UNIVERSITAT POLITÈCNICA DE VALÈNCIA

UNIVERSITAT POLITÈCNICA DE VALÈNCIA
CAMPUS D'ALCOI

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Politécnica Superior de Alcoy

Aplicación de precios dinámicos en el entorno empresarial.

Trabajo Fin de Grado

Grado en Administración y Dirección de Empresas

AUTOR/A: Mascarell Estruch, Joan

Tutor/a: Juan Pérez, Ángel Alejandro

Cotutor/a: Carracedo Garnateo, Patricia

CURSO ACADÉMICO: 2023/2024

RESUMEN

Desde siglos anteriores los precios han sido un instrumento fundamental para empresas y comerciantes. Permiten optimizar las ventas y aumentar la competitividad. Este trabajo se centra en el estudio del conocido precio dinámico, una estrategia de precios que se apoya de la determinación del mejor precio en un momento determinado y unas circunstancias dadas cómo la demanda o disponibilidad del producto.

Todo ello a través del nuevo fenómeno tecnológico, la inteligencia artificial. En concreto, el uso del conocido Machine Learning, una rama de la IA que se centra en desarrollar algoritmos que, a través de un conjunto de datos dados, aprende los patrones y comportamiento de estos con el objetivo de realizar predicciones.

En definitiva, al final de este proyecto entenderemos cómo funcionan algunos de los algoritmos de ML más conocidos aplicados a una estrategia de precios dinámicos a través de tres ejemplos prácticos de tres sectores totalmente diferentes.

Palabras clave: Precios dinámicos, modelos de predicción, Machine Learning, Inteligencia Artificial, algoritmos de aprendizaje.

RESUM

Des de segles anteriors, els preus han sigut un instrument fonamental per a empreses i comerciants. Permeten optimitzar les vendes i augmentar la competitivitat. Aquest treball se centra en l'estudi del conegut preu dinàmic, una estratègia de preus que es basa en la determinació del millor preu en un moment determinat i unes circumstàncies donades, com la demanda o la disponibilitat del producte.

Tot això a través del nou fenomen tecnològic, la intel·ligència artificial. En concret, l'ús del conegut Machine Learning, una branca de la IA que es centra a desenvolupar algorismes que, a través d'un conjunt de dades donades, aprenen els patrons i comportaments d'aquestes amb l'objectiu de realitzar prediccions.

En definitiva, al final d'aquest projecte, entendrem com funcionen alguns dels algorismes de ML més coneguts, aplicats a una estratègia de preus dinàmics, a través de tres exemples pràctics de tres sectors totalment diferents.

Paraules clau: Preus dinàmics, models de predicció, Machine Learning.

ABSTRACT

For centuries, prices have been a essential tool for businesses and shopkeepers, allowing them to enhance sales and increase competitiveness. This work focuses on the study of dynamic pricing, an assessing strategy that determines the best price at a given moment and under specific circumstances such as demand or product availability.

This is achieved through the new technological phenomenon of artificial intelligence. Specifically the use of Machine Learning (ML), wich is a branch of AI that develops algorithms that, through a given set of data, learn patterns and behaviors to make predictions.

In the end, by the end of this project, we will understand how some of the most well-known ML algorithms work when they are applied to a dynamic pricing strategy through three practical examples from three completely different sectors.

Keywords: Dynamic pricing, prediction models, Machine Learning, Artificial Intelligence, learning algorithms.

AGRADECIMIENTOS

Este trabajo se lo dedico a mi abuelo Casimiro, que desgraciadamente no podrá ver al primer miembro de la familia graduarse.

A todos los profesores y compañeros que han hecho que mi etapa en la universidad haya sido más divertida y me han ayudado. Mención especial para Josep Capó por todas las horas que ha invertido a guiarme en todas mis decisiones.

También a mi familia, que siempre han creído en mí y me han dado el empujón emocional necesario para afrontar las adversidades.

Y, por último, a mis dos pilares fundamentales en mi vida, mi madre Silvia y mi novia María. Las dos personas que más me han acompañado en este viaje, que han sufrido conmigo y ahora pueden estar orgullosas.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ACRÓNIMOS:

Q: cuartil

ML: Machine Learning

SK: Scikit-learn

API: Application Programming Interface

KNN= k-nearest neighbors

RFR= RandomForestRegressor

GBR= GradientBoostingRegressor

LR= LinearRegression

SVR= Support Vector Regression

1. Introducción

1.1. Contexto

Las empresas necesitan diariamente aumentar su eficiencia para no perder competitividad y oportunidades. La competitividad es crucial para su existencia y poder crecer. Con el paso del tiempo se han inventado nuevas herramientas que ayuden a mejorar la eficiencia operativa de éstas.

Un ejemplo de esta necesidad es en encontrar el precio que mejor se adecue a las características del cliente y momento de la compra de un servicio o producto. Además, se apoyan de la inteligencia artificial, un fenómeno extraordinario sin techo, que no se sabe hasta donde llegará. En concreto se utiliza el famoso Machine Learning o aprendizaje automático en español.

La inteligencia artificial y el aprendizaje automático han avanzado considerablemente, facilitando el análisis de grandes volúmenes de datos y la identificación de patrones y tendencias (Sutton, 2018). Estas herramientas permiten a las empresas a determinar decisiones cómo el precio dinámico a tiempo real.

1.2. Objetivos del trabajo

El objetivo del trabajo es básicamente fusionar dos ámbitos que en un futuro irán de la mano, la inteligencia artificial y los negocios. Aún desconocemos qué impacto tendrá la inteligencia artificial en nuestras vidas en general y sobre todo en los negocios.

En un futuro no tan lejano, las empresas que no se apoyen de IA morirán. De ahí nace mi curiosidad por fusionar mis conocimientos del mundo de los negocios y aprender el nuevo fenómeno de la inteligencia artificial.

El trabajo está enfocado en tres supuestos. En los tres supuestos el objetivo es aprender cómo funcionan los algoritmos mientras se explican conceptos teóricos. El resultado final será encontrar o determinar aquellos algoritmos que nos permitan obtener precios dinámicos en situaciones completamente diferentes. Por otro lado, estudiar el funcionamiento de la inteligencia artificial para comprender sus aplicaciones.

1.3. ODS

Los ODS, objetivos de desarrollo sostenible, son unas metas que las Naciones Unidas propuse en 2015 como parte de la Agenda 2030, un plan de acción global que busca mejorar el planeta.

En total existen 17 ODS, estos objetivos están diseñados para abordar una variedad de desafíos globales, incluyendo la pobreza, la desigualdad, el cambio climático, la degradación ambiental, la paz y la justicia. Compuestos por 169 metas que sirven para medir de manera cuantitativa si se están cumpliendo.

El objetivo de este apartado es analizar la relación que existe entre el TFG propuesto y los ODS.

Para ello se utilizará una tabla comparativa con todos los ODS existentes y se medirán dependiendo del grado de relación con el trabajo.

ODS	*Alto*	*Medio*	*Bajo*	*No hay relación*
ODS 1. Fin de la pobreza				X
ODS 2. Hombre cero				X
ODS 3. Salud y bienestar				X
ODS 4. Educación y calidad				X
ODS 5. Igualdad de género				X
ODS 6. Agua limpia y saneamiento				X
ODS 7. Energía asequible y no contaminante				X
ODS 8. Trabajo decente y crecimiento económico	X			

ODS 9. Industria, innovación e infraestructura	X			
ODS 10. Reducción de las desigualdades				X
ODS 11. Ciudades y comunidades sostenibles		X		
ODS 12. Producción y consumo responsables	X			
ODS 13. Acción por el clima		X		
ODS 14. Vida submarina				X
ODS 15. Vida de ecosistemas terrestres				X
ODS 16. Paz, justicia e instituciones sólidas				X
ODS 17. Alianzas para lograr los objetivos				X

Tabla 1. Relación del TFG con los ODS. Elaboración propia.

Aunque no podemos descartar que el trabajo propuesto tiene o puede tener influencia en la mayoría de los ODS propuestos por las Naciones Unidas, existen varios que tienen una relación bastante directe y fuerte.

Los ODS más relevantes relacionados con el TFG son: ODS 8: Trabajo decente y crecimiento económico, ODS 9: Industria, innovación e infraestructura, ODS 11: Ciudades y comunidades sostenibles, ODS 12: Producción y consumo responsables y el ODS 13: Acción por el clima.

La implementación de la IA en una estrategia de precios dinámicos optimiza la eficiencia operativa de las empresas, aumentando así la competitividad y por consecuencia ayuda a un desarrollo favorable del crecimiento económico. Además, representa una innovación tecnológica ya que, ayuda a mejorar la infraestructura y eficiencia de las industrias que mejor aprovechan este fenómeno.

Optimizar el precio en diferentes áreas como la del transporte terrestre o aéreo, e incluso optimizar el precio de las viviendas, contribuye a generar una gestión más eficiente de

estos reduciendo congestión de vehículos e incluso personas y mejorar la sostenibilidad al emitir menos CO2. Por lo tanto, podemos hablar de que, gracias a implementar la IA en una estrategia de precios dinámicos, ayuda de manera directa o indirecta a una mejora del cambio climático.

La IA facilita un consumo y una producción más responsables al ajustar la oferta y la demanda de manera precisa. Esto da lugar a una distribución más eficiente de las compras, ventas y producción, evitando tanto las compras excesivas como la sobreproducción.

1.4. Metodología

Para poder entender y profundizar sobre los conceptos e historia relacionados con el ML y los precios objeto de estudio en este trabajo, se han realizado estudios exhaustivos en diferentes portales web, libros y consultas a expertos trabajadores de la UPV y conocidos.

En cuanto a los algoritmos y parámetros establecidos, se ha comparado multitud de páginas web con diferentes ejemplos de modelos de ML, todo ello con el objetivo de encontrar la combinación de toda esa información y adaptación eficiente en este trabajo. Además, consultas a expertos por si hay posibles errores u omisiones y cursos online.

1.5. Orden documental

El objetivo de este apartado consiste en indicar la estructura de este documento. Se separará en capítulos y este será el orden:

Capítulo 1. Introducción

En este capítulo se introduce y se presenta el tema que se estudiará, explicando los objetivos que intenta abarcar y una justificación de la relevancia del tema elegido y el porqué de la importancia del nuevo fenómeno de la inteligencia artificial y los precios dinámicos.

Capítulo 2. Marco teórico e historia

En este capítulo introduciremos algún concepto teórico sobre precios y estrategias de fijación de precios ya que será necesario para entender mejor los ejemplos prácticos. Además, se estudiará la historia de los precios y evolución, lo que nos ayudará a comprender por qué las empresas o comerciantes cada cierto tiempo tienen la necesidad de cambiar su estrategia de precios.

Capítulo 3. Ejemplos prácticos

Este capítulo será crucial para ver la importancia del trabajo. Consiste en la realización de tres ejemplos prácticos, los dos primeros ejemplos son dos modelos de fijación de precios, cada uno de dos sectores diferentes. En el primero se utilizará una empresa de transporte, a la cual, tras un estudio de su base de datos se le aplica un modelo de ML para obtener un modelo de fijación de precios basado en los árboles de decisión. En segundo supuesto se genera un modelo de predicción de precios basado en redes neuronales para un conjunto de datos de diferentes características de viviendas de la ciudad de Boston. Por último, el tercero se trata de una comparación entre algunos de los modelos de ML para predecir precios para ver cual funciona mejor ante una base de datos de diferentes vuelos.

Todo ello mientras se exponen los conceptos teóricos sobre estadística y ML que se han necesitado para la elaboración del trabajo.

Capítulo 4. Conclusiones y mejoras

En este capítulo, se resumen los objetivos del estudio y los hallazgos clave logrados. Se incluye una evaluación crítica de las limitaciones encontradas durante la investigación y se sugieren posibles caminos para futuras investigaciones.

2. Marco teórico e historia

En este apartado se estudiarán diferentes definiciones y conceptos sobre los precios y sus estrategias. Además, su evolución histórica. Todo ello con el objetivo de entender la necesidad que tienen las empresas y comerciantes en general de adaptarse a las nuevas necesidades del mercado. En definitiva, focalizar la atención en la importancia histórica que tienen los precios en la sociedad.

2.1. Concepto

Los precios dinámicos son una estrategia de fijación de precios altamente popular en la actualidad, ya que permiten ajustar los precios de productos o servicios en función de diversas circunstancias específicas. Esta estrategia tiene como objetivo principal optimizar los recursos y maximizar los beneficios de una empresa.

La estrategia de precios dinámicos se define como una técnica que implica ajustar el precio de un producto o servicio según una serie de variables, como la demanda del mercado, la disponibilidad del producto, la competencia, la estacionalidad, el costo y otros factores relevantes (Pupavac, 2016).

La flexibilidad en la fijación de precios permite a las empresas adaptarse rápidamente a las condiciones cambiantes del mercado y tomar decisiones informadas para maximizar sus ingresos. Esta capacidad de respuesta es fundamental para aprovechar las oportunidades y mitigar los efectos de las condiciones desfavorables de la demanda (Chod, 2005).

Los precios dinámicos son una herramienta estratégica poderosa que permite a las empresas ajustar sus precios de manera inteligente y eficiente, con el fin de alcanzar sus objetivos financieros y mejorar su posición competitiva en el mercado.

2.2. Historia

Los algoritmos de fijación de precios han evolucionado desde simples estrategias basadas en costos hasta complejos modelos predictivos utilizando machine learning (Breiman, 2001) (Bishop, 2006).

La historia de los precios dinámicos se remonta a siglos pasados, cuando los comerciantes ajustaban los precios según la disponibilidad y la demanda de los productos. En aquel entonces, el trueque era una forma común de comercio, donde se intercambiaban productos sin la necesidad de utilizar dinero.

Durante el siglo XIX, se introdujo el concepto de precio fijo, donde los precios de los productos se establecen independientemente de la demanda que tuvieran en el mercado.

En el siglo XX, comenzó a popularizarse un método de fijación de precios conocido como discriminación de precios. Este método implicaba aplicar diferentes precios a un mismo producto según el perfil del cliente o ciertas condiciones específicas. Un ejemplo extremo de este tipo de discriminación fue la diferencia significativa en el precio de los billetes de ferrocarril en Estados Unidos, donde se cobraba un precio mucho menor a los locales en comparación con los afroamericanos (Hinz, 2011).

Tras la evolución de la tecnología y los métodos tradicionales de fijación de precios, se implementó un método más avanzado conocido como precios dinámicos. Este enfoque utiliza algoritmos y datos en tiempo real para ajustar los precios de los productos o servicios según diversos factores, como la oferta y la demanda, la hora del día, la ubicación del cliente, el historial de compras y otros datos relevantes (Mackay, 2021).

A lo largo de la historia, los métodos de fijación de precios han evolucionado para adaptarse a las necesidades del mercado y a las estrategias comerciales de las empresas.

2.3. Principios básicos y mecanismos

Los precios dinámicos son una estrategia que busca optimizar los beneficios, maximizar recursos y fidelizar clientes mediante la variación del precio de los productos o servicios. Esta práctica se basa en el análisis en tiempo real de datos, los cuales se recopilan, procesan y analizan de forma continua para una toma de decisiones óptima, con el objetivo de establecer el precio más adecuado en cada situación (El Youbi, 2023).

Para llevar a cabo esta técnica, se emplean algoritmos de fijación de precios, que son modelos matemáticos diseñados para automatizar y optimizar el proceso de fijación de precios. Estos algoritmos permiten ajustar los precios de manera dinámica y eficiente, teniendo en cuenta diversos factores como la demanda del mercado, la competencia, la estacionalidad y otros datos relevantes.

Además, los precios dinámicos facilitan la segmentación de clientes, una herramienta que divide a los clientes en segmentos o características similares con el fin de satisfacer sus necesidades de manera más efectiva (Meng, 2017). Esto es fundamental, ya que permite a la empresa personalizar los precios y estrategias para adaptarse de manera más precisa a las preferencias y comportamientos de sus clientes, lo que contribuye a mejorar la satisfacción del cliente y aumentar la lealtad hacia la marca.

2.3.1. Mecanismos de fijación de precios dinámicos más relevantes

- Precios basados en la demanda: Se ajusta en tiempo real según la demanda del mercado. Ha demostrado ser eficaz en sectores como las aerolíneas y los hoteles, donde la capacidad es fija a corto plazo y perecedera (Elmaghraby, 2003).
- Precios basados en el costo: El precio se fija en función de los costos de producción y distribución, buscando cubrirlos y obtener un margen de beneficio (Neubert, 2022).
- Precios basados en la competencia: El precio se fija en función de los precios de productos similares en el mercado, buscando ser competitivo (Gönsch, 2012).
- Precios personalizados: El precio se ajusta en función de las características del cliente, como su historial de compras o ubicación (Arslan, 2011).
- La fijación de estrategias basadas en reglas (Ngugi, 2024): implica establecer una serie de directrices o criterios, tales como metas, límites mínimos o máximos, y otros objetivos definidos. El propósito de estas estrategias es aplicar ajustes en los precios cuando se alcanzan estas directrices u objetivos predefinidos. Por ejemplo, si las ventas de un producto aumentan en un 50%, se activa un ajuste del precio del 15% con el fin de mantener el nivel de ventas sin comprometer la rentabilidad.

