UNIVERSIDAD EUROPEA MIGUEL DE CERVANTES

ESCUELA POLITECNICA SUPERIOR

TITULACIÓN:

MÁSTER UNIVERSITARIO EN GESTIÓN Y ANÁLISIS DE GRANDES VOLÚMENES DE DATOS: BIG DATA



TRABAJO FIN DE MÁSTER

RECOMENDADOR DE PRECIOS DE VENTAS DE VEHÍCULOS DE SEGUNDA MANO

AUTOR

ANTONIO DURÁN SÁNCHEZ

TUTOR

MARIO VILLAIZÁN VALLELADO

VALLADOLID, ABRIL 2024

**Contenido**

[1. objetivos del trabajo 3](#_Toc169631073)

[2. análisis de la situación 4](#_Toc169631074)

[3. obtención, procesado y almacenamiento de datos 7](#_Toc169631075)

[3.1. Variables fecha 12](#_Toc169631076)

[3.2. Variables numéricas 13](#_Toc169631077)

[3.3. Variables en formato texto 13](#_Toc169631078)

[4. análisis exploratorio 13](#_Toc169631079)

[5. diseño e implementación de los modelos 22](#_Toc169631080)

[5.1. Regresión lineal 23](#_Toc169631081)

[5.2. Regresión ridge 26](#_Toc169631082)

[5.3. Regresión Lasso 28](#_Toc169631083)

[5.4. Árbol de regresión 30](#_Toc169631084)

[5.5. Random forest 32](#_Toc169631085)

[5.6. Boosted trees 33](#_Toc169631086)

[5.7. Máquinas de vector soporte 35](#_Toc169631087)

[6. análisis de los resultados obtenidos 37](#_Toc169631088)

[7. conclusiones y planes de mejora 41](#_Toc169631089)

[8. bibliografía 44](#_Toc169631090)

[9. anexo 46](#_Toc169631091)

[9.1. CÓDIGO DEL WEB SCRAPING 46](#_Toc169631092)

[9.2. Carga de datos 47](#_Toc169631093)

[9.3. transformación de datos 48](#_Toc169631094)

[9.4. Gráficos y tablas 51](#_Toc169631095)

[9.5. modelos 53](#_Toc169631096)

# objetivos del trabajo

En este trabajo se pretende desarrollar una herramienta que permita a un vendedor de vehículos de segunda estimar el precio de venta por el que debería anunciar su coche. El objetivo es desarrollar un modelo de aprendizaje supervisado que permita predecir este precio, partiendo de datos disponibles en webs de ventas de coches.

Este estudio resulta interesante para la toma de decisiones tanto de compradores como de vendedores. A los vendedores les permitirá determinar un precio de venta adecuado para sus coches, alineado con el mercado. Por su parte, los compradores podrán decidir qué coche les conviene más, teniendo en cuenta las características que estén buscando y los precios estimados.

Con este objetivo en mente, se han identificado una serie de objetivos más específicos:

* Recolección de datos. Los datos se recopilarán mediante procesos de web scraping desde páginas webs de venta de vehículos de segunda mano. Haciendo esto se puede asegurar que la información del estudio es realista y contemporánea.
* Procesamiento de datos. Se utilizarán técnicas de transformación de datos para poder procesarlos. Esto se debe a que los datos de la web no pueden utilizarse directamente, si no que requieren modificaciones como cambio de tipo de datos, extracción de patrones en cadenas, modificación de unidades, etc.
* Análisis exploratorio de datos. Es importante tener una idea clara de cómo son los datos con los que se va a realizar el modelo. Para ello se realizarán distintos tipos de visualizaciones que permitan analizar los comportamientos individuales de las variables.
* Análisis de relación de variables. No todas las variables que se encuentren en la web de venta de coches tienen que influir en el modelo. Se va a analizar la relación entre las distintas variables y la variable objetivo con el fin de tomar decisiones sobre incluir o no ciertos parámetros en el modelo.
* Construcción de modelos de aprendizaje automático. Como objetivo final, tras todo lo anterior, se desarrollarán distintos modelos de aprendizaje automático y se compararán entre sí, utilizando métricas como el Error Cuadrático Medio o el coeficiente de determinación R2.

Para completar estos objetivos, se ha elegido Python como lenguaje de programación, debido a su gran atractivo en el contexto del análisis de datos. Python ofrece una amplia variedad de bibliotecas de código abierto, como pandas y scikit-learn, que facilitan tanto la manipulación de datos como la creación de modelos de aprendizaje automático. Además, la robustez de Python en la ingeniería de software, en general, refuerza esta elección (McKinney, Python for Data Analysis, 2022). Por lo tanto, se usará este lenguaje tanto para la obtención de datos como para su análisis y la creación de modelos.

En conclusión, este proyecto abarca todas las etapas del proceso de ciencia de datos, desde la extracción y preparación de datos hasta la implementación de algoritmos de aprendizaje supervisado. Aunque el enfoque principal es el mercado de coches de segunda mano, las metodologías y técnicas utilizadas son aplicables a diversos tipos de datos. La esencia del proyecto radica en las técnicas y modelos aplicados, más que en la temática específica de los datos analizados.

# análisis de la situación

El proyecto de desarrollo de una herramienta para estimar el precio de venta de vehículos de segunda mano surge en un contexto donde el mercado de compraventa de automóviles usados está experimentando una constante evolución. El mercado de vehículos de ocasión siempre ha sido relevante para la industria del automóvil y presenta una gran relevancia económica, ya que superó los 900 mil millones de USD en 2021, y sus predicciones consideran una tendencia al alza. Predecir con exactitud los precios de los coches usados puede proporcionar información valiosa a compradores y vendedores, lo que permite tomar decisiones con conocimiento de causa. Además, estimar el valor de mercado de un coche usado puede proporcionar información relevante sobre los costes posteriores, como las revisiones periódicas del coche, los gastos de reparación y los costes del seguro, etc (Dutulescu, et al., 2023).

En concreto, tras la pandemia del COVID-19, a finales de 2020, se detectó una escasez de chips en la industria automovilística que, en menor medida, se puede notar en la actualidad (Ramani, Ghosh, & Sodhi, 2022). Por esta falta de microchips y la inflación de estos últimos años, la venta de coches de ocasión ha experimentado un crecimiento considerable, siendo la preferencia de muchos conductores (Figueras, 2023). Además de esto, este auge también se debe a la transparencia comercial que transmite el entorno online, teniendo en cuenta que cuando se habla de este fenómeno se hace referencia a la compra mediante concesionarios, que permiten una compra segura, sencilla y con todas las garantías pertinentes (Metrópoli, 2023).

Es por esto que la realización de procesos de web scraping en webs de compra y venta de vehículos de segunda mano es un método útil para la recolección de datos en este proyecto, ya que permite conocer el mercado desde el punto de vista de las empresas de venta y no desde los ojos de los particulares. Con esto, se puede implementar una herramienta que beneficie tanto al consumidor como al vendedor y, además, permita conocer el estado del mercado.

Es importante resaltar que los precios de venta de los vehículos pueden fluctuar dependiendo de diversos factores bien conocidos en este mercado. Por ejemplo, un coche con mayor potencia tiende a ser más caro que uno con menos potencia, y un coche más antiguo y con muchos kilómetros será más barato que uno con poco recorrido. Además de estos factores técnicos, también influyen factores sociales o valoraciones subjetivas aplicables en todos los mercados, como la popularidad de una marca o modelo, lo que puede provocar un aumento de precios que no necesariamente se correlaciona con las características intrínsecas del coche.

En cuanto a las técnicas de análisis de datos de implementación de modelos predictivos para el precio de venta de vehículos, existen diversos artículos que ya tratan este tema. En concreto, en (Muti & Kazum, 2023), así como en (Chandak, Ganorkar, Sharma, Bagmar, & Tiwari, 2019), utilizan diferentes técnicas como modelos de regresión lineal y árboles de regresión para abordar la predicción de precios en este contexto. Estas metodologías proporcionan una base sólida y comprobada para el desarrollo de modelos predictivos en el mercado de vehículos de segunda mano.

En el caso de este proyecto, se utilizará la potencia del lenguaje Python para el desarrollo de todo el proceso de análisis de datos. En concreto, se utilizará la librería *scikit-learn*, que permite una implementación sencilla de los diferentes modelos de aprendizaje supervisado que pueden ser interesantes en este estudio. Esta librería permite trabajar con diferentes algoritmos, como la regresión lineal, árboles de decisión, máquinas de vector soporte y muchos otros (Buitinck, et al., 2011). Los modelos utilizados se detallarán a lo largo del trabajo y se explicará su parametrización y características.

Además de scikit-learn, en el trabajo se hará uso intensivo de las librerías NumPy y pandas, esenciales para la manipulación y estructuración de datos. **NumPy** es fundamental para trabajar con arreglos multidimensionales y operaciones matemáticas avanzadas, lo que facilita el manejo eficiente de grandes volúmenes de datos numéricos (Oliphant, 2006). Por su parte, **pandas** proporciona estructuras de datos flexibles, como DataFrames, que son ideales para la manipulación y el análisis de datos tabulares. Estas herramientas permiten transformar y preparar los datos para que sean aptos para el modelado, realizando tareas como la limpieza, la transformación y la agregación de datos de manera eficiente (McKinney, Data Structures for Statistical Computing in Python, 2010). La combinación de estas librerías asegura un flujo de trabajo coherente y eficiente desde la recolección de datos hasta el desarrollo del modelo predictivo.

