lado, he aplicado conocimientos adquiridos durante el grado, tales como la actualización y gestión de datos en bucles, la ordenación y ajuste de tablas, y el manejo de excepciones y errores en Python. También

he mejorado en la definición y uso de funciones en Python, y en la impresión, verificación y consolidación de resultados.
Por lo tanto me gustaría agradecer a mis tutores por ayudarme y monitorearme para poder estructurar y gestionar un complejo problema de optimización, manejar datos asociados a este problema.

Bibliografía.

1. Vuelos por periodos. Hoja de cálculo. Ernesto Molto Quiles.
2. Expected Future Value Decomposition Based Bid Price Generation for Large-Scale Network Revenue Management.
3. Valores duales.