Los precios dinámicos son una herramienta que nos permite abarcar más y de manera más eficiente, múltiples puntos de venta:

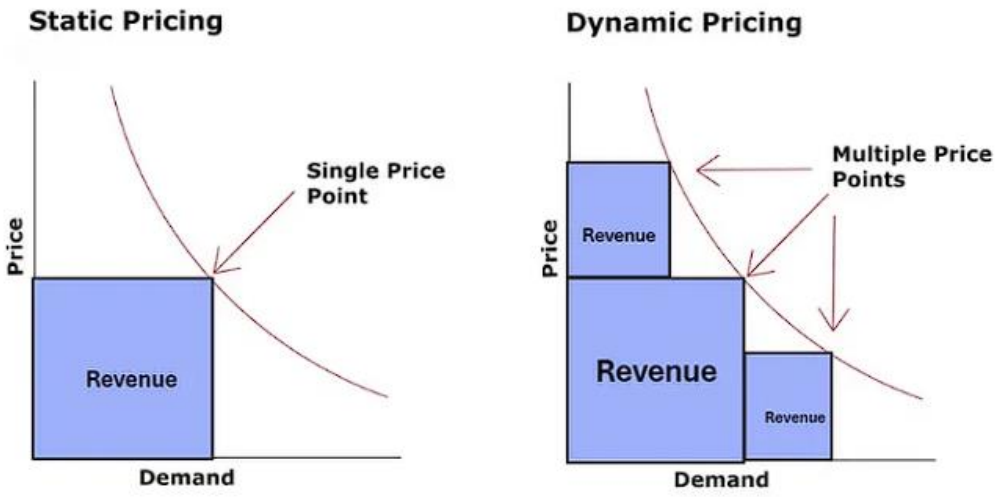

Figura 1. Diferencias entre precios fijos y precios dinámicos (Ngugi, 2024).

Implementar una estrategia de precios dinámicos implica ajustar los precios según las diferentes condiciones del mercado. Cuando la demanda disminuye, el precio se reduce, lo que estimula las ventas y ayuda a mantener un nivel óptimo de ingresos. Por el contrario, en períodos de alta demanda, los precios aumentan, lo que permite maximizar los ingresos sin comprometer el nivel de ventas.

En contraste, una política de precios fijos implica que el precio permanece inalterado independientemente de las fluctuaciones en la demanda u otros factores del mercado. Esto puede resultar en oportunidades perdidas para ajustarse a las condiciones cambiantes del mercado y optimizar los ingresos.

3. Ejemplos prácticos

En este apartado estudiaremos tres casos prácticos de diferentes características. También, servirá cómo apoyo para introducir conceptos de ML y IA.

3.1. Estrategia de precios dinámicos para la aplicación Dash

Dash es una aplicación de transporte que determinaba los precios de las rutas de manera ineficiente. En un sector tan competitivo como el del transporte, en el cual, existen servicios tradicionales cómo los Taxis, los autobuses y nuevos servicios y empresas cómo Uber o incluso Bablacar, tener un sistema ineficiente de fijación de precios propiciará una pérdida importante de cuota de mercado.

Por lo tanto, en este estudio se creará un modelo de Machine Learning para optimizar y determinar de manera eficiente el precio del servicio de transporte.

3.1.1. Base de datos

La base de datos recopila un total de 1000 datos con perfiles distintos de cada cliente sujeto a un conjunto de variables que abarcan diferentes aspectos del servicio de transporte, desde la ubicación y el tipo de membresía del cliente hasta la hora del día del viaje y el tipo de vehículo utilizado. Con el objetivo de abarcar todos los posibles perfiles de clientes actuales y potenciales, así como los distintos factores típicos en un transporte cómo el tiempo o la distancia.

Los datos se han organizado en categorías para facilitar su análisis y comprensión:

- **Location_Category**: Categoría de ubicación del viaje (urbana, suburbana, rural).
- **Customer_Loyalty_Status**: Estado de lealtad del cliente (regular, plata, oro).
- **Time_of_Booking**: Hora del día en que se realiza la reserva del viaje (mañana, tarde, noche, tarde-noche).
- **Vehicle_Type**: Tipo de vehículo utilizado para el viaje (premium, económico).
- **Expected_Ride_Duration**: Duración esperada del viaje.
- **Historical_Cost_of_Ride**: Costo histórico del viaje.
- **Number_of_Riders**: Número de pasajeros en el viaje.

- **Number_of_Drivers**: Número de conductores asignados al viaje.
- **Number_of_Past_Rides**: Número de viajes previos del cliente.
- **Average_Ratings**: Calificación promedio del cliente.

Tras entender mejor el conjunto de datos que será la base para realizar nuestro modelo se importará al software elegido con el lenguaje de programación de Python, un lenguaje de programación de código abierto. El software elegido es Google Collab, una aplicación gratuita de Google que permite ejecutar y escribir código Python, en específico Python versión 3.

3.1.2. Análisis descriptivo del conjunto de datos

Una vez el conjunto de datos ya está introducido en el software, importamos las primeras bibliotecas.

```
import pandas as pd
import plotly.express as px
import plotly.graph_objects as go

data = pd.read_csv("/content/dynamic_pricing.csv")
print(data.head())
```

Figura 2 Importar librerías y base de datos. Elaboración propia.

La librería pandas se centra en el análisis de datos, ofrece herramientas para manipular datos tabulares. Por otro lado, Plotly es una biblioteca de visualización interactiva que permite crear gráficos dinámicos en Python.

La función print(data.head()) nos permite visualizar las primeras 5 filas.

	Number_of_Riders	Number_of_Drivers	Location_Category	\
0	90	45	Urban	
1	58	39	Suburban	
2	42	31	Rural	
3	89	28	Rural	
4	78	22	Rural	

	Customer_Loyalty_Status	Number_of_Past_Rides	Average_Ratings	\
0	Silver	13	4.47	
1	Silver	72	4.06	
2	Silver	0	3.99	
3	Regular	67	4.31	
4	Regular	74	3.77	

	Time_of_Booking	Vehicle_Type	Expected_Ride_Duration	\
0	Night	Premium	90	
1	Evening	Economy	43	
2	Afternoon	Premium	76	
3	Afternoon	Premium	134	
4	Afternoon	Economy	149	

	Historical_Cost_of_Ride
0	284.257273
1	173.874753
2	329.795469
3	470.201232
4	579.681422

Figura 3 Primeras 5 datos de la base de datos. Elaboración propia.

Después, se lleva a cabo un análisis exploratorio de datos para observar con mayor detalle las estadísticas descriptivas de los datos.

El análisis descriptivo es el aspecto más básico de la estadística, y es utilizado para describir los datos. Su objetivo es resumir las características importantes del conjunto de datos de forma clara y concisa (Wilks, 2011).

En otras palabras, visualizar de manera más simple sin introducirnos en la modelización de los datos para ver de manera más rápida y simple cómo se comportan con el objetivo de tomar decisiones de manera más eficiente.

	Number_of_Riders	Number_of_Drivers	Number_of_Past_Rides	Average_Ratings	Expected_Ride_Duration	Historical_Cost_of_Ride
count	1000.0000	1000.0000	1000.000000	1000.000	1000.00000	1000.000000
mean	60.372000	27.076000	50.031000	4.257220	99.58800	372.502623
std	23.701506	19.068346	29.313774	0.435781	49.16545	187.158756
min	20.000000	5.000000	0.000000	3.500000	10.00000	25.993449
25%	40.000000	11.000000	25.000000	3.870000	59.75000	221.365202
50%	60.000000	22.000000	51.000000	4.270000	102.00000	362.019426
75%	81.000000	38.000000	75.000000	4.632500	143.00000	510.497504
max	100.00000	89.000000	100.000000	5.000000	180.00000	836.116419

Tabla 2. Principales resultados descriptivos. Elaboración propia.

Estos son los resultados obtenidos:

- **Count;** el número de observaciones totales para cada variable.
- **Media;** el promedio del valor de cada variable.
- **Desviación estándar**; que muestra la dispersión de los datos alrededor de la media. Por ejemplo, en el caso del número de pasajeros, la media de pasajeros son 60.372000, por lo tanto, de normal se espera esa cantidad de pasajeros, pero, la desviación estándar nos proporciona la información de que puede variar en 23,7 pasajeros de más o de menos. En consecuencia, la desviación nos ofrece un rango mayor de los números de pasajeros que se espera, de entre 36,67 hasta 84,07 pasajeros.
- **Min;** el valor más pequeño de entre todos los valores, siguiendo el ejemplo de la primera variable, el mínimo de pasajeros son 20.
- **Max;** el valor más elevado.

- **Percentil 25:** este percentil indica que un 25% de los datos estudiados tiene un valor de 40 o menos en el ejemplo estudiado.
- **Percentil 50:** conocido también como la mediana, indica que un 50% de los datos tiene al menos un valor de 60 pasajeros.
- **Percentil 75:** si el percentil 75% es 81, significa que al menos el 75% de los datos del número de pasajeros tienen un valor de 81 pasajeros o menos. Ayuda a entender qué proporción de los casos tienen valores relativamente altos.

Posteriormente se genera un gráfico de dispersión, un tipo de diagrama matemático que utiliza las coordenadas cartesianas para mostrar los valores de dos variables para un conjunto de datos, (Jarrell, 2019).

Las dos variables que se analizan es la relación entre la relación entre la duración esperada del viaje y el costo histórico del viaje, con el objetivo de determinar si existe algún tipo de patrón entre estas.

```
# Plot a scatter plot for the expected ride duration vs historical
cost of ride
fig = px.scatter(data, x='Expected_Ride_Duration',
                 y='Historical_Cost_of_Ride',
                 title='Expected Ride Duration vs Historical Cost
of Ride',
                 trendline='ols')
fig.show()
```

Figura 4. Gráfico de diagramas de dispersión para la duración esperada. Elaboración propia.

A simple vista se aprecia una clara tendencia ascendente, a mayor duración del viaje, los costes históricos aumentan. Esto proporciona información valiosa ya que indica que un factor crucial en el modelo es que a mayor distancia se recorra el precio debe ser mayor para poder soportar los gastos más elevados y no disminuir el beneficio. Puede ser debido al gasto de combustible o, si un mismo pasajero recorre un trayecto de mayor tiempo, si no paga un precio más elevado se pierde la oportunidad de que otro pasajero pueda de que otro pasajero tome el viaje.

Otra manera de estudiar el coste histórico es a través de un diagrama de cajas. Un diagrama de caja, del inglés, boxplot, es una representación de una variable cuantitativa o categórica con el propósito de identificar rápidamente los cuartiles del conjunto de datos (Rodó, 2021).

Se compara los costes históricos con el momento del día, con el objetivo de analizar si el momento del día influye en la cantidad de coste que soportan los clientes.

```
fig = px.box(data, x='Time_of_Booking',
             y='Historical_Cost_of_Ride',
             title='Historical Cost of Ride Distribution by Time of
Booking')
fig.show()
```

Figura 5. Gráfico de diagrama de barras. Elaboración propia.

Antes de interpretar el gráfico hay que tener claro que son los cuartiles, cómo dice (Westreicher, 2021) un cuartil es cada uno de los tres valores que pueden dividir un grupo de números, ordenados de menor a mayor, en cuatro partes iguales. Estas divisiones son ejecutadas a través de los 3 percentiles, el de 25%, 50% conocido cómo la mediana y el de 75%.

El punto de referencia será el Q2 ya que es el cuartil dos, que representa la mediana y divide el conjunto de datos en dos partes iguales, con el 50% de los datos por encima y el 50% por debajo de este valor.

El resultado es que por la tarde los costes son más elevados, exactamente un valor de 387,47. Esto puede ser debido a que por la tarde es cuando más tráfico de vehículos y personas hay, por lo tanto, el tiempo suele ser mayor cómo se ha estudiado anteriormente.

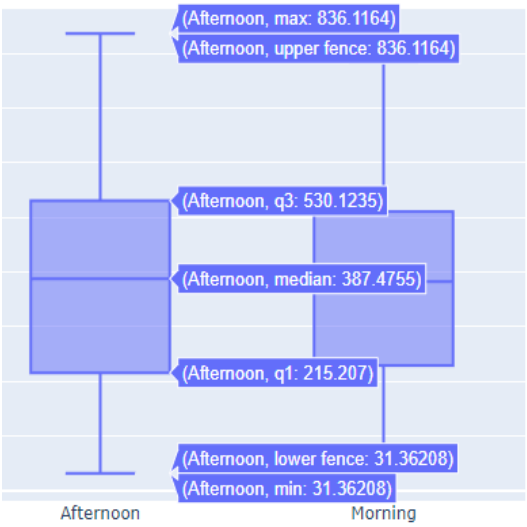

Figura 6. Resultados del diagrama de cajas. Elaboración propia.

Seguido de por la mañana 382,2, aquí el tráfico es menor ya que es el horario normal de trabajo y estudios, pero aún hay movimiento y el tráfico puede verse congestionado.

Cuando anochece, el tráfico es más fluido debido a que las personas suelen estar ya en casa, pero aun así la cifra es de 360,7.

Por la noche es el valor más reducido, exactamente de 336,09, ya que, a estas horas pocas personas suelen estar en movimiento y el tráfico tiende a ser mucho más ligero debido al descenso significativo de la actividad en comparación con el día y la tarde, además el tráfico de vehículos pesados es considerablemente menor.

Por lo tanto, podemos concluir que el momento del día donde se necesite el servicio de transporte afecta a los precios.

El tipo de vehículo con el que se presta el servicio también puede ser un factor determinante ya que, un coche económico no tiene las mismas necesidades que uno de lujo o premium.

Por ello, se ha generado otro diagrama de cajas:

```
fig = px.box(data, x='Vehicle_Type',
             y='Historical_Cost_of_Ride',
             title='Historical Cost of Ride Distribution by Vehicle
Type')
fig.show()
```

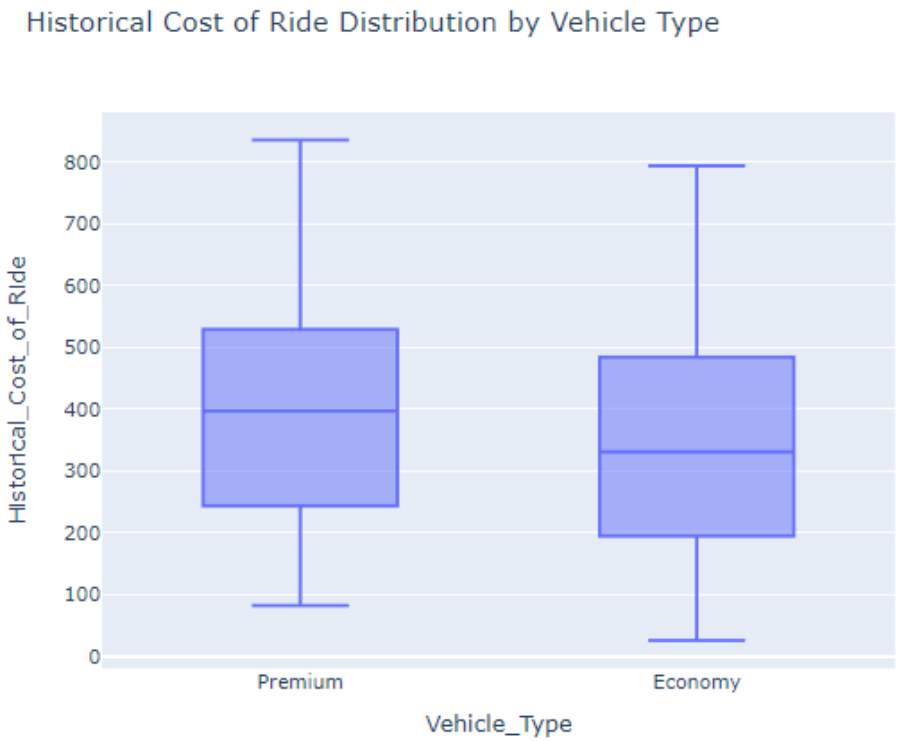

Figura 7. Diagrama de cajas para el costo del viaje según el tipo de vehículo. Elaboración propia.