Para complementar el análisis y facilitar la interpretación de los datos, también se utilizarán las librerías de visualización ***matplotlib* y *seaborn***. ***matplotlib*** es una librería versátil que permite crear una amplia gama de gráficos, desde simples diagramas de líneas hasta complejas visualizaciones tridimensionales. Es una herramienta esencial para generar gráficos y diagramas que ayudan a ilustrar patrones y relaciones en los datos (Hunter, 2007). Por otro lado, ***seaborn*** se basa en *matplotlib* y proporciona una interfaz de alto nivel para crear gráficos estadísticos atractivos y fáciles de interpretar. *seaborn* es particularmente útil para explorar y entender la distribución de los datos y las relaciones entre diferentes variables realizando gráficos como los *boxplot* (Waskom, 2021). Estas herramientas serán fundamentales para realizar un análisis exploratorio de los datos, lo que ayudará a identificar tendencias y patrones que puedan influir en la estimación del precio de los vehículos, así como relaciones entre variables.

En cuanto a la forma de recogida de datos, se ha optado por la realización de técnicas de web scrapping. El web scraping es un proceso de extracción de datos de la web que se adapta a determinadas necesidades. Esta técnica proporciona las herramientas necesarias para recopilar datos de sitios web, ya sea para fines personales o profesionales, teniendo en cuenta las consideraciones legales (Chapagain, 2019). Gracias a estas técnicas, la investigación tiene un valor real, ya que los datos se han obtenido de páginas de venta de vehículos actuales y en movimiento. Todo el proceso de recogida de información se detallará en su sección correspondiente dentro de este trabajo.

Para la ejecución de todo este proceso se ha utilizado Anaconda. Anaconda es un software gratuito que dispone de herramientas diseñadas para la investigación científica, con acceso a diferentes entornos en los que se permite programar en Python o R. Estos entornos se conocen como entornos de desarrollo integrado (IDE) y facilitan enormemente el desarrollo de código. Estos IDE contienen muchas funciones útiles para escribir, editar y depurar código, visualizar e inspeccionar datos, almacenar variables, presentar resultados y colaborar en proyectos (Rolon-Mérette, Ross, Rolon-Mérette, & Church, 2020).

Por último, este proyecto no solo se centra en ofrecer una solución tecnológica para la estimación de precios de vehículos de segunda mano, sino que también pretende contribuir al entendimiento más amplio de las dinámicas del mercado automotriz actual. Al integrar técnicas avanzadas de análisis de datos y aprendizaje automático, la herramienta desarrollada no solo proporcionará valor a vendedores y compradores, sino que también demostrará la aplicabilidad y versatilidad de estas metodologías en diversos contextos de negocio.

# obtención, procesado y almacenamiento de datos

Como se ha comentado anteriormente, con el fin de que la investigación tenga valor real, los datos utilizados en este trabajo se han extraído de una web de venta de vehículos de ocasión utilizando técnicas de *web scraping*.

En concreto, en este trabajo se han utilizado los datos de una página web de venta de coches de segunda mano Autohero (Autohero, 2024). Con los procesos de scraping se han logrado extraer un total de 2411 coches, teniendo en cuenta que se han realizado cinco extracciones en los meses de abril y mayo de 2024, con el fin de añadir un mayor número de elementos al estudio, además de mayor variedad en los datos. Si en diferentes extracciones aparecía el mismo coche repetido, se ha optado por mantener el último registro, es decir, se han eliminado duplicados de matrícula manteniendo la última entrada.

Todo el proceso de scraping se ha construido con lenguaje Python, utilizando las librerías *Beautiful Soup* (Richardson, 2023) y *Selenium* (Software Freedom Conservancy, 2024). Estas librerías permiten la exploración de diferentes páginas webs de forma sencilla a través de diferentes marcadores que se pueden encontrar inspeccionando el código fuente de las páginas.

En primer lugar, se ha utilizado Selenium para cargar la página de inicio de Autohero y guardar el enlace de cada coche. Esta librería es útil porque, a través de un driver de un navegador web, que en este caso ha sido Google Chrome, se pueden programar movimientos y acciones en la página web.

**Figura 1**

Página web de Autohero

Interfaz de usuario gráfica, Sitio web

Descripción generada automáticamente

Debido a la estructura de la web, que se puede ver en la Figura 1, al abrirse solo aparecen cierto número de coches. Debido a esto, no se puede leer directamente el código html de la página directamente. Es por esto por lo que se ha utilizado Selenium, ya que permite definir procesos de scroll en la página de manera efectiva para formar la aparición de todos los enlaces.

Con esto en cuenta, se ha realizado un proceso automático que recorre la página de inicio a fin y, cuando el navegador no puede bajar más, se han recorrido uno a uno los enlaces, en orden de relevancia según la propia web.

Tras esto, cada enlace se ha consultado con Beautiful Soup. Esta librería recopila de manera precisa toda la información de una web, con la posibilidad de extraer datos concretos.

**Figura 2**

Url de ejemplo de un coche

Una captura de pantalla de un celular con la imagen de un coche

Descripción generada automáticamente

Como se puede observar en la Figura 2, cada enlace dispone de fotos del coche, que no se van a explotar en este estudio, y los datos de la marca, modelo, versión y precio. Es importante señalar que se utilizará el precio al contado como variable objetivo, ya que no todos los coches disponen del valor de precio financiado que se muestra en la Figura 2.

Dentro de esta misma página, en la parte inferior se pueden encontrar otros datos de interés de los vehículos, que serán los datos más relevantes a la hora de modelar.

**Figura 3**

Datos disponibles en cada coche



**Figura 4**

Otros datos disponibles en cada coche



En la Figura 3 y en la Figura 4 se pueden apreciar otras características relativas al coche en cuestión. Todas estas características se han extraído con el proceso de web scraping y se han unido en tablas de datos, en las que cada fila es un coche y cada columna es una característica, teniendo en cuenta que el número de características que se muestra en cada coche puede ser diferente. El código detallado del proceso se puede consultar en el ANEXO 9.1.

Con este primer proceso se han obtenido 30 variables, aunque estas variables no están depuradas y, en algunos casos, no son explotables, como el estándar de calidad que se puede ver en la Figura 4 que es un enlace para consultar ciertas notaciones que no son relevantes para este trabajo. El listado de variables obtenidas es el siguiente:

1. Marca. Es una de las variables que se conoce que suele influir sobre los precios de los coches. Es una variable categórica.
2. Modelo. Para cada marca existen distintos modelos de coche. El modelo suele tener asociadas ciertas características inherentes, aunque puede depender de la versión. Variable categórica
3. Versión. Los modelos varían según su equipamiento de serie. Este equipamiento lo suele marcar el modelo. Variable categórica.
4. Precio. Precio del coche al contado en euros. Variable numérica.
5. Primera matriculación. Fecha de la primera matriculación del vehículo. Este dato es útil para conocer la antigüedad del coche. Variable fecha.
6. Kilometraje. Kilómetros que ha recorrido el vehículo. Esta variable refleja si el vehículo ha tenido más o menos uso. Variable numérica.
7. Carburante. Tipo de combustible que utiliza el motor. Variable categórica.
8. Transmisión. Tipo de cambio de marcha. Variable categórica.
9. Potencia. Cantidad de trabajo que realiza un coche por unidad de tiempo para impulsarse, medido en caballos de vapor y kilovatios. Variable numérica.
10. Tracción. Mecanismo de movimiento de las ruedas del vehículo. Variable categórica.
11. Tipo de vehículo. Se trata del estilo del coche, que tiene que ver con la forma, tamaño y otras características. Variable categórica.
12. Puertas. Número de puertas del coche. La puerta del maletero se considera puerta cuando tiene luna. Variable numérica discreta.
13. Número de asientos. Cantidad de asientos de los que dispone el vehículo. Variable numérica discreta.
14. Color. Color de la carrocería del coche. Variable categórica.
15. Tapicería. Material del que están fabricados los asientos. Variable categórica.
16. Tipo de rueda. Valor que refleja para qué estación o clima están pensadas las ruedas. Variable categórica.
17. Motor original. Indicador que refleja si un coche preserva su motor original o ha sido sustituido. Variable binaria.
18. Cilindrada. Volumen de los cilindros del vehículo, medido en centímetros cúbicos. Variable numérica.
19. Consumo. Litros que consume el motor por cada 100 kilómetros recorridos. Esta variable incluye el valor de consumo en ciudad, fuera de ciudad y consumo combinado. Variable numérica.
20. Clase de eficiencia CO2. Clasificación de la eficiencia del motor. Variable categórica.
21. Emisiones de CO2. Gramos de emisión expulsados por kilómetro recorrido. Variable numérica.
22. Estándar de calidad. Este campo solo contiene un enlace que lleva hasta los estándares de calidad de Autohero. No se va a utilizar.
23. País de origen. País del que es originario el vehículo. Variable categórica.
24. Número de llaves. Número de llaves que tiene disponibles el coche. Variable numérica discreta.
25. Coche accidentado y reparado. Marca que identifica si un coche ha tenido algún accidente y ha necesitado reparación. Variable binaria.
26. La última revisión se realizó el. Fecha en la que se ha realizado la última revisión del coche. Variable fecha.
27. Tipo de IVA. Indicador dicotómico que refleja si el tipo de IVA es deducible o no. Variable binaria.
28. ITV válida hasta. Fecha hasta la que es válida la última ITV pasada. Variable fecha.
29. Matrícula. Valor que sirve para identificar los coches. No es explotable
30. Número de inventario. Valor que identifica el vehículo en el inventario de la web. No explotable.