Este diagrama busca comparar si el tipo de vehículo afecta a los costes. Siguiendo la mediana o Q2 que es de un valor de 397,5 en el caso del premium, vemos que es razonablemente más caro o tiene más costes para el consumidor este servicio ante el económico que es de 331,23.

Además, el coste mínimo en el caso del coche premium es de 82,52, mientras que en el caso del coche económico es de 25,99.

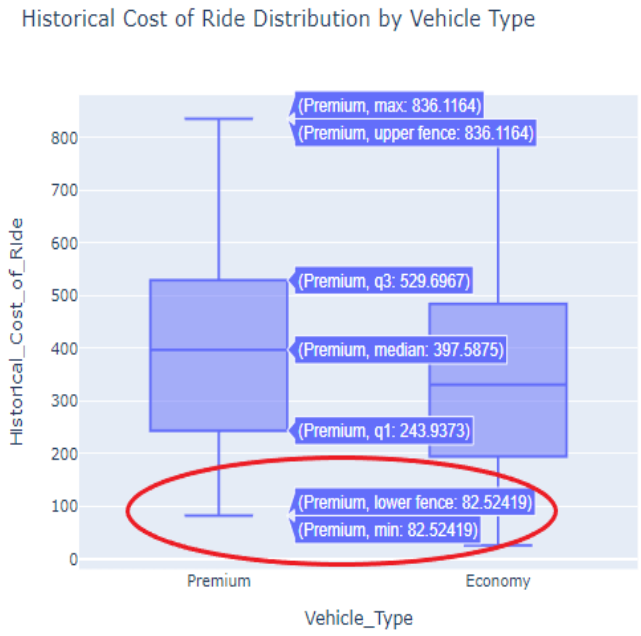

Figura 8. Resultados numéricos figura 7. Elaboración propia.

Para finalizar el análisis exploratorio de los datos, se genera una matriz de correlación.

La correlación consiste en indicar si existe relación entre dos variables, o, dicho en otras palabras, si la variación de una variable afecta a otra. Además, indica la fuerza de esta relación

Una correlación cercana a 1 indica una relación positiva, lo que significa que, si el valor de una variable aumenta, el valor de la otra variable tiende a aumentar también. En el caso de una correlación de 1, la relación es perfecta, lo que significa que las variables aumentan en la misma proporción.

Una correlación cercana a -1 indica una relación negativa, lo que significa que, si el valor de una variable aumenta, el valor de la otra variable tiende a disminuir en proporción. En el caso de una correlación de -1, la relación es perfecta pero negativa.

Una correlación cercana a 0 indica que no hay una relación lineal entre las variables.

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Cargar los datos desde el archivo CSV
data = pd.read_csv("/content/dynamic_pricing.csv")

# Eliminar columnas no numéricas
data_numeric = data.select_dtypes(include=['float64', 'int64'])

# Calcular la matriz de correlación
correlation_matrix = data_numeric.corr()

# Visualizar la matriz de correlación como un mapa de calor
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm',
fmt=".2f")
plt.title('Matriz de Correlación')
plt.show()
```

Figura 9. Código matriz de correlación. Elaboración propia.

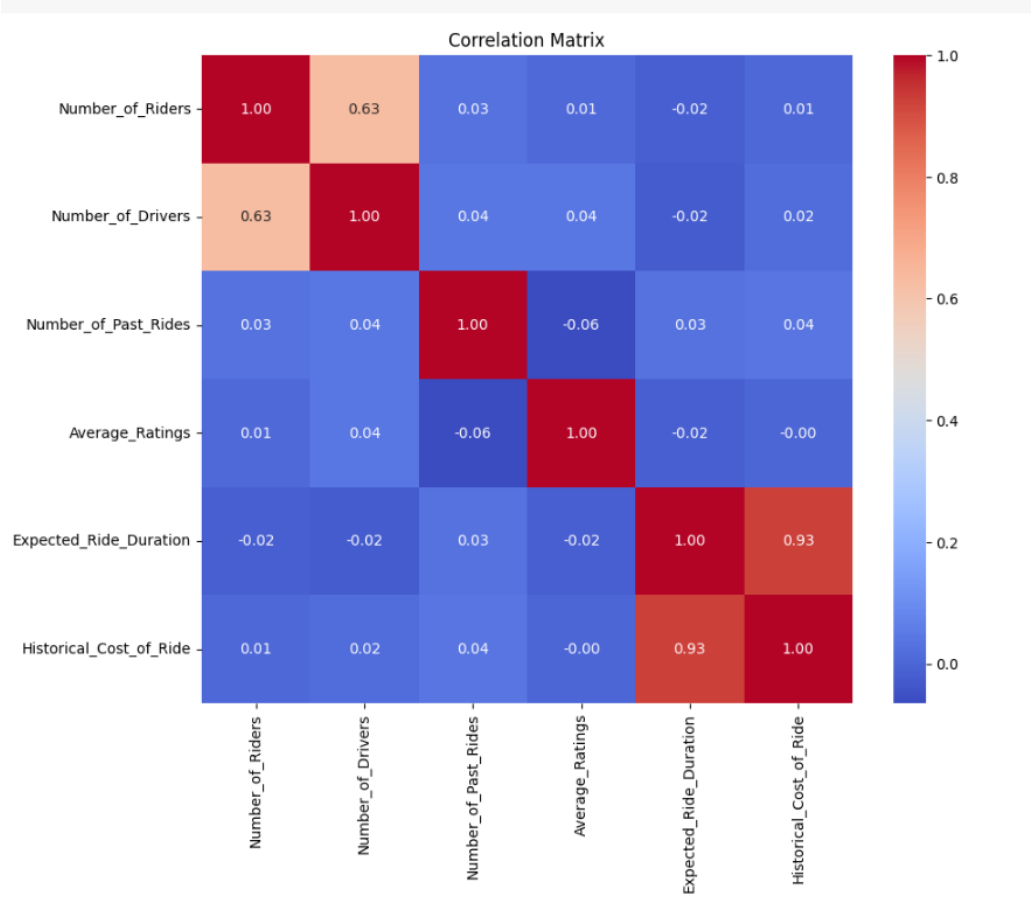

Tabla 3. Matriz de correlación. Elaboración propia.

Existe una correlación de 0,93, duración esperada del viaje y el costo histórico del viaje. Además, es una correlación positiva casi perfecta, por lo tanto, si se estima que la duración del viaje será elevada, los costes aumentarán casi en la misma proporción.

Existe otra correlación positiva entre el número de conductores y número de pasajeros de 0,63, es una información útil pero no tan relevante debido a que no afecta directamente al coste y que la correlación no es tan fuerte.

Este gráfico, además, nos indica que en el conjunto de datos estudiado solo se tiene en cuenta la duración esperada del viaje para establecer los costos para el consumidor, o dicho de otra manera, el precio para la empresa. Por lo tanto, esta empresa necesita un sistema más eficiente de fijación de precios, que tenga en cuenta más variables que afecten al precio. Sumado a eso, un sistema que se adapte a los cambios de cada situación para determinar un precio lo más ajustado posible.

3.1.3. Estrategia de precios dinámicos

El objetivo es implantar una estrategia de precios que se adapte a los costes de los viajes según el nivel de oferta y demanda. En situaciones de alta demanda y pocos conductores el precio aumentará y por el contrario en situaciones de poca demanda y exceso conductores el precio reducirá.

Lo que supone para la empresa optimizar los precios ya que en una situación de alta demanda los precios aumentan y se generan más ingresos y ante un escenario de demanda baja, los precios disminuyen para atraer a los clientes. Por lo tanto, equilibra la oferta y la demanda porque ante demandas muy elevadas si aumentan los precios la demanda disminuye y no se genera saturación, por el contrario, si existe una demanda baja los precios disminuyen, generando una mayor demanda.

Otro factor importante es que ayuda a la competitividad empresarial puesto que los precios se adaptan a la demanda y al consumidor de manera más eficiente, esto genera que los clientes prefieran la empresa frente a otras.

Tras explicar los beneficios y objetivos del modelo, se procede a avanzar en la implementación del código:

```
import numpy as np
# Calcular el multiplicador de demanda basado en percentiles para
alta y baja demanda
high_demand_percentile = 75
low_demand_percentile = 25
data['demand_multiplier'] = np.where(data['Number_of_Riders'] >
np.percentile(data['Number_of_Riders'], high_demand_percentile),
                                     data['Number_of_Riders'] /
np.percentile(data['Number_of_Riders'], high_demand_percentile),
                                     data['Number_of_Riders'] /
np.percentile(data['Number_of_Riders'], low_demand_percentile))

# Calcular el multiplicador de oferta basado en percentiles para
alta y baja oferta
high_supply_percentile = 75
low_supply_percentile = 25
data['supply_multiplier'] = np.where(data['Number_of_Drivers'] >
np.percentile(data['Number_of_Drivers'], low_supply_percentile),
                                     np.percentile(data['Number_of_
Drivers'], high_supply_percentile) / data['Number_of_Drivers'],
                                     np.percentile(data['Number_of_
Drivers'], low_supply_percentile) / data['Number_of_Drivers'])

# Definir los factores de ajuste de precios para alta y baja
demanda/oferta
demand_threshold_high = 1.2 # Umbral de alta demanda
demand_threshold_low = 0.8 # Umbral de baja demanda
supply_threshold_high = 0.8 # Umbral de alta oferta
supply_threshold_low = 1.2 # Umbral de baja oferta

# Calcular el costo ajustado del viaje para precios dinámicos
data['adjusted_ride_cost'] = data['Historical_Cost_of_Ride'] * (
    np.maximum(data['demand_multiplier'], demand_threshold_low) *
    np.maximum(data['supply_multiplier'], supply_threshold_high)
```

Figura 10. Implementación de la estrategia de precios dinámicos. Elaboración propia.

Primero se define los percentiles con el objetivo de definir la demanda alta y baja. El percentil 25 representa los niveles de demanda baja, mientras que el percentil 75 niveles de demanda alta.

La finalidad de definir estos percentiles es obtener el multiplicador de la demanda, el multiplicador de la demanda se utiliza para comparar los niveles de demanda basados en las referencias actuales estadísticas (los percentiles) frente a los nuevos niveles con el propósito de ajustar los precios de manera dinámica y adecuada según las condiciones cambiantes del mercado y garantizar una eficiente asignación de recursos.

Para calcular el multiplicador de la demanda, en el caso de ser baja (igual o menor al 25%) se utiliza la siguiente fórmula:

$$Multiplicador\ demanda\ baja = \frac{Número\ de\ pasajeros}{Percentil\ 25\ demanda}$$

Por lo tanto, si el número de pasajeros es menor al percentil estadístico 25 (el 25% de los datos con menor valor), divide el número de pasajeros entre ese mismo percentil para obtener el multiplicador.

En el caso del multiplicador de la demanda en un escenario alto el procedimiento es el mismo, pero sobre el percentil 75.

$$Multiplicador\ demanda\ alta = \frac{Número\ de\ pasajeros}{Percentil\ 75\ demanda}$$

Asimismo, calcula el multiplicador de la oferta, utiliza un procedimiento que en el de la demanda. Con la finalidad de obtener un valor que refleje la variación actual de la oferta frente a la histórica o estadística.

$$Multiplicador\ oferta\ baja = \frac{Percentil\ 25\ oferta}{Número\ de\ pasajeros}$$

$$Multiplicador\ oferta\ alta = \frac{Percentil\ 75\ oferta}{Número\ de\ pasajeros}$$

En este problema partiremos en el supuesto económico de que la diferencia que radica entre la demanda y la oferta es que son fenómenos opuestos, o, en otras palabras, inversamente proporcional. En el caso de que la demanda sea alta, la oferta es baja y viceversa.

Tras esto, se definen 4 umbrales;

- **Umbral de demanda alta**: con un valor de 1,2. Define si la demanda actual se considera alta ya que, en el caso de que el multiplicador proporcione un valor mayor, significa que la demanda actual es mayor a la histórica.

- **Umbral de demanda baja**: valor de 0,8. La demanda actual se considera baja cuando el multiplicador proporciona un valor menor que el umbral de demanda más bajo. Esto indica que la demanda actual es menor que la histórica en relación con el nivel estadístico representado por el percentil de baja demanda.
- **Umbral de oferta alta**: valor de 0,8. En el caso de que el multiplicador calcule un valor menor significa que la oferta actual es mayor a la histórica.
- **Umbral de oferta baja**: valor de 1,2. Si el multiplicador de oferta es mayor que este umbral, significa que la oferta actual es baja en comparación con los niveles históricos.

Seguidamente, se calcula el coste del viaje ajustado, multiplicando el máximo del multiplicador de la demanda con el umbral inferior de la demanda y el máximo del multiplicador de la oferta con el umbral superior de la demanda. Lo que se consigue es reflejar el mayor impacto de la demanda y la oferta en el precio ajustado y además fijar los límites que proporcionan los umbrales.

Tras implementarlo, se puede concluir que viajes son rentables y cuáles no a través de este código;

```
# Calculando el porcentaje de beneficio para cada viaje usando una
estrategia de precios dinámicos
data['profit_percentage'] = ((data['adjusted_ride_cost'] -
data['Historical_Cost_of_Ride']) / data['Historical_Cost_of_Ride'])
* 100

# Identificar viajes rentables cuando el porcentaje de beneficio es
positivo
profitable_rides = data[data['profit_percentage'] > 0]

# Identificar viajes con pérdidas cuando el porcentaje de beneficio
es negativo
loss_rides = data[data['profit_percentage'] < 0]

# Calcular la cantidad de viajes rentables y con pérdidas
profitable_count = len(profitable_rides)
loss_count = len(loss_rides)
```

Figura 11. Calculo viajes rentables. Elaboración propia.

El código básicamente lo que hace es calcular el porcentaje del beneficio actualizado, comparando el coste dinámico nuevo y el coste histórico con la siguiente fórmula;

$$Porcentaje\ de\ beneficio = \frac{(Costo\ ajustado\ - Coste\ histórico)}{Coste\ histórico}\ x\ 100$$

Sí el resultado es positivo se considera un viaje rentable, si es negativo se considera no rentable. Tras realizar un recuento del número de viajes rentables y no rentables, se genera un gráfico.

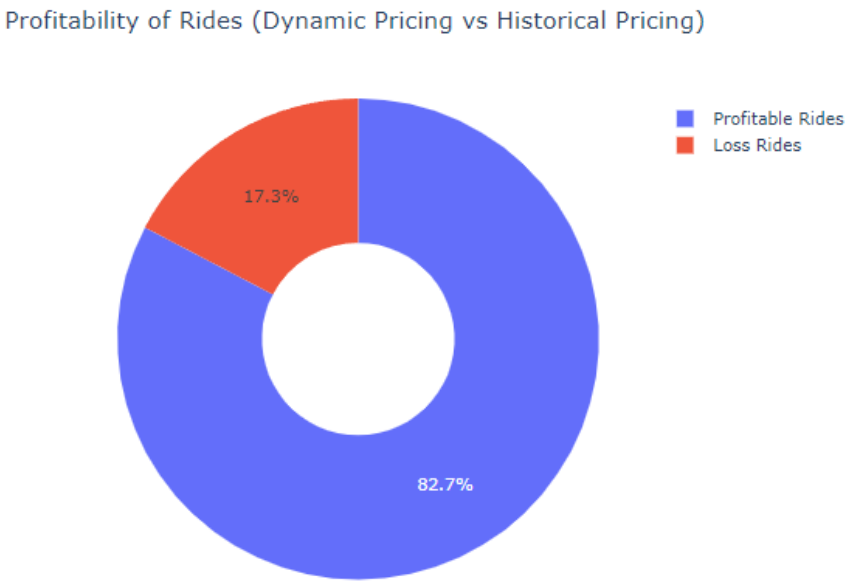

Figura 12. Viajes rentables vs. No rentables. Elaboración propia.

La información que proporciona el gráfico es que un 173 (17,3%) viajes, después de tener en cuenta los niveles de oferta y demanda y además ajustar el precio a dichos niveles, no son rentables.

Si se compara el diagrama de dispersión antes de determinar los nuevos precios adaptados y el actual podemos observar que la dispersión ha aumentado significativamente ya que el precio es determinado por nuevas variables.