Esta lista de datos de cada coche se ha almacenado en ficheros Excel diferentes para cada extracción.

Continuando con el procesado de los datos, en primer lugar, se han unido las distintas extracciones obtenidas en una única tabla, teniendo en cuenta que, si una misma matrícula aparece en dos ficheros de extracción, se mantendrá el último registro de ese vehículo.

Tras esto, se han eliminado todos los registros que tienen algún valor nulo para que el modelado sea más adecuado. Además, se han modificado el tipo y formato de las variables para que sean más sencillas de modelar.

## Variables fecha

Para estudiar las variables de fecha se ha optado por modificar el formato y calcular cuántos días han pasado o van a pasar desde el día de estudio. El objetivo de la creación de las variables que cuentan los días hasta el día actual es poder hacer estudios cuantitativos sobre esas fechas como, por ejemplo, calcular las medias de tiempo.

## Variables numéricas

La mayor parte de las variables numéricas se han extraído como una cadena de caracteres. Dependiendo del caso, se ha realizado un tipo de transformación u otro.

Los tipos de modificaciones que se han realizado son los siguientes:

* Cambio de tipo de número a entero o decimal.
* Supresión de la unidad de medida
* Extracción de patrones en los campos para obtener los consumos en ciudad, fuera de ciudad y combinado.

## Variables en formato texto

Modelizar el precio con variables categóricas en formato texto no es una tarea simple. Para poder realizar los diferentes modelos se van a convertir las variables categóricas en variables dummies y se van a codificar las variables binarias para que tomen los valores 0 y 1. Además, para simplificar los modelos, por cada n categorías que tenga una variable categórica, se construirán n-1 columnas binarias, teniendo en cuenta que la categoría n sería aquella con ceros en el resto de columnas.

Al convertir las variables de tipo cadena en tipo numérico se facilita la elaboración de modelos de regresión.

# análisis exploratorio

Antes de empezar con la modelización de la variable precio, es necesario analizar el resto de las variables independientes. La idea es comprender el comportamiento de los datos e intentar eliminar información redundante.

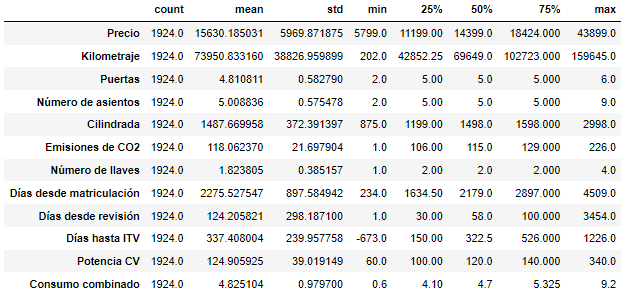
Para realizar esto, se utilizarán diferentes técnicas de estudio descriptivos de los datos y representaciones en formato gráfico, que permitirán analizar el comportamiento de las diferentes variables.

Es importante señalar que antes de empezar el análisis exploratorio de datos se han tomado algunas decisiones sobre el conjunto de datos. Por ejemplo, se he eliminado del estudio la variable versión, ya que es una variable categórica con muchas opciones y produce un número excesivo de variables dummies y se ha optado por conservar únicamente el consumo combinado, descartando el consumo en ciudad y fuera, ya que estos últimos no aparecen siempre en el listado de características y los 3 consumos están demasiado relacionados entre sí como para tomarlos como 3 variables independientes. Además, se han eliminado los registros que tienen alguna variable sin valor o nula, obteniendo así un total de 1924 registros.

En primer lugar, se analizarán las variables numéricas, tanto continuas como discontinuas.

**Figura 5**

Estadísticos descriptivos de las variables numéricas.

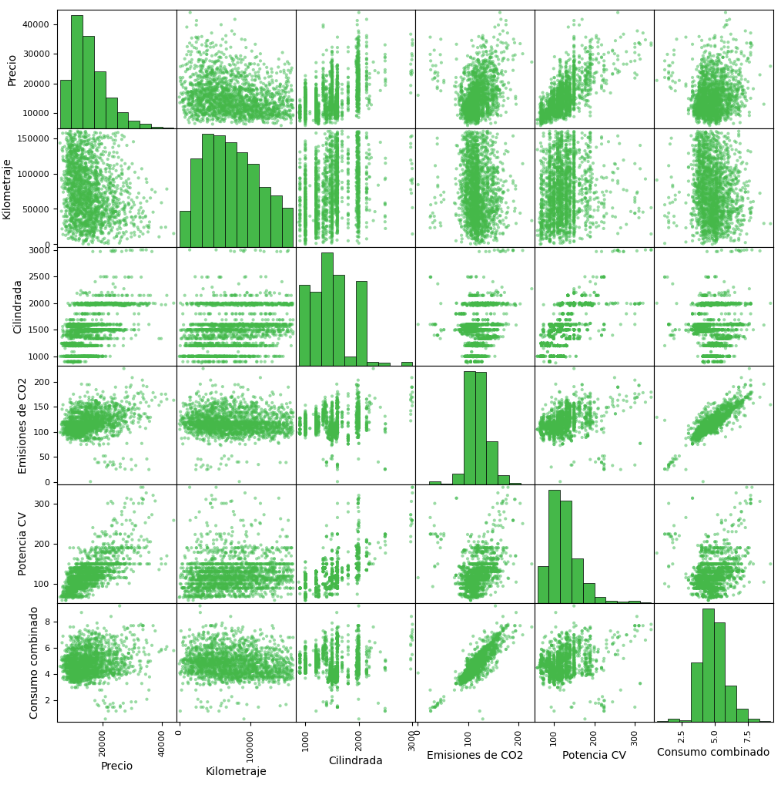


En la Figura 5 se puede observar un resumen estadístico de las variables numéricas. El único resultado resaltable es la diferencia de escala entre las variables, teniendo rangos muy pequeños y de valores pequeños como el número de llaves, y valores y rangos más grandes, como en el caso del kilometraje. Este resultado puede indicar que es necesario un escalado de las variables para que tengan valores similares, que es interesante a la hora de realizar los modelos.

Continuando con el análisis de las variables numéricas, es interesante comprobar cómo están relacionadas unos con otras y, de manera más detallada, con la variable objetivo, precio.

**Figura 6**

Diagrama de dispersión de las variables numéricas



En la Figura 6 se han excluido las variables con valores discretos que tienen un recorrido pequeño, ya que no aportan mucho valor visual en este tipo de gráficos. En este gráfico se puede comprobar la relación que existe entre algunas variables, como el caso de la emisión de CO2 que parece estar relacionada linealmente con el consumo, resultado que a priori era esperable y lógico. Además del comentado anteriormente, existe otro comportamiento lineal, aunque no tan marcado, que es entre la potencia y el precio, y que, como en el caso anterior, es un resultado coherente.

Es interesante estudiar también la relación de las variables numéricas discretas más pequeñas, que son el número de puertas, asientos y llaves, con la variable objetivo. Al ser discretas y con pocos valores, el análisis es similar al que se podría hacer con variables categóricas.

**Figura 7**

Boxplot del número de puertas (izquierda), asientos (centro) y llaves (derecha) con el precio

Gráfico, Gráfico de cajas y bigotes

Descripción generada automáticamente

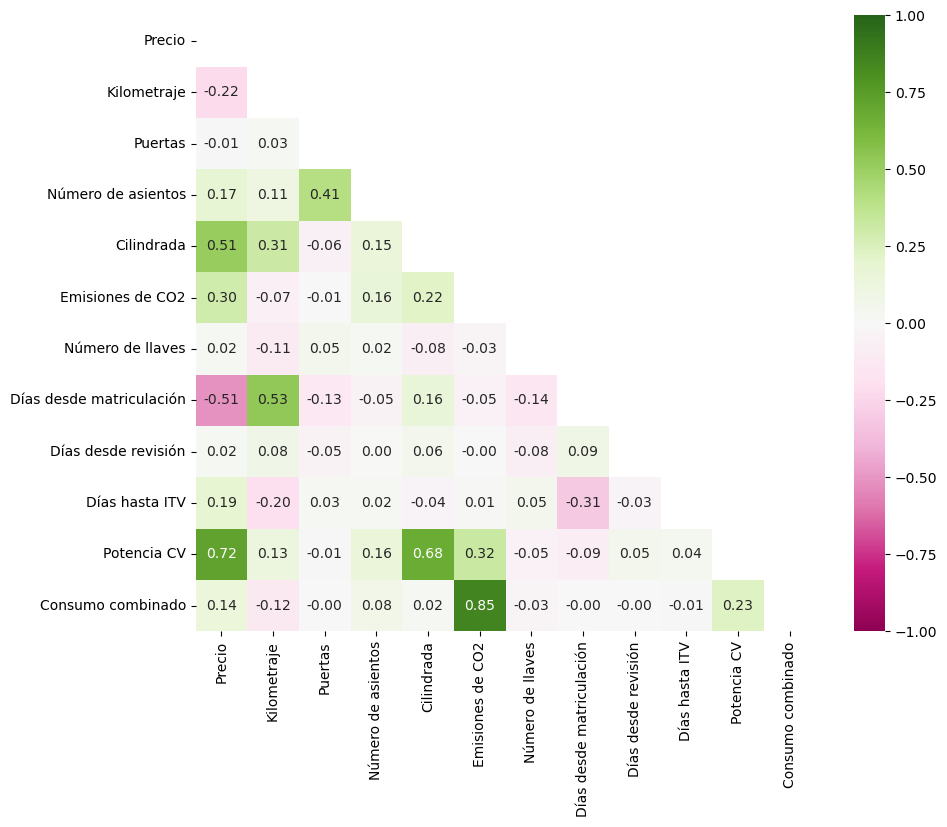
En la Figura 7 se puede comprobar que, mientras que el número de llaves y puertas no parecen influir demasiado en el precio de los vehículos, el número de asientos sí, ya que hay una cierta relación que indica que cuantos más asientos tenga un coche más precio tendrá, cosa que tiene sentido ya que el número de asientos es un indicador del tamaño del coche.