```
fig = px.scatter(data,
                 x='Expected_Ride_Duration',
                 y='adjusted_ride_cost',
                 title='Expected Ride Duration vs Cost of Ride',
                 trendline='ols')
fig.show()
```

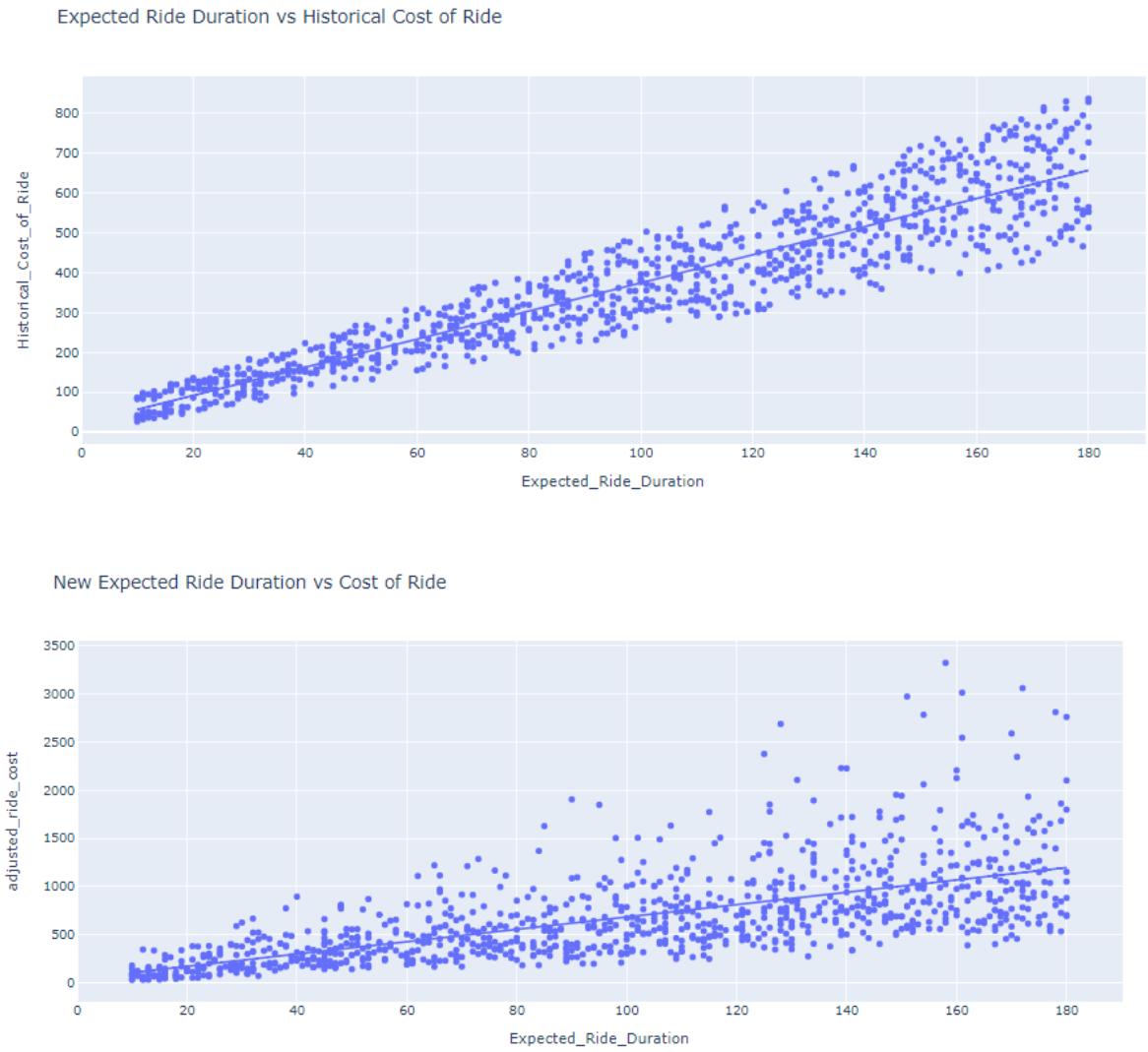

Figura 13. Diagrama dispersión para comparar duración del viaje y coste del viaje entre utilizar una estrategia de precios dinámicos y no utilizarla. Elaboración propia.

3.1.4. Procesamiento de datos y aplicación del modelo.

Tras finalizar todos los preparativos previos para definir en qué variables se apoya la fijación de los precios dinámicos, es la hora de crear el modelo de ML que ayude a predecir nuevos precios dinámicos.

En concreto, el modelo utilizado es el Random Forest Regression Model. Un modelo de árboles aleatorios crea múltiples árboles de decisión que representan nodos, cada árbol elige los mejores datos para elaborar una predicción. Posteriormente, se realiza el promedio de los resultados y obtenemos un único resultado. En el supuesto de predicción de precios de vuelos se explicará más profundamente el porqué es un buen modelo de predicción.

Además, se trata de un modelo de regresión lineal que según el portal web (DATAtab, 2024) el objetivo de una regresión lineal simple es predecir el valor de una variable dependiente a partir de una o varias variables independientes.

En este caso la variable dependiente es el coste del viaje ajustado, mientras que las independientes son el número de pasajeros, número de conductores, tipo de vehículo, hora de reserva y la duración esperada del viaje.

Primero se elaborará un preprocesamiento de datos, esta técnica consiste en preparar los datos para el modelado.

El preprocesamiento de datos es una etapa crítica en el desarrollo de modelos predictivos eficientes (Gelman, 2006).

Las operaciones que se realizan son;

- Detectar las variables numéricas y las categóricas.
- Identificar si en una variable numérica falta algún dato, en el caso de que falte algún dato se rellena con el valor de la media de esa misma variable.
- Detección y manejo de valores atípico de alguna variable.
- Se utiliza el rango intercuartílico (IQR) que consiste en identificar el Q1 y Q3 del conjunto de datos de la variable, posteriormente se restan (Q3 - Q1) con el objetivo de obtener una medida de dispersión. Luego se aplica la siguiente fórmula para determinar si un valor es atípico:
- $límite\ inferior = q1 - (1.5 * IQR)$
- $límite\ superior = q3 + (1.5 * IQR)$
- Si un valor está por debajo del Q1 - 1.5 veces el IQR calculado se considera atípico y si está por encima de Q3 + 1,5 veces el valor del IQR también. Los outliers se reemplazan por la media de los datos sin tener en cuenta éstos. En concreto existen 2 outliers.

- Rellenar espacios vacíos en variables categóricas utilizando el valor que más frecuente (moda) en esa misma variable. El modelo tiene 3 missing data.

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler

def data_preprocessing_pipeline(data):
    # Identificar características numéricas y categóricas
    numeric_features = data.select_dtypes(include=['float',
'int']).columns
    categorical_features =
data.select_dtypes(include=['object']).columns

    # Manejo de valores faltantes en características numéricas
    data[numeric_features] =
data[numeric_features].fillna(data[numeric_features].mean())

    # Detección y manejo de valores atípicos en características
numéricas usando IQR
    for feature in numeric_features:
        Q1 = data[feature].quantile(0.25)
        Q3 = data[feature].quantile(0.75)
        IQR = Q3 - Q1
        lower_bound = Q1 - (1.5 * IQR)
        upper_bound = Q3 + (1.5 * IQR)
        data[feature] = np.where((data[feature] < lower_bound) |
(data[feature] > upper_bound),
                                 data[feature].mean(),
data[feature])

    # Manejo de valores faltantes en características categóricas
    data[categorical_features] =
data[categorical_features].fillna(data[categorical_features].mode()
.iloc[0])

    return data
```

Figura 14. Entrenamiento del modelo predictivo de ML. Elaboración propia-

Tras el preprocesamiento se transforman las variables categóricas en numéricas y el siguiente paso es dividir el conjunto de datos y ejecutar el modelo de regresión Random Forest:

```
# Convirtir la hora de la reserva a una característica numérica
data["Time_of_Booking"] = data["Time_of_Booking"].map({"Afternoon":
0, "Evening": 1, "Morning": 2, "Night": 3})

# Convirtir el tipo de vehículo a una característica numérica
data["Vehicle_Type"] = data["Vehicle_Type"].map({"Premium": 1,
"Economy": 0})

El siguiente paso es dividir el conjunto de datos y ejecutar el
modelo de regresión Random Forest.

# Dividir los datos en entrenamiento y prueba
from sklearn.model_selection import train_test_split
x = np.array(data[["Number_of_Riders", "Number_of_Drivers",
"Vehicle_Type", "Time_of_Booking", "Expected_Ride_Duration"]])
y = np.array(data[["adjusted_ride_cost"]])

x_train, x_test, y_train, y_test = train_test_split(x,
                                                    y,
                                                    test_size=0.2,
                                                    random_state=42)

# Reestructurar y a un array 1D
y_train = y_train.ravel()
y_test = y_test.ravel()

# Entrenar un modelo de regresión Random Forest
from sklearn.ensemble import RandomForestRegressor
model = RandomForestRegressor()
model.fit(x_train, y_train)
```

Figura 15. Proceso de preprocesamiento de datos y entrenamiento del modelo de regresión Random Forest. Elaboración propia,

Esta parte del código consiste en importar desde la biblioteca Scikit-learn la función train_test_split, una función que nos permite entrenar y probar los datos divididos en dos variables, X y Y.

Primero se definen las variables X y Y:

- Las variables X o variables independientes, que está formada por "Number_of_Riders", "Number_of_Drivers", "Vehicle_Type", "Time_of_Booking" y "Expected_Ride_Duration".
- La variable Y o variable dependiente, conformada por la variable calculada anteriormente; adjusted_ride_cost.

Luego a la variable X y a la variable Y se le establece el entrenamiento y la prueba. El entrenamiento consiste en proporcionar al modelo un conjunto de datos, en nuestro caso los datos de las variables X e Y para que éste se adapte al conjunto de datos y poder obtener mejores resultados, en otras palabras, el modelo aprende a partir de estos datos. Por otro lado, el test que consiste en aquellos datos que no se han utilizado para la fase de entrenamiento ayudan a determinar cómo se comporta el modelo ante la entrada de nuevos datos. En este caso, el 20% de los datos se utilizan en la fase de test, por lo tanto, el resto de los datos se utilizan en la fase de entrenamiento.

Se convierte la estructura de datos a unidimensional en el caso de que sean bidimensionales para un mejor procesamiento de los datos. Por último, se importa la función RandomForestRegressor y se ejecuta el entrenamiento de la variable X e Y.

Tras incorporar el algoritmo, se puede ejecutar código que sirva cómo entrada de los valores de las diferentes variables para poder observar las diferencias en función de cómo varían las variables independientes el precio final cambia.

```
# Probar el modelo de ML usando algunos valores de entrada
def get_vehicle_type_numeric(vehicle_type):
    vehicle_type_mapping = {
        "Premium": 1,
        "Economy": 0
    }
    vehicle_type_numeric = vehicle_type_mapping.get(vehicle_type)
    return vehicle_type_numeric

def get_time_of_booking_numeric(time_of_booking):
    time_of_booking_mapping = {
        "Afternoon": 0,
        "Evening": 1,
        "Morning": 2,
        "Night": 3
    }
    time_of_booking_numeric =
time_of_booking_mapping.get(time_of_booking)
    return time_of_booking_numeric

# Realizar predicciones usando valores de entrada del usuario
def predict_price(number_of_riders, number_of_drivers,
vehicle_type, time_of_booking, Expected_Ride_Duration):
    vehicle_type_numeric = get_vehicle_type_numeric(vehicle_type)
    if vehicle_type_numeric is None:
        raise ValueError("Tipo de vehículo inválido")

    time_of_booking_numeric =
get_time_of_booking_numeric(time_of_booking)
    if time_of_booking_numeric is None:
        raise ValueError("Hora de reserva inválida")

    input_data = np.array([[number_of_riders, number_of_drivers,
vehicle_type_numeric, time_of_booking_numeric,
Expected_Ride_Duration]])
    predicted_price = model.predict(input_data)
    return predicted_price
```

Figura 16. Prueba del modelo de ML con valores de entrada. Elaboración propia.

Lo primero es mapear aquellas variables categóricas en valores numéricos que el código o modelo de ML pueda ejecutar:

- En el caso de que el coche sea “Economy” el código interpretará que el valor asignado es 0, mientras que, en el caso de ser “Premium” el valor es 1.

- De manera similar, se realiza el mapeo para el momento de la reserva: si la reserva se hace en "Afternoon", se asigna el valor 0; para "Evening", se asigna el valor 1; "Morning" se mapea con el valor 2; y "Night" con el valor 3.

La segunda parte de este código consiste en definir predict_price, en otras palabras, se obtiene la predicción del precio según las variables introducidas.

Esto se consigue mediante la detección de las variables mapeadas anteriormente, además, en el caso de que se introduzca una variable no identificada, por ejemplo en el caso de tipo de vehículo que sea “Premium” (1) o “Economy” (0), se detecte un error y se avisa mediante "Invalid vehicle type". Con la función input_data = np.array de la biblioteca NumPy se genera una matriz con 5 columnas, una por variable y una fila que contendrá los datos de cada columna

Posteriormente se utiliza la función model.predict(input_data), la parte del código que calcula el precio dinámico a partir de todas las variables introducidas.

3.1.5. Pruebas y R^2

El modelo está acabado, pero, ahora se necesita código que nos permita introducir cambios a las variables independientes para ver cómo varía el precio en función de las características del viaje.

```
predicted_price = predict_price(user_number_of_riders,
user_number_of_drivers, user_vehicle_type, user_time_of_booking,
Expected_Ride_Duration)
user_number_of_riders = 65
user_number_of_drivers = 30
user_vehicle_type = "Economy"
user_time_of_booking = "Afternoon"
Expected_Ride_Duration = 50
print("The predicted price for the ride is:", predicted_price)
```

Figura 17. Código plantilla para realizar pruebas. Elaboración propia.

- **Número de viajeros**: 80
- **Número de conductores**: 25
- **Tipo de vehículo**: Económico
- **Tiempo de reserva**: Tarde
- **Duración esperada del viaje**: 40
- **Precio predicho por el algoritmo de ML**: 464,5695.

Debido a que aumenta la demanda porque el número de pasajeros es mayor, además la oferta disminuye debido a una disminución de conductores el precio aumenta significativamente. Dado que la duración esperada disminuye el precio no aumenta tanto cómo si se hubiera mantenido en 50.

El tercer escenario consiste en disminuir significativamente la demanda y aumentar la oferta para observar el contraste entre un escenario con demanda alta y uno con baja. Además, el tiempo de viaje será por la noche, que disminuye el coste ya que no hay tanto tráfico.

- **Número de viajeros**: 30
- **Número de conductores**: 40
- **Tipo de vehículo**: Económico
- **Tiempo de reserva**: Noche
- **Duración esperada del viaje**: 40
- **Precio predicho por el algoritmo de ML**: 194.872.

El precio disminuye drásticamente más de la mitad.

El último escenario consiste en comparar cómo afecta el tipo de vehículo al precio manteniendo las mismas características o valores en las otras variables.

- **Número de viajeros**: 30
- **Número de conductores**: 40
- **Tipo de vehículo**: Premium
- **Tiempo de reserva**: Noche
- **Duración esperada del viaje**: 40
- **Precio predicho por el algoritmo de ML**: 267,3538.

El precio en el caso de que en el mismo tipo de viaje cambie la variable tipo de coche, modifica el precio en un aumento de 72,48.

Para finalizar con este ejemplo se ejecutará una parte del código que nos proporcione el nivel de fiabilidad del modelo.

```
#presición
from sklearn.metrics import r2_score

#predicción del modelo
y_pred = model.predict(x_test)

r2_score(y_test, y_pred)*100
```

Figura 18. Precisión del modelo. Elaboración propia.

En esta parte del modelo se calcula el R2 que como explica (Piepho, 2019) se define como la proporción de la suma de cuadrados corregida que explica el modelo, en otras palabras, mide el nivel de efectividad o confianza del modelo.

El resultado de R2 es de 86.49%, lo que indica que nuestro modelo explica aproximadamente el 86% de la variabilidad de la variable dependiente que en este caso es el precio, basándose en las variables independientes utilizadas. Por lo que podemos afirmar que el modelo es efectivo y confiable.

3.2. Predicción de precios de viviendas en Boston

3.2.1. Introducción

El mercado inmobiliario de Boston, con su continua atracción y análisis exhaustivo, es un tema que siempre capta la atención. Utilizando una herramienta avanzada, podemos investigar en profundidad los diversos factores que influyen en los precios de las propiedades en esta dinámica e histórica ciudad.

Por ello, el objetivo de este estudio consiste en determinar cómo varía el valor medio por metro cuadrado de las viviendas de Boston, utilizando un modelo de redes neuronales.

Una red neuronal artificial en inteligencia artificial es un modelo computacional inspirado en la estructura y funcionamiento del cerebro humano, diseñado para reconocer patrones y realizar tareas como clasificación y predicción mediante el aprendizaje a partir de datos (Ruiz, 2020).

3.2.2. Instalación de librerías

El primer paso es instalar todas aquellas librerías que se necesitan para que la red neural se ejecute correctamente y se pueda realizar el modelo de regresión efectivamente.

```
!pip install wandb

# Librerías estándar de Python
import random
from datetime import datetime

# Librerías para manipulación y análisis de datos
import pandas as pd
import numpy as np

# Librerías para aprendizaje automático y profundo
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.model_selection import train_test_split

# Métricas de evaluación
from sklearn.metrics import mean_squared_error as mse
from sklearn.metrics import mean_absolute_error as mae
from sklearn.metrics import r2_score

# Otras librerías
import wandb

# Desactivar advertencias
import warnings
warnings.filterwarnings('ignore')
```

Figura 19. Instalación librerías. Elaboración propia.

El módulo random se utiliza en aquellos modelos en los que se necesite generar números aleatorios y datetime se utiliza para manipular y operar con fechas, lo que será crucial para este modelo.

Pandas es una biblioteca que se utiliza para manipular y analizar datos y además es el instrumento que lee el archivo con los datos. Para realizar los cálculos complejos que requiere este tipo de modelos de ML se utiliza la librería Numpy.

También se importa TensorFlow, una biblioteca de código abierto desarrollada por Google para computación numérica y aprendizaje profundo de gran escala.

El aprendizaje profundo según los expertos es una técnica de aprendizaje automático que permite a los sistemas informáticos modelar representaciones de datos en múltiples niveles de abstracción, aprendiendo directamente de los datos sin necesidad de intervención humana para diseñar detectores de características específicas (LeCun, 2015). Además, utiliza Keras una API (métodos y reglas para interactuar con una biblioteca o software) de alto nivel dentro de TensorFlow que facilita la creación y entrenamiento de modelos de aprendizaje profundo.

Para crear y entrenar un modelo de red neuronal con Keras y TensorFlow, se utilizan varios componentes clave, que además determinarán los hiper parámetros del modelo.

- **Sequential** es necesario para construir un modelo secuencial. Un modelo secuencial en redes neuronales organiza las capas de manera lineal, permitiendo el flujo de datos desde la capa de entrada hasta la capa de salida sin bucles ni conexiones recurrentes (Goodfellow, 2016).
- **Dense** y **Dropout** añaden capas al modelo: Dense para capas totalmente conectadas y Dropout para prevenir el sobreajuste.
- **Adam** se usa como optimizador para ajustar los pesos del modelo.
- **EarlyStopping** detiene el entrenamiento si no hay mejora en una métrica monitoreada, evitando el sobreajuste y reduciendo el tiempo de entrenamiento.

El sobreajuste en machine learning según (Ying, 2019) se refiere a un fenómeno donde un modelo se ajusta demasiado bien a los datos de entrenamiento, capturando incluso el ruido y las peculiaridades específicas de esos datos. Como resultado, el modelo tiene un rendimiento excelente en el conjunto de entrenamiento, pero falla al generalizar a datos nuevos o no vistos, lo que se traduce en un bajo rendimiento en el conjunto de prueba.

El train_test_split divide los datos para realizar la fase de entrenamiento.

Desde la biblioteca sklearn.metrics se importa tres medidas herramientas de medición de rendimiento del modelo, son métricas que evalúan de forma cuantitativa la eficacia de predicción del modelo:

- **Mean_squared_error**: El error cuadrático medio se define como la expectativa de la desviación al cuadrado de un estimador con respecto a un parámetro verdadero (Schluchter, 2014). A menor valor menor error del modelo.
- **Mean_absolute_error**: El error absoluto medio es una medida utilizada para evaluar la precisión de un modelo de predicción, definida como la media de las diferencias absolutas entre los valores predichos y los valores observados. El MAE es más interpretable que otras medidas basadas en el valor absoluto, como el error cuadrático medio (Schutz, 1973).
- **r2_score**: Coeficiente determinación.

Con wandb lo que se consigue es un registro y análisis de las métricas de rendimiento en la fase de entrenamiento del modelo, más adelante se explica con mayor profundidad. Además se ignoran las advertencias para una salida limpia de los datos y una modelización más eficaz con la utilidad warnings.filterwarnings('ignore').

3.2.3. Preparación del modelo

Tras importar todas las librerías, métricas y funcionalidades necesarias para el modelo de red neuronal, lo que se necesita a continuación es generar un diccionario vacío. Un diccionario vacío se crea con el objetivo de tener un espacio en el que almacenar todos aquellos datos y métricas más importantes o en otras palabras un lugar donde registrar todos los experimentos del modelo.

```
prms_hist = {}
```

Luego, se definen qué datos se utilizarán en la fase de entrenamiento y cuales en la de test. También se fijan los hiper parámetros del modelo de red neuronal. Los hiper parámetros en una red neuronal son variables de alto nivel que configuran la estructura del modelo y el proceso de aprendizaje, y no se ajustan durante el entrenamiento (Makwe, 2020). En este caso, los hiperparámetros son los componentes claves mencionados anteriormente.

```
def nn_model(prms, X, y):
    X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=prms['validation_split'])

    # Definir el modelo
    model = Sequential()
    model.add(Dense(prms['neurons'], activation=prms['activation'],
input_shape=(13,)))
    model.add(Dropout(prms['dropout']))
    model.add(Dense(prms['neurons'],
activation=prms['activation']))
    model.add(Dropout(prms['dropout']))
    model.add(Dense(1))
```

Figura 20. Definición y entrenamiento del modelo de red neuronal. Elaboración propia.

- Se genera un modelo secuencial con model = Sequential()
- Model.add(Dense(prms['neurons'], activation=prms['activation'], input_shape=(13,))), esta función nos permite introducir el número de neuronas que el modelo utilizará y además el número de datos que se espera, 13.
- Si el modelo detecta sobreajuste "apagará" un determinado porcentaje de las neuronas con la función model.add(Dropout(prms['dropout'])).
- Para que el modelo sea aún más robusto y fiable añade otra capa Dense. Ya que, la primera capa sirve para detectar y aprender características iniciales y con una segunda capa aprender características de los datos más complejas. También añade otra capa Dropout, con el objetivo de evitar aún más el sobreajuste.
- Finalmente se crea una capa con una única salida (Dense(1)), ya que esta última capa es la que nos proporcionará el valor final del modelo de regresión.

El siguiente paso consiste en compilar el modelo, lo cual es crucial antes de entrenarlo. Este paso configura el modelo con la función de pérdida, las métricas de evaluación y el optimizador.

La compilación de este modelo consiste en registrar las diferencias entre el valor real y las predicciones con la función loss, luego estas variaciones servirán para calcular las métricas explicadas anteriormente. Estas diferencias se optimizan, si se optimizan las diferencias el modelo intentará reducirla y por lo tanto será más eficiente.

```
    # Compilar el modelo
    model.compile(loss=prms['loss'], metrics=prms['metrics'],
optimizer=Adam(lr=prms['learning_rate']))

    # Ajustar el modelo
    history = model.fit(X_train, y_train,
                        batch_size=prms['batch_size'],
                        epochs=prms['epochs'],
                        validation_split=prms['validation_split'],
                        verbose=0)

    # Realizar predicciones
    y_pred = model.predict(X_test)

    # Guardar precisión, recall y r2 en prms
    prms['r2'] = r2_score(y_test, y_pred)
    for met in history.history.keys():
        prms[met + 'h'] = history.history[met][-1]

    return prms, history, y_test, y_pred
```

Figura 21. Compilación, ajuste y evaluación del modelo. Elaboración propia.

Con la función model.dit se entrenan los datos de las variables independientes y las dependientes. La frecuencia de actualización de los datos (lotes de datos que el modelo analiza) se define con la función batch_size. El número de veces que el algoritmo de entrenamiento verá el conjunto completo de datos será con la función epochs. La fracción de los datos de entrenamiento que se utilizará como datos de validación, para analizar y evitar sobreajuste ('validation_split').

La verbosidad según (Loukas, 2021) en una red neuronal se refiere a la cantidad de información detallada que la red procesa durante su entrenamiento y funcionamiento, incluyendo la complejidad de su arquitectura y el número de parámetros involucrados, es de 0.

Se ejecutan las predicciones de la variable independiente sobre el test de las dependientes y se calcula el coeficiente de determinación (R2) midiendo las diferencias entre los valores reales de Y y las predicciones obtenidas.

El siguiente paso es la lectura de la base de datos. El nombre es HousingData.csv que contiene información sobre viviendas en Boston, con un total de 506 entradas y 14 columnas.

Las columnas contienen variables de interés que determinan la variación del valor de las viviendas, en función de estas variables se intenta cuantificar dicho valor con el propósito de obtener un precio más justo o adaptado. Las variables son las siguientes:

- **CRIM**: Tasa de criminalidad en la ciudad.
- **ZN**: Porcentaje de terrenos residenciales grandes.
- **INDUS**: Porcentaje de área ocupada por negocios industriales.
- **CHAS**: Indica si la propiedad está cerca del río Charles (1 = sí, 0 = no).
- **NOX**: Nivel de contaminación del aire.
- **RM**: Promedio de habitaciones por vivienda.
- **AGE**: Porcentaje de viviendas construidas antes de 1940.
- **DIS**: Distancia a los centros de empleo en Boston.
- **RAD**: Accesibilidad a carreteras principales.
- **TAX**: Tasa de impuestos a la propiedad.
- **PTRATIO**: Relación de alumnos por profesor en las escuelas de la ciudad.
- **B**: Medida que involucra la proporción de población afroamericana.
- **LSTAT**: Porcentaje de personas de bajos ingresos.
- **MEDV**: Valor medio de las viviendas.

```
boston = pd.read_csv('/content/HousingData.csv')
boston.dropna(inplace=True)
boston
```

Figura 22. Lectura base de datos. Elaboración propia.

Output:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4
5	0.02985	0.0	2.18	0.0	0.458	6.430	58.7	6.0622	3	222	18.7	394.12	5.21	28.7
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
499	0.17783	0.0	9.69	0.0	0.585	5.569	73.5	2.3999	6	391	19.2	395.77	15.10	17.5
500	0.22438	0.0	9.69	0.0	0.585	6.027	79.7	2.4982	6	391	19.2	396.90	14.33	16.8
502	0.04527	0.0	11.93	0.0	0.573	6.120	76.7	2.2875	1	273	21.0	396.90	9.08	20.6
503	0.06076	0.0	11.93	0.0	0.573	6.976	91.0	2.1675	1	273	21.0	396.90	5.64	23.9
504	0.10959	0.0	11.93	0.0	0.573	6.794	89.3	2.3889	1	273	21.0	393.45	6.48	22.0

Tabla 4. Vista previa del conjunto de datos. Elaboración propia.

3.2.4. Ejecución del modelo

Para continuar con el modelo se necesita crear una cuenta en la plataforma de Wandb, esta plataforma nos permite registrar y analizar los experimentos de la red neuronal.

Seguido introducimos el código en el software de programación, con los experimentos que se registran en Wandb:

```
wandb.login(key='XXXXX')

for _ in range(20):
    run_name = datetime.now().strftime("%y%m%d_%H%M%S")

    # Inicializar una nueva ejecución en Weights and Biases
    wandb.init(project="kaggle_regression_boston", name=run_name)

    # Separar las características (X) y la variable objetivo (y)
del conjunto de datos Boston
    X = boston.drop("MEDV", axis=1).values
    y = boston["MEDV"].values

    # Definir los hiperparámetros para la ejecución actual
seleccionando valores aleatorios de las opciones disponibles
    prms = {
        "neurons": random.choice([8, 16, 32, 64, 128, 256, 512]),
        "batch_size": random.choice([8, 16, 32, 64, 128, 256,
512]),
        "epochs": random.choice([8, 16, 32, 64, 128, 256, 512]),
        "validation_split": random.choice([0.1, 0.2, 0.3, 0.4,
0.5]),
        "dropout": random.choice([0.1, 0.2, 0.3, 0.4, 0.5]),
        "loss": random.choice(["mse", "mae"]),
        "metrics": random.choice(["mse", "mae"]),
        "learning_rate": random.choice([0.001, 0.01, 0.1, 0.2,
0.3]),
        "activation": random.choice(["relu", "sigmoid", "tanh"]),
        "optimizer": random.choice(["adam", "sgd", "rmsprop"])
    }
```

Figura 23. Login en Wandb y preparación del modelo. Elaboración propia.

Se realiza un bucle con 20 experimentos, cada uno con nombre propio y fecha de realización. Se separan las variables que configuran las características (X) del objetivo (Y).

- Variables X; Todas las variables menos MEDV (CRIM, ZN, INDUS, CHAS, NOX, RM, AGE, DIS, RAD, TAX, PTRATIO, B, y LSTAT.
- Variable Y; MEDV el valor medio de las viviendas ocupadas por sus propietarios.

Luego se selecciona un valor aleatorio de todos los hiperparámetros, con el objetivo de realizar experimentos lo más aleatorios posibles y abarcar un amplio rango de

configuraciones. Esto ayuda a explorar cómo diferentes combinaciones de hiperparámetros afectan el rendimiento del modelo, permitiendo encontrar la configuración óptima.

- "**neurons**": Número de neuronas. Selección aleatoria entre 8, 16, 32, 64, 128, 256 y 512.
- "**batch_size**": Tamaño del lote. Selección aleatoria entre 8, 16, 32, 64, 128, 256 y 512.
- "**epochs**": Número de épocas del modelo. Selección aleatoria entre 8, 16, 32, 64, 128, 256 y 512.
- "**validation_split**": Que porcentaje de datos se utilizan en la validación o test, por tanto el resto al entrenamiento. Slección aleatoria entre 0.1, 0.2, 0.3, 0.4, 0.5.
- "**dropout**": Porcentaje de desactivación de neuronas para evitar sobreajuste. Selección aleatoria entre 0.1, 0.2, 0.3, 0.4, 0.5.
- "**loss**": Cálculo de las pérdidas
- "**metrics**": Selección aleatoria entre el error cuadrático medio y el error absoluto medio.
- "**learning_rate**": Ratio de aprendizaje. Determina la magnitud de los cambios que se hacen en los pesos del modelo durante el proceso de optimización (Bengio, 2012). Selección aleatoria entre 0.001, 0.01, 0.1, 0.2 y0.3.
- "**activation**": La función de activación permite modificar valores e introducir valores de no linealidad (valores que no pueden representar su relación con una función lineal, son más complejos). Se selecciona aleatoriamente entre "relu", "sigmoid" y "tanh".
- **Relu** (Rectified Linear Unit): permite la entrada al modelo solo sí esta está entre un número mayor que 0. En el caso de ser menor la entrada es 0.
- **Sigmoid**: Convierte una entrada en un valor entre 0 y 1. Se utiliza la siguiente fórmula:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

- **Tanh** (Tangente hiperbólica): convierte la entrada en un valor entre -1 y 1, para introducir valores no lineales y que el modelo sea más complejo, exactamente la forma de la función es más curva. Utiliza la siguiente fórmula:

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

- "**optimizer**": Los optimizadores permiten ajustar los pesos de las variables para minimizar las pérdidas. Selección aleatoria entre "adam", "sgd" y "rmsprop".
- **SGD** (Stochastic Gradient Descent): actualiza los pesos utilizando un lote o subconjunto de datos aleatorios en cada iteración, en lugar de usar todo el conjunto de datos.
- **RMS** (Root Mean Square Propagation): mantiene una media móvil de los cuadrados de los gradientes (derivadas parciales que conforman la curva) recientes y divide el gradiente por la raíz cuadrada de esta media, ajustando la tasa de aprendizaje para cada peso.
- **Adam** (Adaptive Moment Estimation): Combina los dos anteriores.

Luego, se inicia el entrenamiento del modelo con los hiperparámetros establecidos, además se actualizan cada experimento y se registra en el diccionario.

```
prms, history, _, _ = nn_model(prms, X, y)

    wandb.config.update(prms)

    prms_hist[run_name] = prms


  # wandb log
    for i in range(len(history.history['loss'])):
        for met in history.history.keys():
            wandb.log({met: history.history[met][i]})

    # wandb finalizado
    wandb.finish()
```

Figura 24. Diccionario de registro, login en Wandb y finalización. Elaboración propia.

Una época según (Chollet, 2018) en el contexto del entrenamiento de redes neuronales se define como una presentación completa del conjunto de datos de entrenamiento al algoritmo de aprendizaje. En otras palabras, una época es el número de veces que el algoritmo analiza y estudia el conjunto de datos (separados en lotes en este caso) de entrenamiento en cada experimento.

El código registra las métricas por épocas (bucle externo) y además registra cada una de las métricas (bucle interno). Con el objetivo de tener un registro detallado por tiempo y unidad preciso.

Finalmente, cuando los 20 experimentos han sido ejecutados se detiene el modelo.

3.2.5. Análisis de los resultados

Tras finalizar el modelo, el software Wandb ha elaborado un gráfico resumen de cada experimento. Se observan los resultados según la elección aleatoria entre los hiper parámetros[1] y las tres métricas[2] establecidas.