Es importante señalar que, siguiendo los resultados de este gráfico, se puede optar por descartar algunos casos. Primero, los vehículos que tienen dos asientos y un precio superior a 20000€ y, además, el vehículo que tiene 4 llaves y un precio de más de 40000€. Estos datos pueden provocar desajustes en el modelo ya que son comportamientos demasiado extremos en valores de precio que se esperan mucho menores. Para el caso del coche con 4 llaves, se descartará directamente debido a que al descartarlo se elimina una característica de la variable número de llaves, lo que simplificará el modelo. Los casos de coches con 2 asientos y precios excesivamente caros se estudiarán en la implementación de los modelos, cuando se compruebe si esos valores provocan ruido en el modelo.

Tras estos análisis, es conveniente terminar calculando la matriz de correlación, ya que nos proporciona valores aritméticos sobre la relación lineal de las variables.

**Figura 8**

Representación de la matriz de correlación



En la *Figura 8* se han añadido todas las variables numéricas y se ha representado el coeficiente de correlación lineal. Como se ha mencionado anteriormente, las variables más correladas son las emisiones con el consumo y el precio con la potencia, aunque cabe destacar la relación entre la cilindrada y la potencia que no se apreciaba tan bien en la Figura 6. Además, cabe señalar la relación entre los días desde la matriculación del vehículo con el precio y kilometraje, y la de la cilindrada con el precio.

Los resultados obtenidos hasta ahora se mantienen en la línea de la lógica en este campo de estudio, ya que no se han obtenido relación entre variables que a priori no tengan ningún tipo de asociación. Estos resultados pueden ser utilizados más adelante si se requiere una reducción de dimensionalidad del problema, ya que se puede optar por descartar alguna variable redundante y permite saber qué variables van a ser más influyente en el modelado del precio.

A continuación, se estudiarán las variables categóricas y su relación con el precio.

**Figura 9**

Boxplot del precio según la marca

Gráfico, Gráfico en cascada

Descripción generada automáticamente

Como era esperable, con lo obtenido en la Figura 9 parece que la marca del coche tiene bastante influencia sobre el precio, teniendo coches caros de marcas como Cupra y Alfa Romeo, y coches más baratos de Lancia y Chevrolet.

Antes de continuar los análisis, se van a descartar las marcas que cuentan con menos de 4 coches en los datos. Estas marcas serían Chevrolet, Jaguar, Lancia, Abarth, Ssangyong y Land Rover. La eliminación de estas marcas es debida a que, al tener pocos registros, no podrán reflejar bien si su marca es influyente en el precio.

Teniendo en cuenta esta distinción entre marcas, sería interesante estudiar si ocurre algo similar con los modelos de cada marca.

**Figura 10**

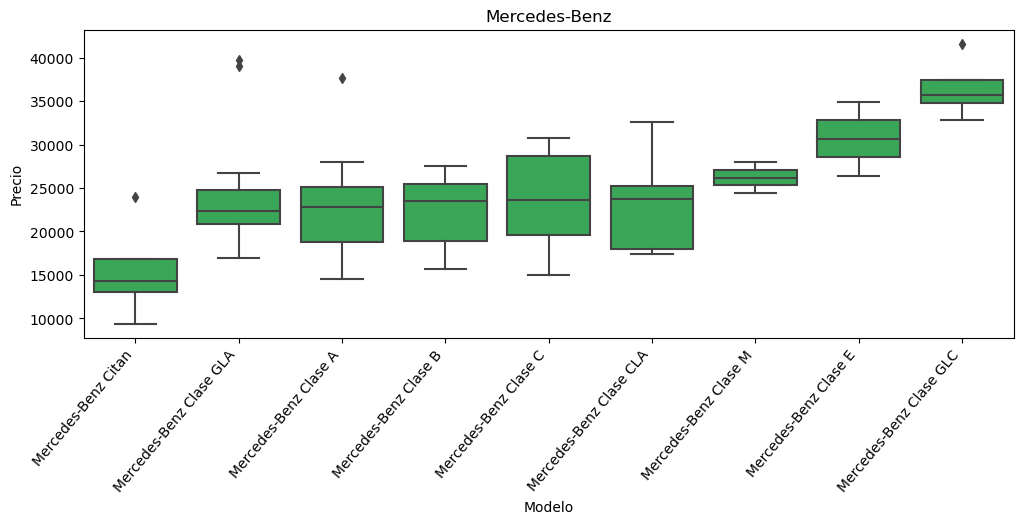
Boxplot del precio según los modelos de Fiat

Gráfico, Gráfico de cajas y bigotes

Descripción generada automáticamente

**Figura 11**

Boxplot del precio según los modelos de Mercedes-Benz



Tanto en la Figura 10, como en la Figura 11 se puede comprobar que existe una relación entre precio y modelo, teniendo cada marca modelos más baratos y caros. Sin embargo, al tener tantos modelos distintos, la creación de variables dummies para modelizar el problema sería demasiado grande, por lo que se va a descartar esta variable del problema. Además, esta variable tiene relación con otras, como el tipo de coche o el número de asientos, que permitirían diferenciar coches dentro de las marcas sin necesidad de conocer el modelo de coche exacto.

**Figura 12**

Boxplot del precio en función del carburante

Gráfico, Gráfico de cajas y bigotes

Descripción generada automáticamente

En cuanto al precio según el tipo de carburante, con lo obtenido en la Figura 12, se puede observar una inclinación en el aumento de precio de los coches híbridos. Además, se puede apreciar que solo existe un registro de coche de Gas Natural, por lo que se descartará, ya que eliminará una categoría de esta variable.

**Figura 13**

Boxplot del precio en función del tipo de cambio

Gráfico, Gráfico de cajas y bigotes

Descripción generada automáticamente

Con respecto a la transmisión o tipo de cambio, se observa que los coches automáticos son más caros que los manuales y la *Figura 13* muestra que solo se disponen de un registro en Semi-automático y otro en Doble embrague. Debido a esto y para reducir categorías, se va a optar por reclasificar esos registros en el tipo automático, quedando los tipos manual y automático. Esto se hace ya que puede resultar interesante tener la distinción entre manual y no manual o automático.

**Figura 14**

Boxplot del precio en función del tipo de vehículo

Gráfico, Gráfico de cajas y bigotes

Descripción generada automáticamente

En la figura Figura 14 se puede comprobar que existen diferencias de precio en cuanto al tipo de vehículo, que parece estar relacionado con el tamaño del coche. Además, como solo existe un registro de tipo Pickup, se eliminará del estudio.

**Figura 15**

Boxplot del precio en función de la clase de eficiencia

Gráfico, Gráfico de cajas y bigotes

Descripción generada automáticamente

En cuanto a la clase de eficiencia que se muestra en la Figura 15, también se puede distinguir cierta tendencia en los precios y otra característica con un único elemento que, de igual manera que en los anteriores casos, se eliminará del modelo.

Con esto, ya se han comentado la relación del precio con las variables más interesantes. No obstante, antes de continuar con el análisis, cabe señalar que la variable País de origen solo presenta el valor “España”, por lo que no influirá en el modelo y se descartará este parámetro para simplificar.

A continuación, se analizarán las correlaciones de todas las variables con la variable objetivo, tras convertir en variables dimmies todas las variables categóricas.

**Tabla 1**

Correlación lineal de las variables con el precio (valor absoluto)

|  |  |
| --- | --- |
| **Variable** | **Correlación** |
| Potencia CV | 0,72 |
| Transmisión: manual | 0,55 |
| Cilindrada | 0,51 |
| Días desde matriculación | 0,51 |
| Tapicería: tejido | 0,38 |
| Tipo de vehículo: coche pequeño | 0,36 |

En la Tabla 1 se pueden observar las variables con mayor correlación lineal con el precio. Cabe señalar que en este estudio no se ha encontrado ninguna correlación relevante con los colores de los coches, por lo que se va a optar por eliminarlos del modelo, ya que es una de las variables con más categorías e introduce muchas variables dummies en el algoritmo. Con esto, el dataset a modelar tiene 63 columnas.

# diseño e implementación de los modelos

Con los datos en formato tabla ya limpios y estructurados se puede comenzar a plantear diferentes modelos de aprendizaje supervisado. El objetivo de estos modelos será la variable precio que, al ser numérica, da información de que los modelos necesarios serán de regresión.

Los diferentes algoritmos a utilizar serán la Regresión Lineal, con sus variantes *Ridge* y *Lasso,* Árboles de Regresión, *Random Forest*, *Boosted Trees* y Máquinas de Vectores Soporte*.*

Para la realización de los diferentes modelos se van a dividir los datos en conjuntos de entrenamiento y test. Gracias a esto se podrán definir métricas que permitan evaluar los modelos y decidir la preferencia entre unos y otros. La partición seleccionada será del 70% para los datos de entrenamiento y 30% para los datos de test. Los datos de entrenamiento servirán para generar los modelos y los de test para evaluarlos.