Las líneas de color cálido (naranja/rojo) son los experimentos con una mayor ratio de determinación R^2, por otro lado, cuánto más gélido (morado/azul) peor ratio de determinación. Se indica en la columna de la derecha que determina el R^2.

El objetivo de este gráfico es ver de manera visual qué configuraciones de los hiper parámetros del modelo de red neuronal nos proporciona un mayor coeficiente de determinación. Para posteriormente ajustar estos hiper parámetros y obtener el mejor precio de las viviendas.

[1] Ver hiper parámetros y sus posibles configuraciones aleatorias.
[2] MSE, MAE y R^2

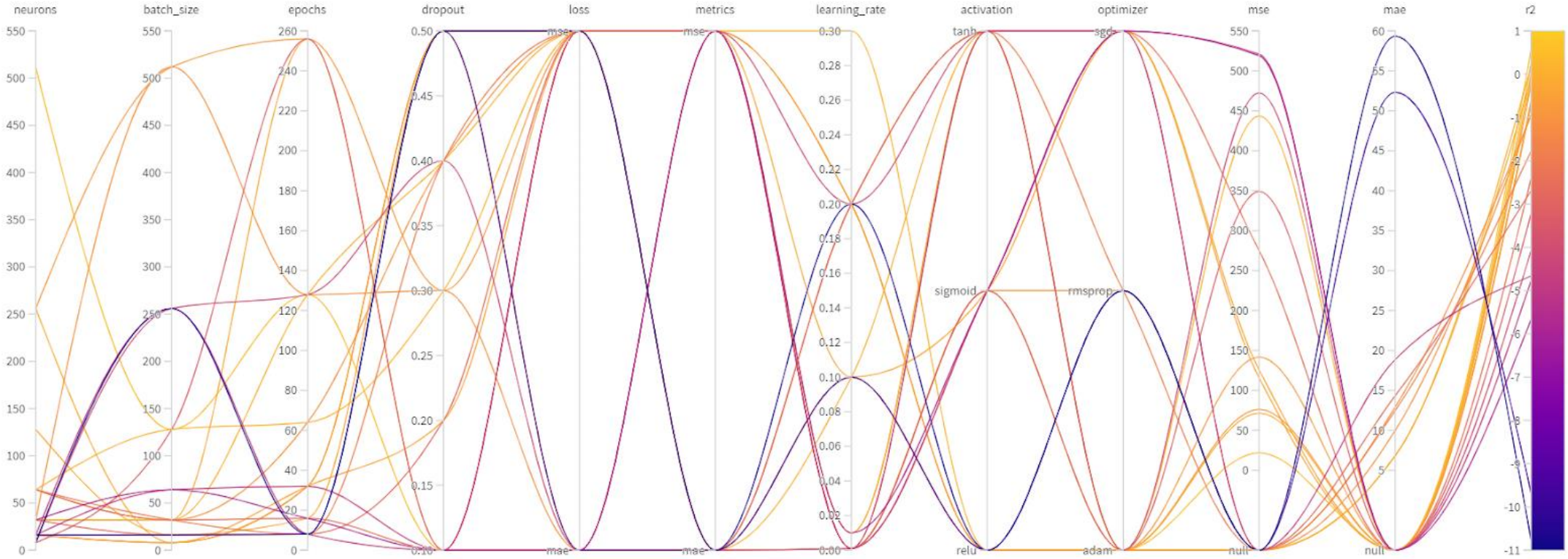

Figura 25. Registros Wandb. Elaboración propia.

De este gráfico podemos obtener las siguientes conclusiones;

- El número de neuronas óptimo oscila entre los 200 y 300. Valores demasiado altos pueden contener sobreajuste y valores demasiado bajos pueden no capturar suficiente información.
- El tamaño del lote debe ser entre 50 y 150. Tamaños mayores a 150 permiten más rapidez debido a que ajusta los pesos, pero esto supone que el modelo tarda más en encontrar los mejores pesos. Por otro lado, tamaños de lotes menores permiten mayor convergencia (el modelo aprende y encuentra los mejores pesos de manera más rápida).
- Un número demasiado elevado de épocas puede no mejorar el rendimiento del modelo y además provocar sobreajuste. Un número reducido pueden no ser suficientes para que el modelo sea eficiente. Por ello para este modelo el rango óptimo es entre 100 a 200 aproximadamente.
- Las tasas de dropout entre 0.2 y 0.3 están asociadas con mejores resultados. Demasiado dropout puede eliminar demasiada información útil, mientras que muy poco no proporciona suficiente regularización.
- Para la función loss y metrics es indiferente que métrica escoger, tienen indiferentemente líneas con buen rendimiento y con mal rendimiento.
- Las tasas de aprendizaje alrededor de 0.01 a 0.02 están asociadas con mejores resultados. Una tasa muy alta puede hacer que el modelo oscile y no converja, por lo tanto cambia con demasiada frecuencia los pesos y no mejora gradualmente. Mientras que, una tasa muy baja puede hacer que el modelo tarda demasiado en aprender.
- La mejor función de activación de la 'relu' ya que, introduce no linealidad en el modelo, lo que permite a la red neuronal aprender relaciones complejas en los datos.
- El mejor optimizador es ‘Adam’ probablemente porque combina los dos, 'sgd' y 'rmsprop'.

Además, podemos observar dos interacciones;

- Si el número de neuronas es medio o moderado y la tasa de aprendizaje es moderada, el modelo encuentra un buen equilibrio para aprender suficientes datos sin sobreajuste, y además, ajustando los pesos eficientemente.
- Tamaños de lote pequeños y un dropout moderado permite actualizar los pesos eficientes frecuentemente y evitar sobreajuste por apagar demasiadas neuronas.

El mejor resultado, o, en otras palabras, la mejor combinación de hiperparametros obtenido, el que proporciona un mayor R^2, es el siguiente;

- **'neurons'**: 256,
- **'batch_size'**: 32,
- **'epochs'**: 512,
- **'validation_split'**: 0.2,
- **'dropout'**: 0.2,
- **'loss'**: 'mse',
- **'metrics'**: 'mae',
- **'learning_rate'**: 0.01,
- **'activation'**: 'relu',
- **'optimizer'**: 'adam',

Resultados:

- **'r2'**: 0.82156545587455,
- **'lossh'**: 23.00958824157715,
- **'maeh'**: 3.7171719074249268,
- **'val_lossh'**: 17.585655212402344,
- **'val_maeh'**: 3.1363725662231445

El modelo con mejores resultados es una red con 256 neuronas, un tamaño de lote de 32 muestras y un total de 512 épocas, lo que implica 512 pasadas completas sobre el conjunto de datos de entrenamiento. Además, se destinó el 20% de los datos de entrenamiento para la validación del modelo, aplicando un dropout del 20% para regularizar y prevenir el sobreajuste. La función de pérdida seleccionada fue el error cuadrático medio (MSE), mientras que la métrica de evaluación elegida fue el error absoluto medio (MAE). La tasa de aprendizaje se estableció en 0.01, determinando los ajustes de los pesos del modelo en cada paso. Se utilizó la función de

activación ReLU (Rectified Linear Unit) y el optimizador elegido para ajustar los pesos del modelo fue Adam.

En cuanto a los resultados obtenidos, el coeficiente de determinación R^2 del modelo fue de 0.8215, lo que indica que el modelo explica aproximadamente el 82.15% de la variabilidad[3] de los datos de validación. En términos de pérdida, el MSE en los datos de entrenamiento fue de 23.01, mientras que el MAE en entrenamiento fue de 3.72. Para los datos de validación, el MSE fue de 17.59 y el MAE fue de 3.14, demostrando un rendimiento satisfactorio del modelo en ambos conjuntos de datos.

3.2.6. Conclusiones

La mejor combinación del modelo ha sido generada, el siguiente paso es realizar la regresión utilizando estos hiper parámetros, para obtener una predicción del MEDV, precio medio de las casas y compararlo con el valor real. Con el objetivo de observar si existen diferencias entre estos dos valores.

```
new_prms = {
                "neurons": best_prms['neurons']
                , "batch_size": best_prms['batch_size']
                , "epochs": best_prms['epochs']
                , "validation_split": best_prms['validation_split']
                , "dropout": best_prms['dropout']
                , "loss": best_prms['loss']
                , "metrics": best_prms['metrics']
                , "learning_rate": best_prms['learning_rate']
                , "activation": best_prms['activation']
                , "optimizer": best_prms['optimizer']
}
_, _, real, pred = nn_model(new_prms, X, y)
pd.DataFrame({"real": real, "pred": pred.flatten()}).head(10)
```

Figura 26. Impresión resultados y predicciones. Elaboración propia

Básicamente lo que se hace es determinar los valores de los hiper parámetros con los datos del mejor resultado, el comentado anteriormente. Luego se realiza el modelo de regresión,

[3] La variabilidad puede influir en la facilidad de entrenamiento y en la capacidad del modelo para producir soluciones deseables (Zhang, 2021)

las variables independientes son determinadas por los hiper parámetros. Por otro lado, la dependiente es el MEDV.

Se genera una tabla con visualización de los 10 primeros valores con dos columnas, el MEDV real y la predicción.

	real	predicción
0	10.4	10.032229
1	27.1	22.271235
2	18.8	16.553595
3	22.6	21.344164
4	7.0	8.903506
5	22.2	24.559198
6	23.1	24.119539
7	27.0	32.419521
8	18.1	15.676092
9	17.2	12.480157

Tabla 5. MEDV real vs. Predicción. Elaboración propia.

El resultado se mide en miles de dólares por metro cuadrado. Se observan diferencias en todos los valores, además algunas significativas de más de 5.000 $/m², cómo es el caso de la vivienda 1, 7 y 9. Esto puede sugerir dos conclusiones;

- El modelo, en ciertas viviendas, puede no ser lo suficientemente bueno ya que, un R^2 de 0. 0.8215 es un coeficiente aceptable, pero es posible que el modelo no esté capturando adecuadamente todas las complejidades y variabilidades de los datos. Por lo tanto, en algunas viviendas con ciertos parámetros complejos, la predicción puede no ser totalmente óptima.
- Los datos reales están mal medidos. Si el valor medio de las viviendas de Boston está mal medido, con mucho ruido y valores atípicos o inadecuados para su cálculo, existirán diferencias entre la predicción y el valor real. Por ello, el precio por metro cuadrado debe

ajustarse y alinearse a la predicción del modelo, para ser óptimo y que englobe de manera más eficiente las complejidades de la vivienda.

3.3. Comparación de modelos para predecir precios de vuelos

3.3.1. Introducción

El objetivo de este estudio es generar un modelo de ML que compare varios algoritmos de regresión para predecir los precios de vuelos de diferentes características cómo la aerolínea, escalas, tiempo etc… Se utiliza un conjunto de datos de vuelos con estas características. Los algoritmos evaluados incluyen RandomForestRegressor, GradientBoostingRegresso, k-nearest neighbors (KNN), LinearRegression, y Support Vector Regression (SVR), todos implementados utilizando la biblioteca scikit-learn (SK).

El resultado obtenido consiste en identificar qué modelo proporciona las predicciones más precisas y comprender las razones detrás de su rendimiento. Para ello, se utilizan tres métricas de evaluación: el coeficiente de determinación R^2, el Mean Absolute Error (MAE), y el Root Mean Squared Error (RMSE).

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestRegressor,
GradientBoostingRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.linear_model import LinearRegression
from sklearn.svm import SVR
import matplotlib.pyplot as plt
```

Figura 27. Importar librerías. Elaboración propia.

Se importan las librerías Pandas y Numpy para manipular, analizar y realizar cálculos computacionales

También se importa desde SK la función train_test_split que se utilizará para dividir los datos en datos de entrenamiento y datos de prueba.

StandardScaler ayuda a que todas las características del modelo tengan el mismo peso en la fase de entrenamiento. Esto es gracias a que estandariza los datos de cada característica, esto significa que elimina la media de los datos y se divide entre la varianza de la característica. El resultado obtenido es que ahora todos los valores de la característica están en la misma escala, útil en KNN. Fórmula:

$$Z = \frac{X - \mu}{\sigma}$$

En este modelo volvemos a utilizar el RandomForestRegressor, explicada en el supuesto de la aplicación DASH.

Utilizaremos un nuevo algoritmo llamado GradientBoostingRegressor, la idea clave detrás del GBR es minimizar la función de pérdida usando un enfoque de descenso de gradiente (Friedman, 2001).

KNeighborsRegressor es una regresión que utiliza como base el algoritmo de KNN. Este algoritmo utiliza qué tipo de categoría es el dato a predecir a través del tipo de dato más cercano.

LinearRegression es una regresión lineal común.

SVR busca encontrar una línea en un hiperplano separando de la manera más eficiente dos clases de conjunto de datos. Esta separación consiste en mantener distancia definida entre los puntos que representan cada dato, este criterio se conoce cómo epsilon (ε).

Se importan metrics para evaluar rendimientos y plt para la elaboración de gráficos.

3.3.2. Carga y manipulación de datos

Le indicamos al modelo leer el conjunto de datos que se utiliza. Se trata de vuelos con diferentes características, las características se agrupan por columnas. Con estos datos realizaremos la comparación de métricas y coeficiente de determinación del modelo.

```
# Cargar datos
flying_data = pd.read_excel('/content/Flying_Data.xlsx')
flying_data.dropna(inplace=True)
```

Figura 28. Cargar la base de datos. Elaboración propia.

Descripción de las columnas:

- **Airline**: Nombre de la aerolínea.
- **Date_of_Journey**: Fecha del viaje.
- **Source**: Ciudad de origen del vuelo.
- **Destination**: Ciudad de destino del vuelo.
- **Route**: Ruta del vuelo, incluyendo paradas intermedias.
- **Dep_Time**: Hora de salida del vuelo.
- **Arrival_Time**: Hora de llegada del vuelo.
- **Duration**: Duración del vuelo.
- **Total_Stops**: Número total de paradas intermedias.
- **Additional_Info**: Información adicional sobre el vuelo.
- **Price**: Precio del vuelo.

En el modelo, para convertir las columnas que conforman las características a tipos de datos se utiliza:

```
def change_into_datetime(col):
    flying_data[col] = pd.to_datetime(flying_data[col])
for col in ['Date_of_Journey', 'Dep_Time', 'Arrival_Time']:
    change_into_datetime(col)
```

Figura 29. Convertir columnas a datetime. Elaboración propia.

Estos datos por tipos constituyen los datos de entrenamiento.

Se extrae el mes y el día ya que el año es el mismo (2019) y se utilizará para profundizar aún más en los resultados, ya que se podrá trabajar sobre variaciones diarias o mensuales. Cuando los datos han sido extraídos, se elimina la columna y formarán parte del conjunto de datos de entrenamiento:

```
flying_data['journey_day'] = flying_data['Date_of_Journey'].dt.day
flying_data['journey_month'] =
flying_data['Date_of_Journey'].dt.month
flying_data.drop('Date_of_Journey', axis=1, inplace=True)
```

Figura 30. Extraer día y mes del viaje. Elaboración propia.

De manera parecida pasa con las horas y los minutos, se extraen, se elimina la columna y se utilizan cómo datos de entrenamiento:

```
# Extraer horas y minutos
def extract_hour(df, col):
    df[col + '_hour'] = df[col].dt.hour
def extract_min(df, col):
    df[col + '_minute'] = df[col].dt.minute
for col in ['Dep_Time', 'Arrival_Time']:
    extract_hour(flying_data, col)
    extract_min(flying_data, col)
    flying_data.drop(col, axis=1, inplace=True)
```

Figura 31. Extraer horas y minutos. Elaboración propia.

Para extraer la duración del vuelo utiliza:

```
flying_data['Duration'] = flying_data['Duration'].apply(lambda x: x
if 'h' in x else '0h ' + x)
flying_data['Duration'] = flying_data['Duration'].apply(lambda x: x
if 'm' in x else x + ' 0m')
flying_data['Duration_hours'] =
flying_data['Duration'].apply(lambda x: int(x.split('h')[0]))
flying_data['Duration_mins'] = flying_data['Duration'].apply(lambda
x: int(x.split('m')[0].split()[-1]))
flying_data.drop('Duration', axis=1, inplace=True)
```

Figura 32. Extraer duración. Elaboración propia.

Busca la letra 'h' para identificar el número de horas, en el caso de no encontrar una 'h' añade un '0h'. Por otro lado, busca la letra 'm' para encontrar los minutos y en el caso de no encontrarlo añade '0m'. Todo ello con el objetivo de fijar un formato único, lo que ayudará posteriormente a la correcta lectura de los datos. Finalmente, estos datos se utilizan cómo datos de entrenamiento.

El siguiente paso consiste en separar las variables categóricas y numéricas. Se utiliza la cadena de texto (str) que significa transformar los valores de esas variables para que sean tratados como cadenas de texto en lugar de otro tipo de dato como números o fechas.

```
cat_col = ['Airline', 'Source', 'Destination', 'Route',
'Total_Stops', 'Additional_Info']
flying_data[cat_col] = flying_data[cat_col].astype(str)
```

Figura 33. Separar características categóricas y numéricas. Elaboración propia.

La columna de ruta puede tener hasta 5 aeropuertos distintos, por ello se extraen los datos, creando 5 columnas separadas y cada aerolínea es una nueva cadena de texto. Además, los valores nulos se rellenan cómo 'None'. Estas columnas forman parte del conjunto de datos de entrenamiento.

```
flying_data['Route_1'] = flying_data['Route'].str.split('→').str[0]
flying_data['Route_2'] = flying_data['Route'].str.split('→').str[1]
flying_data['Route_3'] = flying_data['Route'].str.split('→').str[2]
flying_data['Route_4'] = flying_data['Route'].str.split('→').str[3]
flying_data['Route_5'] = flying_data['Route'].str.split('→').str[4]
flying_data.drop('Route', axis=1, inplace=True)
flying_data.fillna('None', inplace=True)
```

Figura 34. Procesar 'Route' en múltiples columnas. Elaboración propia.

Para un mejor rendimiento del modelo y una lectura de datos más eficaz, se utiliza la técnica one hot-encoding. One-hot encoding convierte cada etiqueta categórica de una columna de variable categórica en una nueva columna binaria (0 o 1).

Ejemplo:

Airline
IndiGo
Air India
Jet Airways

Tabla 6. Columna Airline original. Elaboración propia.

Airline_IndiGo	Airline_Air_India	Airline_Jet_Airways
1	0	0
0	1	0
0	0	1

Tabla 7. Transformación tras One-Hot Encoding. Elaboración propia.

Podemos apreciar cómo en el caso de que en una celda de la columna original de Airline la variable IndiGo está presente, en la celda correspondiente de la columna transformada se le da un valor de 1.

```
flying_data = pd.get_dummies(flying_data, columns=['Airline',
'Source', 'Destination', 'Total_Stops', 'Additional_Info',
                                                    'Route_1',
'Route_2', 'Route_3', 'Route_4', 'Route_5'], drop_first=True)
```

Figura 35. One-hot encoding para todas las variables categóricas. Elaboración propia.

Los valores atípicos del precio, que son aquellos valores no mayores de 40.000 se cambian por el valor de la mediana. Consiguiendo así que aquellos valores que no consideramos representativos no afecten al rendimiento y eficacia.

Se escogen 40.000 ya que los datos a partir de esta cantidad son muy raros y representan un pequeño porcentaje del total. Estos valores están muy por encima de la mayoría de los precios de los vuelos.

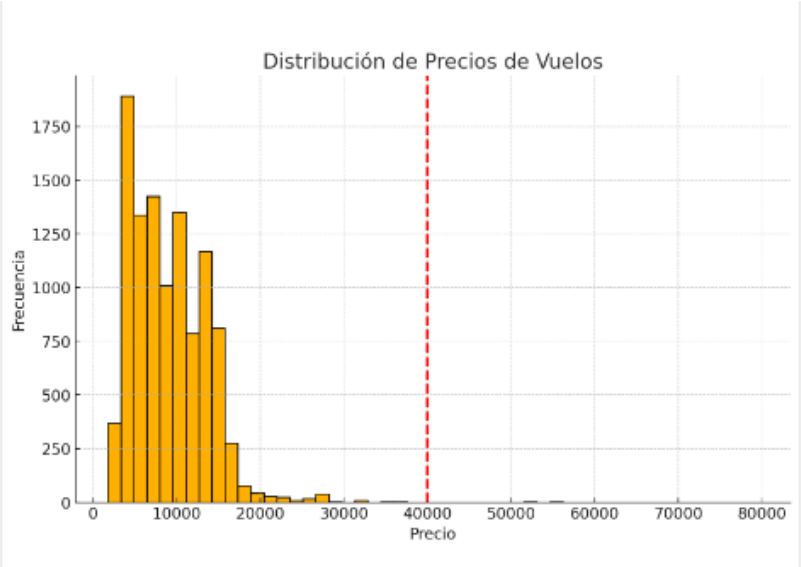

Figura 36. Distribución de los precios. Elaboración propia.

```
flying_data['Price'] = np.where(flying_data['Price'] >= 40000,
flying_data['Price'].median(), flying_data['Price'])
```

Figura 37. Manejo de outliers. Elaboración propia.

3.3.3. Modelos de predicción

En este apartado se preparan los diferentes modelos de predicción y se ejecutan.

Primero, se separan las variables X e Y. La variable objetivo (Y) es el precio, ya que es lo que intentamos predecir a partir de unas variables que caracterizan el modelo (X) que son todas aquellas columnas que hemos obtenido tras las transformaciones.

En los modelos KNN y SVR se necesita normalizar los datos, ya que estos son sensibles a las escalas. Si no se realiza este paso, aquellos valores con mayor escala dominarán[4] 1 a los que menor escala tiene, en el caso del modelo KNN y una convergencia[5] 2 menos eficiente en el caso de el SVR.

[4] Dominancia: se refiere a la influencia o importancia relativa de características específicas en la capacidad predictiva de un modelo.

[5] Converger: proceso en el cual los parámetros del modelo se ajustan para minimizar la función de pérdida o maximizar la precisión del modelo.

```
# Separar datos en X e y
X = flying_data.drop('Price', axis=1)
y = flying_data['Price']

# Normalizar las características para KNN y SVR
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

Figura 38. Separación modelos y normalización. Elaboración propia.

Se dividen los datos un 80% en datos de entrenamiento y un 20% de datos de prueba. Lo mismo con los datos estandarizados. Ambos casos con una aleatoriedad controlada de 42. Posteriormente, se ejecuta la fase de entrenamiento y prueba para realizar predicciones. Además, calculamos las métricas, que en este caso son el R^2, el MAE y el MSE.

```
# DividIr los datos en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
X_train_scaled, X_test_scaled, _, _ = train_test_split(X_scaled, y,
test_size=0.2, random_state=42)

# Función para entrenar y evaluar un modelo
def evaluate_model(model, X_train, X_test, y_train, y_test):
    model.fit(X_train, y_train)
    predictions = model.predict(X_test)
    r2 = metrics.r2_score(y_test, predictions)
    mae = metrics.mean_absolute_error(y_test, predictions)
    rmse = np.sqrt(metrics.mean_squared_error(y_test, predictions))
    return r2, mae, rmse
```

Figura 39. División de datos, entrenamiento y evaluación. Elaboración propia.

Las métricas quedan registradas en unas listas:

```
model_names = ['RandomForest', 'KNN', 'LinearRegression', 'SVR',
'GradientBoosting']
r2_scores = []
mae_scores = []
rmse_scores = []
```

Figura 40. Listas para guardar las métricas. Elaboración propia.

El siguiente paso consiste en la evaluación exhaustiva de los diferentes modelos de regresión, proporcionando los resultados de las métricas para determinar cuál modelo tiene un mejor rendimiento predictivo sobre los datos.

```
# Evaluar RandomForestRegressor
rf_model = RandomForestRegressor()
rf_r2, rf_mae, rf_rmse = evaluate_model(rf_model, X_train, X_test,
y_train, y_test)
r2_scores.append(rf_r2)
mae_scores.append(rf_mae)
rmse_scores.append(rf_rmse)

# Evaluar KNeighborsRegressor
knn_model = KNeighborsRegressor(n_neighbors=5)
knn_r2, knn_mae, knn_rmse = evaluate_model(knn_model,
X_train_scaled, X_test_scaled, y_train, y_test)
r2_scores.append(knn_r2)
mae_scores.append(knn_mae)
rmse_scores.append(knn_rmse)

# Evaluar LinearRegression
lr_model = LinearRegression()
lr_r2, lr_mae, lr_rmse = evaluate_model(lr_model, X_train, X_test,
y_train, y_test)
r2_scores.append(lr_r2)
mae_scores.append(lr_mae)
rmse_scores.append(lr_rmse)

# Evaluar SVR
svr_model = SVR()
svr_r2, svr_mae, svr_rmse = evaluate_model(svr_model,
X_train_scaled, X_test_scaled, y_train, y_test)
r2_scores.append(svr_r2)
mae_scores.append(svr_mae)
rmse_scores.append(svr_rmse)

# Evaluar GradientBoostingRegressor
gbr_model = GradientBoostingRegressor()
gbr_r2, gbr_mae, gbr_rmse = evaluate_model(gbr_model, X_train,
X_test, y_train, y_test)
r2_scores.append(gbr_r2)
mae_scores.append(gbr_mae)
rmse_scores.append(gbr_rmse)
```

Figura 41. Evaluación de los modelos. Elaboración propia.

3.3.4. Comparación modelos

En este apartado se comparará los resultados de la eficiencia de los diferentes modelos y las métricas de errores obtenidas. Se determinará qué modelo se adapta mejor a este tipo de problemas.

3.3.4.1. Comparación por coeficiente R^2

Este coeficiente mide la eficiencia del modelo para predecir la variabilidad de la variable dependiente utilizando las variables independientes incluidas en el modelo de regresión. En este caso, la variabilidad se refiere a que tan diferentes son las variaciones de los precios entre cada vuelo. Por lo tanto, podemos afirmar que el R^2 mide la eficiencia de predicción del modelo.

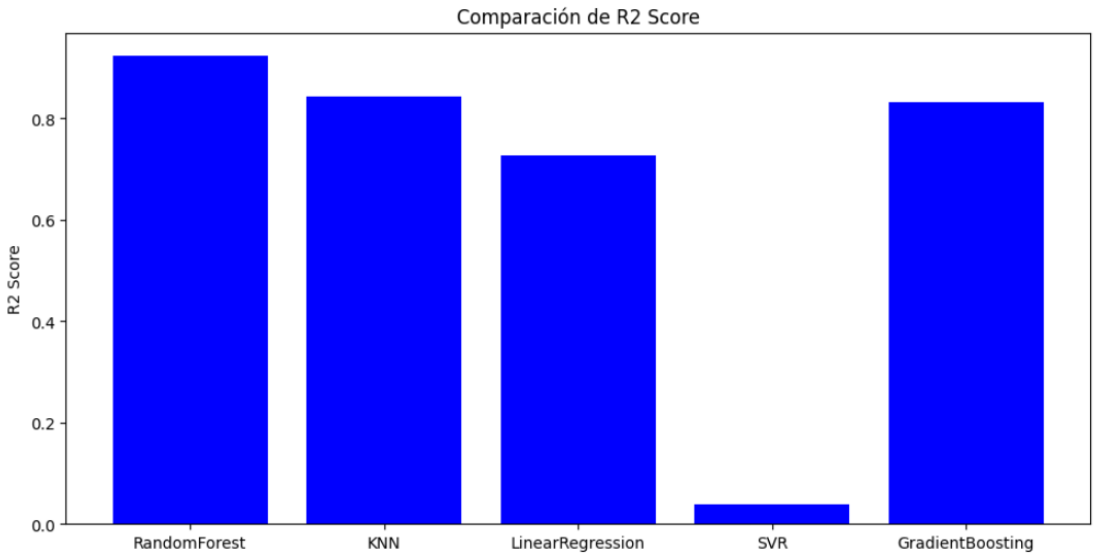

Figura 42. Comparación de R2. Elaboración propia.

- **RandomForest**: 0.9225765110622934
- **KNN**: 0.8422613607617444
- **LinearRegression**: 0.7278653165104381
- **SVR**: 0.039008092200514266
- **GradientBoosting**: 0.8321657295621426

El modelo que mejor se ajusta a este tipo es el RandomForest con un 92,26% de capacidad de eficiencia en la predicción, es el más fiable. Esto es debido a que este tipo de modelos tienen una gran capacidad capacidad para manejar relaciones no lineales y complejas en los datos.

Una relación no lineal entre dos variables se define como una situación en la que los cambios en una variable no resultan en cambios proporcionales en otra variable. En otras palabras, la relación entre las variables no se puede describir con una línea recta y puede implicar una relación cuadrática, exponencial u otra forma de curva (Grambsch, 1991).

Las relaciones complejas se refieren a interacciones entre múltiples variables que no pueden ser descritas de manera simple o directa (Munro, 2003).

En nuestro caso estas son las razones que explican por qué el conjunto de datos tiene características no lineales y complejas;

- **Interacciones**: Las diferentes características como la aerolínea, la ruta, la cantidad de escalas y la duración del vuelo pueden interactuar de maneras complejas. No existe un precio fijo, el precio depende de cada factor.
- **Efectos multiplicativos**: además si combinamos estas interacciones con escenarios de alta demanda, el precio aumenta o disminuye en temporada baja.
- **Temporalidad**: los precios de los vuelos tienen una tendencia estacionaria. Por ejemplo, en épocas de fiestas cómo navidad o verano el precio aumenta. Además, todos los fines de semana suelen aumentar debido al aumento de demanda. Por lo tanto, los precios de los vuelos tienen tendencia no lineal.
- **Competencia entre aerolíneas**: Otro ejemplo de no linealidad reside en que las aerolíneas disminuyen o aumentan los precios en función de la competencia.
- **Sensibilidad al Precio**: La elasticidad de la demanda puede variar dependiendo de factores como la disponibilidad de alternativas, la urgencia del viaje, y la sensibilidad al precio de diferentes segmentos de clientes.
- **Heterogeneidad**[6] **de los datos**: los datos utilizados para realizar el modelo tienen componentes categóricos y numéricos, generando relaciones complejas.

[6] Heterogeneidad: La heterogeneidad en los datos, que incluye componentes categóricos y numéricos, genera relaciones complejas y presenta desafíos significativos para el análisis y la interpretación de los modelos (Nunes, 2020).

Un modelo de árboles aleatorios crea múltiples árboles de decisión que representan nodos, cada árbol elige los mejores datos para elaborar una predicción. Posteriormente, se realiza el promedio de los resultados y obtenemos un único resultado.

Por ello, el modelo consigue realizar predicciones locales (en cada árbol) y capturar los patrones no lineales. Además, cada árbol tiene un subconjunto de datos aleatorio, lo que le añade variabilidad al modelo y contemplar las relaciones complejas.

Los modelos KNN y GradientBoosting son modelos que son óptimos para este tipo de problemas, con un R^2 de 0,84 y 0,83 respectivamente.

Un modelo KNN predice el valor del resultado basándose en los datos más cercanos, lo que permite obtener resultados locales y capturar las relaciones no lineales. Además, dado que solo tiene en cuenta los datos 'vecinos' captura de manera eficiente la complejidad de las características de los datos.

En el caso de GradientBoosting, lo que hace es combinar modelos denominados débiles, cómo por ejemplo árboles simples. Además, mejora cada modelo, por ello, en este modelo cada modelo es mejor que el anterior. Esta razón es la que permite al modelo contemplar de manera eficiente la complejidad y la no linealidad del conjunto de datos.

La regresión lineal simple tiene un R^2 de 72,79%. Es un modelo que se puede usar pero no tan recomendable. La base de una regresión lineal es la linealidad de los datos, por consiguiente, es un modelo que ante un componente fuerte de no linealidad cómo es el caso de este conjunto de datos, no puede elaborar predicciones de manera tan eficiente. Además, carece de herramientas potentes que puedan capturar interacciones complejas. Por tanto, dadas las limitaciones que tiene este tipo de modelo, no es un modelo eficiente para determinar precios de vuelos.

Por último, el modelo SVR tiene un valor de 0,03. Este modelo definitivamente no se puede usar para realizar predicciones de vuelos. Pero, esto es debido a que este modelo es muy sensible a la elección de los hiper parámetros. En el código no están escritos explícitamente los hiper parámetros por ello, el modelo utiliza los hiper parámetros predeterminados;

- **C**: 1.0
- **epsilon**: 0.1
- **kernel**: 'rbf' (Radial Basis Function)

- **gamma**: 'scale'

```
random_search = RandomizedSearchCV(SVR(), param_distributions,
n_iter=10, cv=5, scoring='r2', n_jobs=-1, random_state=42)

# Ajustar el modelo a los datos
random_search.fit(X_train_scaled, y_train)

# Obtener el mejor modelo
best_svr = random_search.best_estimator_

# Evaluar el mejor modelo SVR
svr_r2, svr_mae, svr_rmse = evaluate_model(best_svr,
X_train_scaled, X_test_scaled, y_train, y_test)
```

Figura 43. Búsqueda RandomizedSearchCV para el modelo SVR. Elaboración propia.