Otro dato importante antes de comenzar con la implementación de los modelos es que muchos de estos algoritmos dependen de hiperparámetros que deben establecerse antes de ejecutarlos. A diferencia de los parámetros directos del modelo, que se determinan en el entrenamiento, los hiperparámetros de ajuste se tienen que optimizar cuidadósamente para alcanzar un buen rendimiento (Probst, Boulesteix, & Bischl, 2019).

Usualmente, este tipo de optimización se realiza mediante validación cruzada con *k-folds* y, por tanto, será la técnica utilizada en este proyecto. En este tipo de validación, los datos se subdividen aleatoriamente en k pliegues y cada pliegue se utiliza iterativamente como conjunto de validación para un modelo que se ha entrenado utilizando los pliegues restantes (Soper, 2021).

Para realizar este proceso de validación cruzada, es necesario suministrar al sistema una lista de valores para cada hiperparámetro a optimizar y la validación proporcionará la mejor combinación de parámetros de las listas. Es por esto que hay que señalar que los resultados obtenidos dependen de esa muestra inicial de parámetros, por lo que podrían existir otros parámetros que mejoren los modelos. El problema es que la búsqueda exhaustiva de parámetros requiere de excesivo tiempo y capacidad de CPU.

## Regresión lineal

El primer modelo que se va a probar es el de regresión lineal, debido a su simpleza y facilidad de comprensión. Además, con este modelo se comprobará si los coches con 2 puertas que superaban los 20000€ de precio influyen en la definición del modelo.

La regresión lineal es una herramienta útil para predecir variables cuantitativas. Aunque pueda parecer algo insípida en comparación con algunos de los otros enfoques de aprendizaje automático, este tipo de regresión es muy útil y ampliamente utilizada. Este tipo de regresión consiste en predecir una variable respuesta cuantitativa mediante relaciones lineales entre las variables independientes (James, Witten, Hastie, Tibshirani, & Taylor, 2023).

El modelo de regresión se suele escribir de la siguiente manera:

Donde:

* es la variable es la variable dependiente
* es la variable independiente
* es el coeficiente de cada variable independiente
* es el error cometido

Los coeficientes reflejan el efecto que provoca en la respuesta un incremento en la variable y se estiman con el método de mínimos cuadrados. En concreto, la función a minimizar es la siguiente:

Este modelo supone que no existe colinealidad entre las variables independientes, la normalidad, homocedasticidad e independencia de los errores y que el número de variables independientes es menor que el total de datos que se utilizan en el modelo. Para ejecutar este modelo no es necesario definir hiperparámetros de ajuste.

El código utilizado para la elaboración de este modelo es el siguiente:

|  |
| --- |
| from sklearn.linear\_model import LinearRegression  #Modelo  lm = LinearRegression()  lm.fit(X\_train, y\_train)  #Predicción  y\_pred = lm.predict(X\_test)  #Métricas  mse\_test = np.mean((y\_test-y\_pred)\*\*2)  score = lm.score(X\_test, y\_test) |

En este código se utiliza la función de regresión lineal de la librería scikit-learn para realizar el modelo y calcular el error cuadrático medio y el coeficiente de determinación . Las métricas de evaluación obtenidas son las siguientes:

**Tabla 2**

Métricas del modelo de regresión lineal

|  |  |
| --- | --- |
| **ECM** |  |
| 3975171,542 | 0,8789 |

En la Tabla 2 se puede observar que el coeficiente de determinación obtenido es bastante alto, ya que explica alrededor de un 88% de la variabilidad del problema. No obstante, el Error Cuadrático Medio tiene un valor bastante alto, ya que depende de la magnitud de la variable objetivo. Debido a esto y ya que se están utilizando para el modelo una gran cantidad de variables dummies, que tienen valores 0 y 1, se va a intentar modelar el precio escalándolo al intervalo [0,1], tomando como 0 el menor precio y como 1 el precio más alto.

**Tabla 3**

Métricas del modelo de regresión lineal escalado

|  |  |
| --- | --- |
| **ECM** |  |
| 0,0027 | 0,8789 |

Además, aprovechando la simplicidad del modelo, se va a realizar el mismo proceso eliminando el caso de los coches de 2 puertas con un precio de más de 20000€. Se obtienen los siguientes resultados:

**Tabla 4**

Métricas del modelo de regresión lineal escalado y sin casos extremos de 2 puertas

|  |  |
| --- | --- |
| **ECM** |  |
| 0,0029 | 0,8936 |

En la Tabla 4 se refleja la mejoría en el modelo tras haber descartado los casos extremos de precio en coches de dos puertas, ya que se obtiene un coeficiente de determinación mayor, con el que se explica un 89,4% de la variabilidad de los datos, aunque cabe señalar que el ECM aumenta ligeramente.

Como se ha explicado anteriormente, los modelos de regresión lineal deben verificar ciertas condiciones. A continuación, se verificarán las hipótesis sobre los residuos del último modelo de regresión lineal implementado.

**Figura 16**

Residuos frente a valores predichos

Gráfico, Gráfico de dispersión

Descripción generada automáticamente

La primera hipótesis sobre los residuos es que son independientes, pero, como se puede observar en la Figura 16, existe cierto patrón que indica que se comete más error en valores altos de precio. Además de esto, aplicando el test de Shapiro-Wilk para realizar un contraste de hipótesis sobre la normalidad de los residuos se obtiene un p-valor prácticamente nulo, por lo que se descarta la normalidad de los errores.

Con esto, ya se puede decir que el modelo obtenido no cumple las hipótesis generales de los modelos de regresión lineal múltiple, por lo que, a pesar de obtener unos valores muy buenos en términos de R2, no es aconsejable utilizar este modelo, ya que su comportamiento puede no ser efectivo.

Los cambios aplicados en los datos para este modelo se van a conservar en para los siguientes y las comparativas se realizarán sobre los valores de la Tabla 4.

## Regresión ridge

Este es un tipo de regresión que surge de una regularización de la regresión lineal. En este caso, surge por modificar el problema de mínimos cuadrados que se realiza para obtener los coeficientes del modelo (James, Witten, Hastie, Tibshirani, & Taylor, 2023). En particular, los coeficientes de este modelo se obtienen al minimizar la siguiente expresión:

Donde es un hiperparámetro que hay que definir previamente. Con la ecuación anterior se busca minimizar el RSS y penalizar el valor de los coeficientes. Esta penalización de los cuadrados de los coeficientes sirve como medida contra el sobreajuste del modelo.

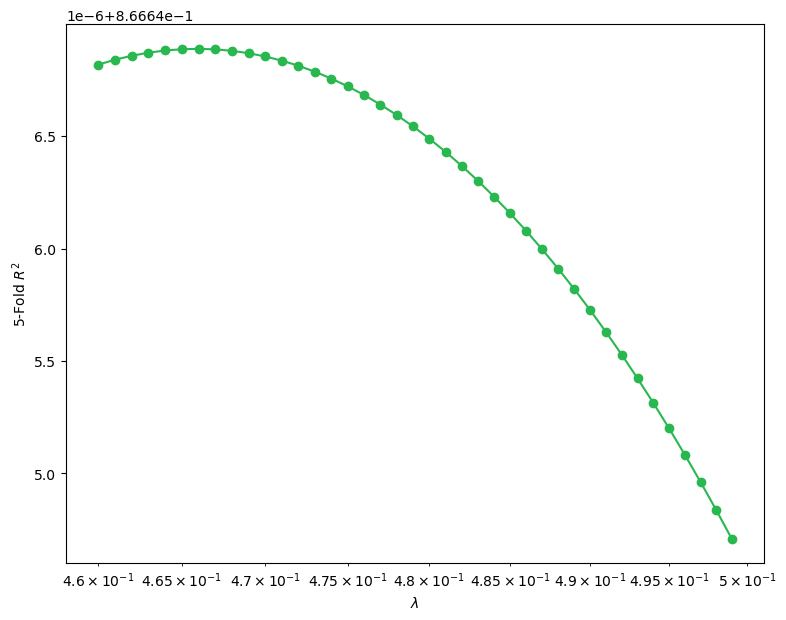
Para obtener el valor óptimo de se han utilizado valores entre 0 y 1, y se ha aplicado el método de validación cruzada para la selección, optimizando el valor de .

El código utilizado para la selección del parámetro es el siguiente:

|  |
| --- |
| from sklearn.linear\_model import Ridge  lambda\_vector = np.arange(0.46, 0.5, 0.001)  param\_grid = {'alpha': lambda\_vector}  grid = GridSearchCV(Ridge(random\_state=0), param\_grid=param\_grid, cv=5)  grid.fit(X\_train, y\_train)  best\_score = grid.best\_score\_  best\_params = grid.best\_params\_ |

**Figura 17**

Evolución de la media de según los valores de en regresión Ridge



En la *Figura 16* se observa cómo crece el valor del coeficiente de determinación hasta un punto y luego comienza a decrecer.

**Tabla 5**

Mejor parametrización para la regresión Ridge

|  |  |
| --- | --- |
| **Media de** |  |
| 0,8667 | 0,466 |

Con esto, se elabora un modelo utilizando la función de regresión Ridge disponible en scikit-learn de la siguiente manera:

|  |
| --- |
| from sklearn.linear\_model import Ridge  #Modelo  ridge = Ridge(random\_state=0,alpha = 0.466).fit(X\_train, y\_train)  #Predicción  y\_pred = ridge.predict(X\_test)  #Métricas  mse\_test = np.mean((y\_test-y\_pred)\*\*2)  score = ridge.score(X\_test, y\_test) |

**Tabla 6**

Métricas del modelo de regresión Ridge

|  |  |
| --- | --- |
| **ECM** |  |
| 0,0029 | 0,8942 |

En la ***Tabla 6*** se observa que el valor del error cuadrático medio no disminuye en este modelo, en comparación con la regresión lineal, mientras que el valor del coeficiente de determinación aumenta, por lo que este modelo explica mejor el problema, aunque el aumento no es muy significativo.

## Regresión Lasso

La regresión Lasso es una alternativa a la regresión Ridge, que surge para corregir una desventaja de esta última. En la regresión Ridge se incluyen todas las variables del modelo, ya que incrementar el valor de λ hace tender a reducir las magnitudes de los coeficientes, pero no sirve para excluir ninguna de las variables (James, Witten, Hastie, Tibshirani, & Taylor, 2023).

Los coeficientes de la regresión Lasso son la solución por mínimos cuadrados de la siguiente expresión:

Esta expresión tiene una formulación similar a la de la regresión Ridge, cambiando el cuadrado por la norma. La diferencia fundamental es que en la regresión Lasso se realiza una selección de variables, las cuales van a tener coeficientes 0 en el modelo.

Como en el caso de la regresión Ridge, para obtener el valor óptimo de se han utilizado valores entre 0 y 1, y se ha aplicado el método de validación cruzada para la selección, optimizando el valor de .

El código utilizado es el siguiente:

|  |
| --- |
| from sklearn.linear\_model import Lasso  alpha\_vector = np.arange(0.00001, 0.0001, 0.000001)  param\_grid = {'alpha': alpha\_vector }  grid = GridSearchCV(Lasso(random\_state=0), param\_grid=param\_grid, cv=5)  grid.fit(X\_train, y\_train)  best\_score = grid.best\_score\_  best\_params = grid.best\_params\_ |

**Figura 18**

Evolución de la media de según los valores de λ en regresión Lasso

Gráfico, Gráfico de líneas

Descripción generada automáticamente

**Tabla 7**

Mejor parametrización para la regresión Lasso

|  |  |
| --- | --- |
| **Media de** |  |
| 0,8668 | 3,9 x 10-5 |

Teniendo en cuenta este resultado, se realiza un modelo con esta parametrización utilizando la función de regresión Lasso de scikit-learn.

|  |
| --- |
| from sklearn.linear\_model import Lasso  #Modelo  alpha\_optimo = grid.best\_params\_['alpha']  lasso = Lasso(random\_state=0,\*\*grid.best\_params\_).fit(X\_train, y\_train)  #Predicción  y\_pred = lasso.predict(X\_test)  #Métricas  mse\_test = np.mean((y\_test-y\_pred)\*\*2)  score = lasso.score(X\_test, y\_test) |

**Tabla 8**

Métricas del modelo de regresión Lasso

|  |  |
| --- | --- |
| **ECM** |  |
| 0,0029 | 0,8952 |

En la *Tabla 8* se ve que el valor del es ligeramente superior a los modelos anteriores, consiguiendo explicar un 89,52 % de la variabilidad de los datos.

## Árbol de regresión

Los árboles de decisión son mecanismos de regresión y clasificación de propósito general que son más usados en los problemas de análisis de datos. La característica principal de los árboles de decisión es una subdivisión recursiva de un conjunto de datos en función de las variables de entrada para crear particiones y conjuntos descendentes asociados, llamados hojas o nodos, que contienen valores intrahoja progresivamente similares y valores interhoja progresivamente diferentes en cualquier nivel del árbol (De Ville, 2013).

En este caso, el modelo es un árbol de regresión y las divisiones en subconjuntos se realizan con preguntas del tipo , ya que la variable respuesta es cuantitativa.

Para este modelo se optimizarán dos parámetros, el número mínimo de elementos de cada hoja y profundidad máxima. Se puede fijar el número mínimo de elementos de cada hoja para reducir la complejidad del modelo, ya que cuántos más elementos se exijan en cada hoja, menos complejidad tendrá el modelo. Por otro lado, la profundidad máxima indica la cantidad de divisiones que se pueden hacer.

|  |
| --- |
| from sklearn.tree import DecisionTreeRegressor  max\_depth = np.arange(9, 16, 1)  min\_samples\_leaf = np.arange(17, 20, 1)  param\_grid = {'max\_depth': max\_depth, 'min\_samples\_leaf': min\_samples\_leaf}  grid = GridSearchCV(DecisionTreeRegressor(random\_state=0),                      param\_grid=param\_grid, cv=5, return\_train\_score=True)  grid.fit(X\_train, y\_train)  best\_score = grid.best\_score\_  best\_params = grid.best\_params\_ |

**Figura 19**

Evolución de la media de según los parámetros del árbol de regresión

Gráfico, Gráfico de líneas

Descripción generada automáticamente

**Tabla 9**

Mejor parametrización del árbol de regresión

|  |  |  |
| --- | --- | --- |
| **Media de** | **Máx. Profundidad** | **Mín. núm. hojas** |
| 0,7379 | 10 | 18 |

Con este resultado, se realiza un árbol de regresión con la función pertinente de scikit-learn.

|  |
| --- |
| from sklearn.tree import DecisionTreeRegressor  #Modelo  arbol1 = DecisionTreeRegressor(random\_state=0,\*\*grid.best\_params\_).fit(X\_train, y\_train)  #Predicción  y\_pred = arbol1.predict(X\_test)  #Métricas  mse\_test = np.mean((y\_test-y\_pred)\*\*2)  score = arbol1.score(X\_test, y\_test) |

**Tabla 10**

Métricas del árbol de regresión

|  |  |
| --- | --- |
| **ECM** |  |
| 0,0073 | 0,7339 |

Como se observa en la Tabla 10, este modelo no mejora ni el modelo de regresión ni los modelos regularizados.

## Random forest

El random forest es un predictor que consiste en la colección de M árboles de decisión aleatorios que se entrenan de forma paralela. Para obtener una estimación final se realiza el promedio de los resultados de cada árbol (Biau & Scornet, 2016). Este tipo de algoritmos en paralelo sirven para casos en los que existe alta varianza y bajo sesgo.

Para este modelo se han optimizado 3 parámetros, el número mínimo de elementos por hojas, máxima profundidad y número de árboles utilizados.

|  |
| --- |
| from sklearn.ensemble import RandomForestRegressor  max\_depth = np.arange(27, 50, 1)  min\_samples\_leaf = np.arange(2, 30, 1)  n\_estimators = [850,851,852]  param\_grid = {'max\_depth': max\_depth, 'min\_samples\_leaf': min\_samples\_leaf, 'n\_estimators': n\_estimators}  grid = GridSearchCV(RandomForestRegressor(random\_state=0),                      param\_grid=param\_grid, cv=5, return\_train\_score=True)  grid.fit(X\_train, y\_train)  best\_score = grid.best\_score\_  best\_params = grid.best\_params\_ |

**Tabla 11**

Mejor parametrización del random forest

|  |  |  |  |
| --- | --- | --- | --- |
| **Media de** | **Máx. Profundidad** | **Mín. núm. hojas** | **Núm. árboles** |
| 0,8332 | 28 | 2 | 852 |

En este caso no se ha realizado una representación gráfica, ya que el problema es tridimensional y su gráfica no sería representable de manera sencilla. Con esta parametrización se puede realizar un modelo de random forest utilizando la función correspondiente de scikit-learn.

|  |
| --- |
| from sklearn.ensemble import RandomForestRegressor  #Modelo  arbol2 = RandomForestRegressor(random\_state=0, max\_depth= 28,                                 n\_estimators = 852).fit(X\_train, y\_train)  #Predicción  y\_pred = arbol2.predict(X\_test)  #Métricas  mse\_test = np.mean((y\_test-y\_pred)\*\*2)  score = arbol2.score(X\_test, y\_test) |

**Tabla 12**

Métricas del random forest

|  |  |
| --- | --- |
| **ECM** |  |
| 0,0041 | 0,8496 |

Como se puede ver reflejado en la Tabla 12, este algoritmo ha mejorado el árbol de regresión estándar, pero no mejora los resultados de los modelos de regresión lineal.

## Boosted trees

Los boosted trees son una técnica de regresión que consiste en la generación de árboles secuenciales en los que cada árbol aprende de los errores del modelo anterior, construyendo un modelo final robusto (Coadou, 2013).

En este caso se optimizarán el número de estimadores empleados, la máxima profundidad y el ratio de aprendizaje. La tasa de aprendizaje indica la contribución de cada árbol en el modelo final.