Nuevos hiper parámetros:

- **kernel**: 'linear'
- **gamma**: 'scale'
- **epsilon**: 0.1
- **C**: 100
- **Resultado R^2:** 0.6927.

Podemos observar que tras una simple búsqueda aleatoria de mejores parámetros se ha aumentado de 0,03 a 0,6927 de eficiencia del modelo. Por lo tanto la clave de un modelo SVR reside en realizar una búsqueda exhaustiva de los mejores hiper parámetros ya que, este modelo tiene potencial de ser un modelo muy eficiente, al nivel del RandomForest, para este tipo de problemas.

3.3.4.2. Comparación MAE

El mean absolute error (MAE) mide el promedio de los errores en valor absoluto. Por lo tanto, el MAE mide la magnitud del error en el modelo, a menor MAE menor error absoluto entre el precio real y el precio predicho tiene el modelo. Cabe mencionar que el MAE no distingue entre error en diferencia positiva o negativa. Además, mide el error en las mismas unidades que las variables, en este caso unidades monetarias ($).

$$MAE = \frac{1}{n}\sum_{i=1}^{n} | yi \ - \hat{y}i \ |$$

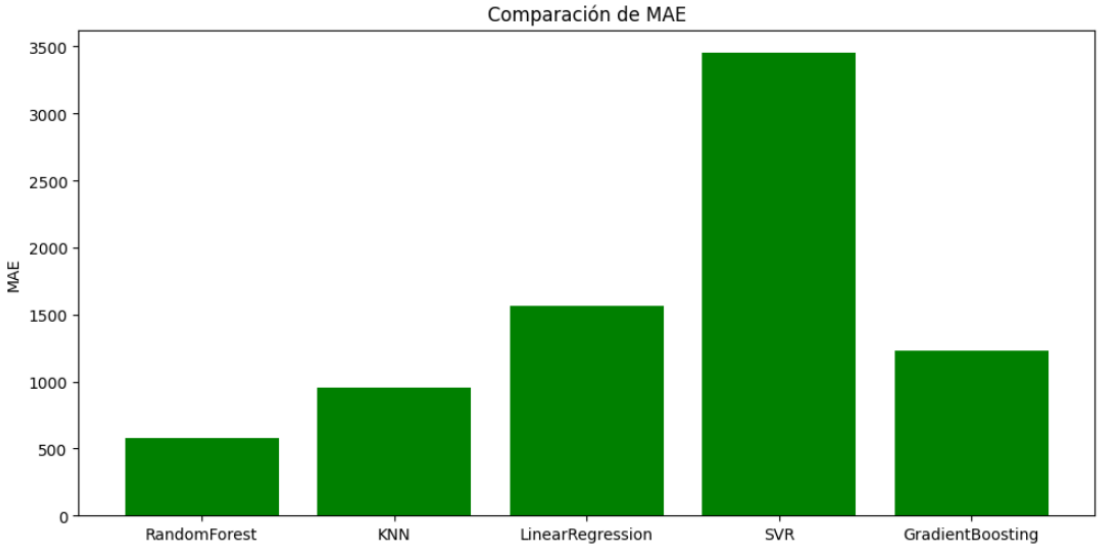

Figura 44. Resultados MAE. Elaboración propia.

- **RandomForest**: 579.5676470129464
- **KNN**: 954.7636874122602
- **LinearRegression**: 1566.6046286056878
- **SVR original**: 3448.408652541582
- **GradientBoosting**: 1227.459518486023

El modelo con menor error absoluto entre el precio real y la predicción es el RandomForest, con un valor de 579,86$.

En esta métrica si vemos una diferencia significativa de 272,73$ entre el modelo KNN y el modelo de Gradient Boosting. Lo que sugiere que en cuanto a error absoluto el KNN es ligeramente mejor que el Gradient Boosting.

La regresión lineal, por lo explicado anteriormente, tiene un error más elevado que el resto de 1566,6 $, por lo tanto de entre los modelos es el que más error absoluto nos proporciona.

A priori, el modelo SVR es el peor modelo con un error de 3448,4 $, lo que nos indica que este modelo no es ni siquiera usable. Pero, tras el ajuste simple de lo hiper parámetros el nuevo resultado del MAE es de 1512,16 $. Un resultado ligeramente mejor que el de la regresión lineal simple, lo que refuerza la teoría de que este modelo, ante una profunda búsqueda de los

mejores hiper parámetros para cada tipo de problema, tiene potencial de ser un modelo muy eficiente para predecir precios u incluso otros factores de características similares.

3.3.4.3. Comparación RMSE

La última métrica por comparar es el Root Mean Squared Error, que mide el error entre la predicción y el precio real, pero con la particularidad de que se calcula cómo la raíz cuadrada de la diferencia de los promedios elevados al cuadrado:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(yi - \hat{y}i)^2}$$

Esta métrica al igual que el MAE se expresa con las mismas unidades que el valor de los datos de la variable objetivo.

La diferencia entre el RMSE y el MAE radica principalmente en los errores denominados grandes. En el RMSE un error de gran magnitud penaliza más o en otras palabras contribuye a que el error del modelo según esta métrica sea mayor. Por otro lado, en el MAE todos los errores tienen el mismo peso.

La combinación de estas dos métricas nos proporciona información útil frente a modelos con comportamientos de error diferentes, permitiéndonos identificar tanto errores sistemáticos como grandes desviaciones ocasionales.

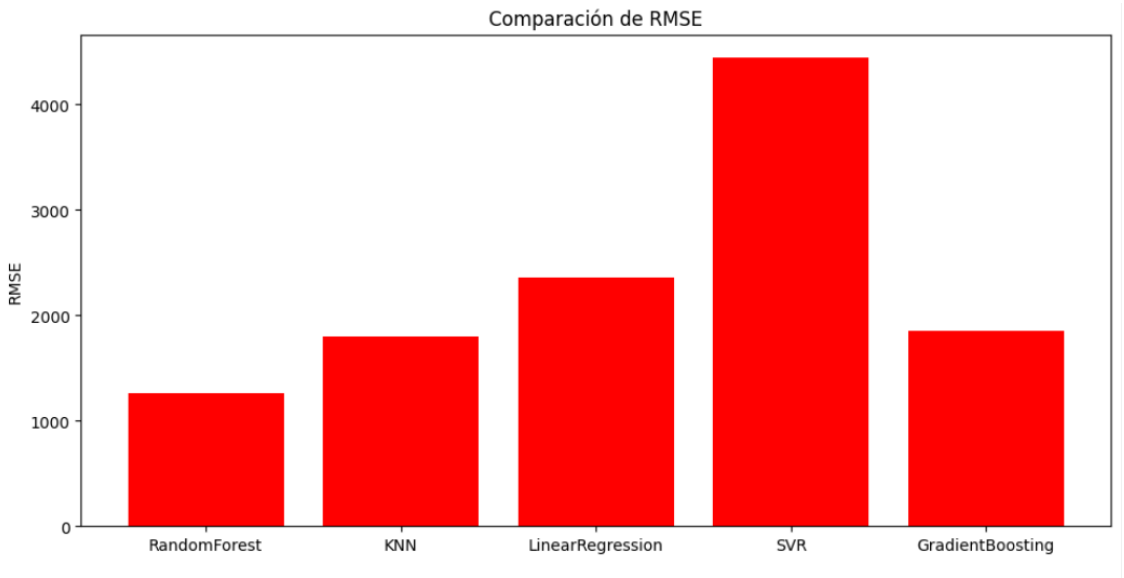

Figura 45. Resultados RMSE. Elaboración propia.

- **RandomForest**: 1259.5043688243195
- **KNN**: 1797.3456070713762
- **LinearRegression**: 2360.7744569625615
- **SVR original**: 4436.315401033345
- **GradientBoosting**: 1853.718134620165
- **SVR transformado**: 2508.4001230803983

Dado que el RMSE eleva al cuadrado las diferencias antes de promediarlos, en todos los modelos el MAE es significativamente menor que el RMSE, lo que sugiere que en general, los modelos son sensibles a errores de gran magnitud.

Los resultados en cuanto a 'ranking' son parecidos a los del MAE.

El algoritmo con menor error cuadrado es el RandomForest, lo que indica que este modelo maneja mejor los outliers y los errores grandes.

Le sigue KNN y GradientBoosting, que también tienen un RMSE bastante bajo en comparación con otros modelos, lo que significa que manejan bastante bien los errores, pero hay algunos errores grandes también. LinearRegression y el SVR original tienen un RMSE más alto, lo que significa que no son tan eficientes para manejar valores atípicos y grandes errores.

El SVR transformado, aunque mejora considerablemente que el original, sigue teniendo un RMSE más alto que los modelos RandomForest, KNN y GradientBoosting, por lo que sigue siendo sensible a grandes errores, aunque esto puede mejorarse al ajustar los hiper parámetros.

3.3.5. Conclusión de la comparación

El mejor modelo para este tipo de problemas definitivamente es el RandomForest, ganando en las tres métricas. Los modelos GradientBoosting y KNN son peores pero usables si creemos que se pueden adaptar más o menos a nuestras necesidades, además, con puntuaciones similares. La regresión lineal no es un modelo aconsejable para predecir precios de vuelos ya que los otros son más eficientes, este queda cómo un modelo demasiado simplista y no consigue abarcar las particularidades del conjunto de datos. Por último, cómo extra, el SVR aunque tiene un el por coeficiente de determinación y un MAE y RMSE mediocre, ante una búsqueda exhaustiva de hiperparametros tiene potencial de superar al GradientBoosting y al KNN, incluso de asemejarse al RandomForest.

4. Conclusiones

Tras estudiar todo el contenido de este Trabajo de Fin de Grado, podemos afirmar que los algoritmos de ML en el mundo empresarial y vida cotidiana representan un gran avance y fenómeno en la toma de decisiones y estrategias.

Los algoritmos de regresión permiten predecir distintas variables cómo la estudiada en el trabajo, el precio.

El ajuste del precio permite a las empresas cómo DASH, especializada en el sector del transporte de personas, identificar dependiendo de cada característica del transporte, cual es el precio óptimo o más eficiente. Esto permite múltiples ventajas cómo colocar de manera más eficiente la relación conductor/a – pasajeros o incluso, de manera menos directa, ayudar a emitir menos CO_2 por evitar transportes innecesarios.

Este trabajo ha demostrado que los algoritmos de ML pueden ser herramientas valiosas para predecir valores inmobiliarios. Permite realizar estimaciones más adaptadas a las condiciones del inmueble, generando fiabilidad en la decisión de compra de los clientes potenciales. También permiten identificar zonas urbanas sobrevaloradas e infravaloradas por lo que se pueden tomar decisiones más acertadas en cuanto a planificación urbana, desarrollo de infraestructura y políticas de vivienda.

En el TFG se ha realizado una investigación de qué algoritmos son eficientes para predecir los precios dinámicos ya que, tras el estudio de una estrategia de fijación de precios dinámicos de vuelos, un sector que depende de esta estrategia, podemos concluir que uno de los mejores algoritmos para predecir pecios dinámicos es el conocido RandomForest. Los algoritmos GradientBoosting y KNN son algoritmos potentes y recomendables. La regresión lineal simple no es muy recomendable ya que es demasiado simplista. Lo curioso es que el modelo SVR es muy sensible a encontrar la combinación de los mejores hiperparámetros, si se realiza una búsqueda exhaustiva de éstos el modelo tiene potencial de ser muy recomendable.

Un problema encontrado en este trabajo es que, aunque se ha realizado búsquedas profundas y comparativas, se depende de base de datos encontradas en internet, dependemos de fiarnos de la calidad y cantidad de los datos, esto en una empresa real no ocurre. Otro problema encontrado es la necesidad de hallar la mejor combinación posible en hiperparámetros en aquellos modelos que lo requieran cómo el SVR, lo que supone una cantidad

de tiempo bastante elevada y conocimientos profundos del modelo. Por último, los modelos estudiados no afrontan los cambios y variaciones externos de carácter temporal y demográficos.

A partir de este trabajo se pueden abordar nuevas vías de investigación cómo modelos de apoye que ayuden a buscar de manera eficiente la búsqueda de hiperparámetros. Además de otros algoritmos más sofisticados y complejos que abarque los problemas encontrados en este trabajo.

En conclusión, podemos afirmar que, tras el estudio realizado en este TFG, una estrategia de precios dinámicos ayuda a aquellas empresas que mejor se aprovechan de esta a ser más competitivas y obtener mejores resultados. Además, nos da una ilustración de la importancia del ML e IA en el panorama actual.

5. Bibliografía

Arslan, H. T. (2011). *Dynamic Pricing Under Consumer Reference-Price Effects.*

Bengio, Y. (2012). *Neural Networks: Tricks of the Trade.* 437-478: Montavon, G., Orr, G. B., & Müller, K.-R.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning.* Springer.

Breiman, L. (2001). Random Forests. *Machine Learning, 45*, 5-32.

Chod, J. &. (2005). Resource flexibility with responsive pricing. *Operations Research, 53*(3), 532-548.

Chollet, F. (2018). *Deep Learning with Python.* Manning Publications.

DATAtab. (2024). *DATAtab e.U.* Obtenido de https://datatab.es/

El Youbi, M. M. (2023). Dynamic pricing strategies optimize benefits, maximize resources, and retain customers through real-time data analysis. *Journal of Business and Economic Research*(5), 123-145.

Elmaghraby, W. &. (2003). Dynamic pricing in the presence of inventory considerations: Research overview, current practices, and future directions. *IEEE Engineering Management Review, 31*(1), 47-47.

Friedman, J. H. (2001). Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, 1189-1232.

Gelman, A. &. (2006). *Data Analysis Using Regression and Multilevel/Hierarchical Models.* Cambridge University Press.

Gönsch, J. K. (2012). Dynamic pricing with strategic customers. *Journal of Business Economics, 83*(5), 505-549.

Goodfellow, I. B. (2016). *Deep Learning.* MIT Press.

Grambsch, P. &. (1991). The effects of transformations and preliminary tests for non-linearity in regression. *Statistics in Medicine*, 697-709.

Hinz, O. H. (2011). Price discrimination in e-commerce? An examination of dynamic pricing in name-your-own-price markets. *MIS Quarterly, 35*(1), 81-98.

Jarrell, S. B. (2019). *Basic Statistics.*

LeCun, Y. B. (2015). Deep learning. *Nature*, 436-444.

Loukas, A. P. (2021). *ArXiv.* Obtenido de https://arxiv.org/abs/2106.04186

Mackay, A. &. (2021). Dynamic pricing algorithms, consumer harm, and regulatory response. *SSRN Electronic Journal*.

Makwe, A. &. (2020). *An empirical study of neural network hyperparameters.* Springer.

Meng, F. K.-J. (2017). An optimal differential pricing in smart grid based on customer segmentation. *IEEE PES Innovative Smart Grid Technologies Conference Europe (ISGT-Europe)*, (págs. 1-6).

Munro, R. C. (2003). Complex spatial relationships. *Third IEEE International Conference on Data Mining*, (págs. 227-234).

Neubert, M. (2022). OP39 The Real-Option Rate Of Return Approach To Inform The Pricing Of Medicines. *International Journal of Technology Assessment in Health Care*.

Ngugi, J. (2024). *Pricing on Point: The Art and Science of Dynamic Pricing*. Obtenido de Medium: https://ngugijoan.medium.com/pricing-on-point-the-art-and-science-of-dynamic-pricing-dd543bf80f01

Nunes, A. T. (2020). The definition and measurement of heterogeneity. *Translational Psychiatry*.

Piepho, H. P. (2019). A coefficient of determination (R^2) for generalized linear mixed models. *Biometrical Journal*, 637-647.

Pupavac, D. (2016). *Dynamic pricing: The future of retail business.* Business Logistics in Modern Management.

Rodó, P. (2021). *Economipedia.com.* Obtenido de https://economipedia.com/definiciones/diagrama-de-caja.html

Ruiz, A. G. (2020). Algoritmo híbrido de redes neuronales artificiales con recocido simulado para predicción en minería de datos. *Computación y Sistemas*, 97-103.

Schluchter, M. D. (2014). *Encyclopedic Dictionary of Archaeology.*

Schutz, H. G. (1973). Mean absolute error as a measure of sensitivity to changes in control. *American Journal of Physiology*, 247-251.

Sutton, R. S. (2018). *Reinforcement Learning: An Introduction.* MIT Press.

Westreicher, G. (2021). *Economipedia.com.* Obtenido de : https://economipedia.com/definiciones/cuartil.html

Wilks, D. S. (2011). *Statistical Methods in the Atmospheric Sciences* (3rd ed ed.). Academic Press.

Ying, X. (2019). An overview of overfitting and its solutions. *Journal of Physics: Conference Series*.

Zhang, Y. &. (2021). Variability of artificial neural networks. *ArXiv*.