Esta optimización se ha realizado en dos partes. Primero se han optmizado juntos los parámetros de número de estimadores y tasa de aprendizaje. Tras obtenerse unos valores estables de esos dos parámetros, se ha fijado el número de estimadores y se han optimizado la tasa de aprendizaje y la máxima profundidad de manera conjunta.

|  |
| --- |
| from sklearn.ensemble import GradientBoostingRegressor  #primero se busca el número de estimadores y la tasa de aprendizaje  n\_estimators = [1800, 1900,2000,2500]  learning\_rate = [0.025,0.029,0.03, 0.035,0.032]  param\_grid = {'n\_estimators': n\_estimators,'learning\_rate': learning\_rate}  grid = GridSearchCV(GradientBoostingRegressor(random\_state=0),                      param\_grid=param\_grid, cv=5, return\_train\_score=True)  grid.fit(X\_train, y\_train)  best\_score = grid.best\_score\_  best\_params = grid.best\_params\_  #se fija el número de estimadores a 1800 y se optimizan los otros dos  learning\_rate = np.arange(0.03, 0.04, 0.001)  max\_depth = np.arange(2, 10, 1)  param\_grid = {'max\_depth': max\_depth,'learning\_rate': learning\_rate}  grid = GridSearchCV(GradientBoostingRegressor(random\_state=0,  n\_estimators = 1800),                      param\_grid=param\_grid, cv=5, return\_train\_score=True)  grid.fit(X\_train, y\_train)  best\_score = grid.best\_score\_  best\_params = grid.best\_params\_ |

**Tabla 13**

Mejor parametrización del boosted trees

|  |  |  |  |
| --- | --- | --- | --- |
| **Media de** | **Máx. Profundidad** | **Tasa de aprendizaje** | **Núm. árboles** |
| 0,9002 | 2 | 0,037 | 1800 |

Con la parametrización obtenida se va a entrenar un modelo de boosted trees con la función de scikit learn.

|  |
| --- |
| from sklearn.ensemble import GradientBoostingRegressor  #Modelo  arbol3 = GradientBoostingRegressor(random\_state=0, learning\_rate = 0.037,                                     max\_depth=2,n\_estimators = 1800).fit(X\_train, y\_train)  #Predicción  y\_pred = arbol3.predict(X\_test)  #Métricas  mse\_test = np.mean((y\_test-y\_pred)\*\*2)  score = arbol3.score(X\_test, y\_test) |

**Tabla 14**

Métricas del boosted trees

|  |  |
| --- | --- |
| **ECM** |  |
| 0,0023 | 0,9152 |

Como se observa en la Tabla 14, este modelo es el que aporta un mejor resultado hasta ahora, explicando un 91,52% de la variabilidad del estudio.

## Máquinas de vector soporte

Las máquinas de vector soporte o SVM son modelos de aprendizaje automático supervisado que se utilizan en problemas de clasificación y regresión. En el caso de la clasificación, lo que se busca es construir un hiperplano que permita separar los datos de manera correcta y esto se consigue generalizar a los problemas de regresión incluyendo una región muy pequeña alrededor de la función, llamada tubo. Esta región permite reformular el problema de optimización a encontrar el mejor tubo que se aproxime a la función continua, equilibrando al mismo tiempo la complejidad del modelo y el error de predicción (Awad & Khanna, 2015).

Este modelo tiene varios hiperparámetros que se pueden optimizar como el kernel, que define la forma de los hiperplanos que dividen los datos, el parámetro C, que penaliza los errores de clasificación y otros parámetros que modifican la forma de los kernel.

En primer lugar, se intentará optimizar utilizando el kernel “rbf”. En este caso, se buscará un valor óptimo de C y del parámetro de forma del kernel utilizado, en este caso gamma.

|  |
| --- |
| from sklearn.svm import SVR  C = np.arange(197499, 197509, 1)  gamma = np.arange(0.00002, 0.000032, 0.000002)  grid = GridSearchCV(SVR(kernel = 'rbf'),                     param\_grid={'C': C, 'gamma': gamma},                     cv = 5, return\_train\_score=True)  grid.fit(X\_train, y\_train)  best\_score = grid.best\_score\_  best\_params = grid.best\_params\_ |

**Tabla 15**

Mejor parametrización del SVR con kernel rbf

|  |  |  |  |
| --- | --- | --- | --- |
| **Media de** | **C** | **Gamma** | **kernel** |
| 0.8713 | 197500 | 3 x 10-5 | *rbf* |

Tras esto, también se ha intentado optimizar el modelo utilizando la función sigmoide.

|  |
| --- |
| from sklearn.svm import SVR  C = np.arange(32, 50, 1)  gamma = np.arange(0.0080, 0.0091, 0.0001)  grid = GridSearchCV(SVR(kernel = 'sigmoid'),                     param\_grid={'C': C, 'gamma': gamma},                     cv = 5, return\_train\_score=True)  grid.fit(X\_train, y\_train)  best\_score = grid.best\_score\_  best\_params = grid.best\_params\_ |

**Tabla 16**

Mejor parametrización del SVR con kernel sigmoide

|  |  |  |  |
| --- | --- | --- | --- |
| **Media de** | **C** | **Gamma** | **kernel** |
| 0,842225 | 42 | 0,0081 | *sigmoid* |

Es remarcable la diferencia en las dimensiones de la parametrización, sobre todo con el parámetro C, que se optimiza para valores mucho más pequeños cuandos e utiliza la función sigmoide. Con las parametrizaciones obtenidas se van a generar y comparar los dos modelos obtenidos.

|  |
| --- |
| from sklearn.svm import SVR  #Modelos  svm = SVR(kernel = 'rbf', \*\*grid.best\_params\_).fit(X\_train, y\_train)  svm2 = SVR(kernel = 'sigmoid', \*\*grid.best\_params\_).fit(X\_train, y\_train)  #Predicción  y\_pred = svm.predict(X\_test)  y\_pred2 = svm2.predict(X\_test)  #Métricas  mse\_test = np.mean((y\_test-y\_pred)\*\*2)  score = svm.score(X\_test, y\_test)  mse\_test2 = np.mean((y\_test-y\_pred2)\*\*2)  score2 = svm2.score(X\_test, y\_test) |

**Tabla 17**

Métricas del SVR

|  |  |  |
| --- | --- | --- |
| **kernel** | **ECM** |  |
| *rbf* | 0,0034 | 0,8742 |
| *sigmoid* | 0,0035 | 0,8732 |

En la **Tabla *17*** se refleja una comparativa entre los dos núcleos utilizados. Con las parametrizaciones obtenidas, el mejor predictor, aunque no por mucha diferencia, es el que utiliza el núcleo *rbf*.

# análisis de los resultados obtenidos

Se han utilizado siete modelos diferentes y, para cada uno de ellos se han calculado métricas de evaluación. Es importante recordar que todos estos modelos se han realizado con un escalado de las variables al intervalo [0,1], lo que ha facilitado la optimización de los hiperparámetros, ya que los algoritmos se han resuelto de una manera más sencilla.

Como resumen de los modelados se tiene lo siguiente:

**Tabla 18**

Comparativa de los diferentes modelos

|  |  |  |
| --- | --- | --- |
| **Modelo** | **ECM** |  |
| Lineal | 0,0029 | 0,8936 |
| Ridge | 0,0029 | 0,8942 |
| Lasso | 0,0029 | 0,8952 |
| Árbol de regresión | 0,0073 | 0,7339 |
| Random forest | 0,0041 | 0,8496 |
| Boosted trees | 0,0023 | 0,9152 |
| SVR | 0,0034 | 0,8742 |

Como se puede comprobar en la Tabla 18, el mejor rendimiento se obtiene con el modelo de árboles potenciados o *boosted trees*, mientras que el modelo que obtiene peores resultados es el árbol de regresión. Cabe destacar que todos estos resultados dependen de la elección de los parámetros utilizada y que, aunque se han obtenido mediante un proceso de optimización con validación cruzada, pueden existir mejores combinaciones de parámetros para que los modelos se ajusten más a los datos.

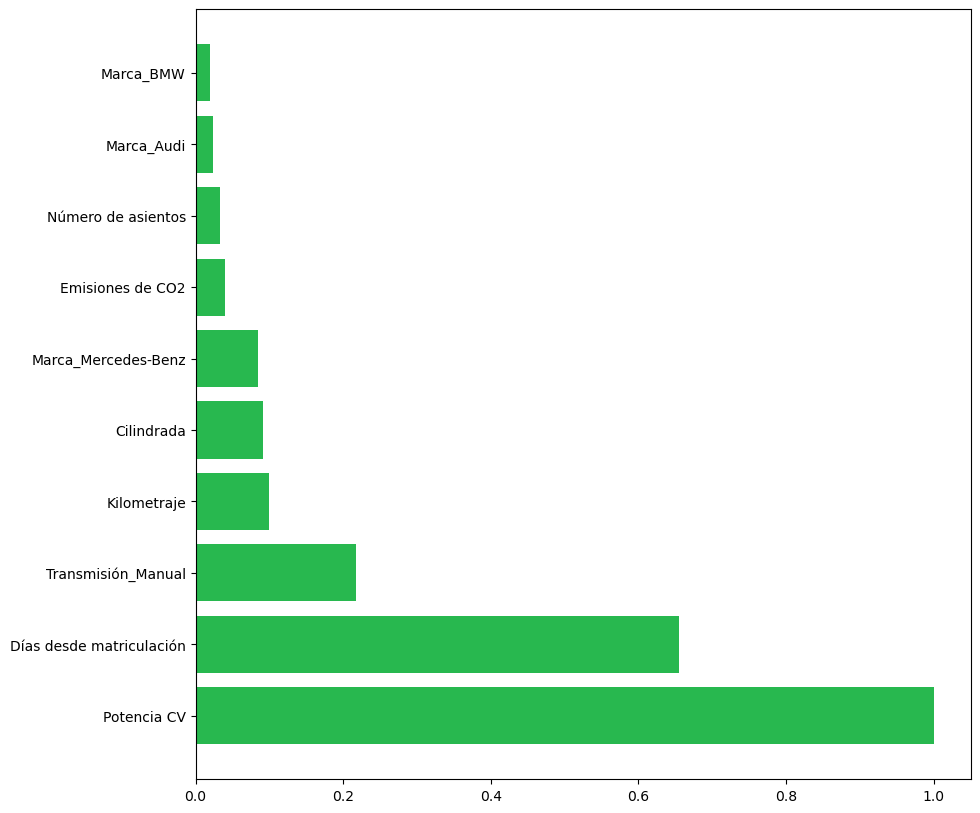
Otra cosa a tener en cuenta es la simplicidad de los modelos. Teniendo en cuenta que los tres primeros son muy simples y que tienen unos resultados buenos, se podría tomar como modelo definitivo cualquiera de ellos, ya que permiten una interpretación sencilla que los otros modelos no.

No obstante, en este estudio no se están tratando un número de datos demasiado grande, por lo que se tomará como mejor modelo el de boosted trees, ya que es el que más reduce el error cuadrático medio y es capaz de explicar un 91,52% de la información.

Los modelos basados en árboles tienen la característica de que permiten medir la importancia de las variables en el modelo. Esta característica es interesante, ya que permite una visión general de qué variable aporta más en la predicción de la variable objetivo, en este caso del precio, pudiendo hacer una comparativa con las correlaciones estudiadas en el análisis exploratorio de datos.

**Figura 20**

Importancia de las variables en el modelo de boosted trees



En la Figura 19 se refleja que la variable más importante para la predicción del precio es la potencia, cosa que es coherente y se puede presuponer antes de realizar el estudio. Además, como se vio en la sección del análisis exploratorio de datos, esta variable era la que tenía mayor correlación lineal con el precio.

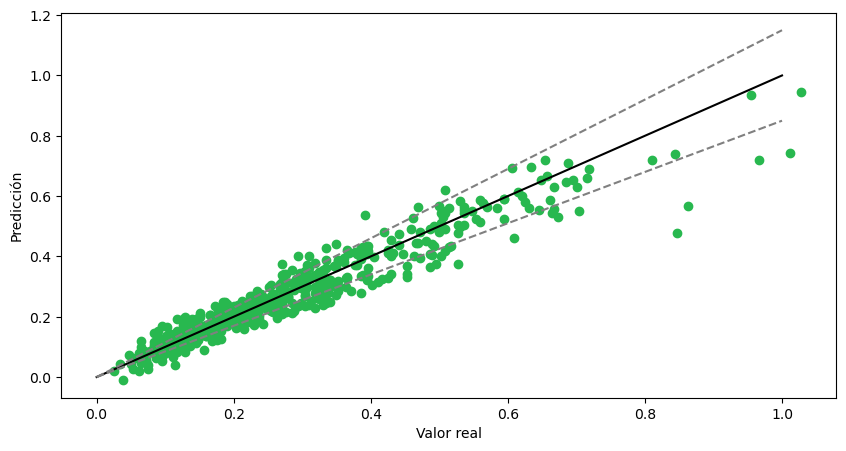
Los días desde matriculación y el tipo de transmisión también aparecían como unas de las variables más correladas linealmente con el precio y encaja con la lógica del estudio. También cabe destacar que las marcas más influyentes a la hora de seleccionar el precio de un coche son aquellas consideradas más populares o lujosas socialmente.

Además de la importancia de las variables, es interesante visualizar cómo de bien realiza las predicciones el modelo seleccionado. Esto se puede hacer enfrentando los valores de predicción y los reales en una gráfica, y comprobando cómo de cerca están los puntos de la recta y = x.

Para observar esto se va a utilizar una nueva extracción de datos como conjunto de validación. En este conjunto se analizará la capacidad de predicción para nuevos datos que no se han utilizado ni en el entrenamiento ni la prueba. Los datos de validación provienen de una nueva extracción de Autohero el día 15 de junio, teniendo 504 coches diferentes.

**Figura 21**

Predicción frente a valores reales de validación. Intervalo de error del 15%

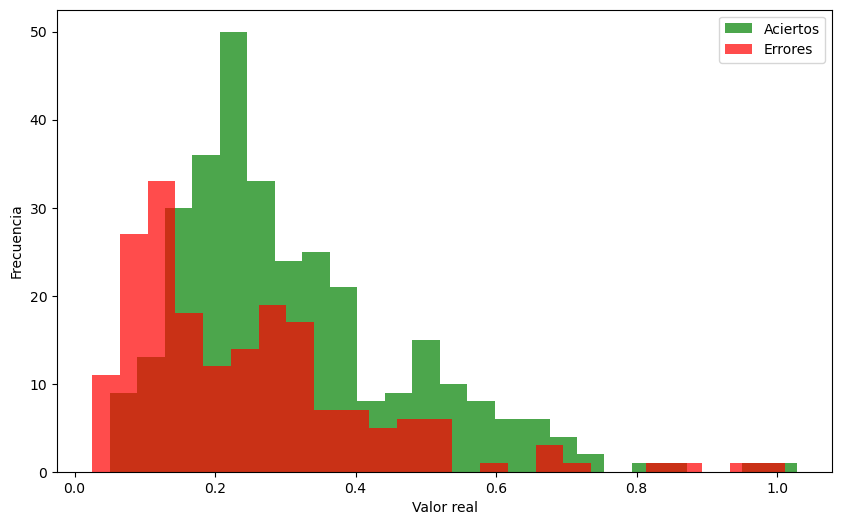


En la Figura 20 se ha representado el valor real frente a la predicción y unas bandas que representan una desviación del 15 % en los valores. Esto es porque, una diferencia del 15% en valores cercanos a 0 es una cantidad pequeña, mientras que en valores más altos la diferencia es mayor. Es importante recordar que se han representado los datos de validación aplicando las mismas transformaciones que los datos originales.

Con esto se comprueba que los valores intermedios se comportan bien frente a las predicciones, pero no se identifica bien el comportamiento de los valores más pequeños. Para observarlo mejor, se van a representar los aciertos y errores, considerando como acierto aquellas predicciones que no se desvían más de un 15% del valor real.

**Figura 22**

Frecuencia de errores y aciertos en datos de validación (15%)



En la Figura 21 se puede comprobar que existe un rango de valores en los que el número de aciertos y errores ronda el 50%, lo que quiere decir que los coches que están en esos intervalos pueden no clasificarse en un rango de precios correcto o, por lo menos, no en un rango menor que el 15% de su valor original. No obstante, el número de aciertos es considerablemente mayor que el de errores

# conclusiones y planes de mejora

Para este trabajo se ha realizado un análisis de regresión en coches de segunda mano con lenguaje Python, apoyando los cálculos con distintas librerías. El objetivo de este proyecto ha sido el desarrollo de un modelo que permita a los vendedores de coches saber qué precio es el habitual para el coche que desean vender, además de permitir a los compradores invertir mejor su dinero, teniendo en cuenta las características de coches que deseen.

En un inicio, se plantearon una serie de objetivos que se han ido desarrollando conforme se avanzaba en los análisis. En primer lugar, se ha realizado una recopilación de datos con web scrapping en la página web de Autohero, con lo que los datos a tratar tienen sentido práctico y sirven para reflejar coherentemente el mercado actual de venta de coches de segunda mano.

Por otra parte, se han procesado y limpiado los datos con el apoyo del lenguaje Python. En este procesado se han configurado columnas de datos con las que posteriormente se ha trabajado, se han extraído patrones de texto para separar número de unidades o separar campos que de origen estaban unidos, se han modificado los tipos de datos y se han generado variables ficticias para facilitar el proceso de modelo, entre otras cosas.

Además, se ha realizado un análisis exploratorio de los datos, indagando en qué aspectos de los datos de estudio resultaba más interesante y qué modificaciones se pueden hacer al conjunto de datos para que los algoritmos sean más eficientes. También se ha analizado cómo se comportan unas variables con respecto a otras, cosa que se ha reflejado en el modelo final.

Por último, se han realizado una serie de modelos de aprendizaje automático con algoritmos de regresión, como son los modelos lineales y sus regularizaciones, árboles de decisión y variantes en paralelo y consecutivas, y regresión con vectores soporte. En este último paso se ha seleccionado el boosted trees como mejor modelo, ya que sus métricas de evaluación eran mejores que el resto.

En conclusión, con el desarrollo de este trabajo se han satisfecho las necesidades iniciales, recogiendo datos relevantes para el mercado de coches de segunda mano, analizando características relevantes, construyendo modelos y comparándolos para establecer la mejor estrategia de predicción, siempre teniendo en cuenta que estos modelos están condicionados por la elección de parámetros. Es por esto, que los análisis de este proyecto están sujetos a mejoras en la elección de los modelos, pero se han obtenido resultados interesantes y satisfactorios, que pueden ser relevantes en un mercado tan importante como el de la compra y venta de coches.

Por otro lado, es importante señalar que los análisis de este estudio se pueden replicar en cualquier ámbito, Por ejemplo, en este estudio no se han tenido en cuenta otro tipo de vehículos que no sean coches, por lo que se puede implementar un análisis y modelos similares para otro tipo de vehículos, como las motos, o incluso aplicarlo en otro tipo de mercado.

En adición a esto, cabe señalar que el trabajo tiene un margen de mejora considerable. Para empezar, como se ha comentado anteriormente, la elección de hiperparámetros en los modelos es crucial en este tipo de proyectos. Con más tiempo y potencia computacional se podrían encontrar mejores valores para los parámetros que los obtenidos en este trabajo.

Por otra parte, se podrían intentar implementar otros tipos de modelos. Existen muchos modelos de regresión o variantes a las utilizadas que no se han explorado en este estudio de venta de coches de segunda mano. La exploración de diferentes algoritmos como otros tipos de máquinas de vector soporte o redes neuronales pueden ocasionar una mejoría en la predicción del precio de los coches a las que este estudio no ha podido llegar. No obstante, los resultados obtenidos con los modelos aplicados son satisfactorios, ya que cumplen con los objetivos previamente marcados.

Otra posible área de mejora sería en la selección de variables. En el desarrollo de este trabajo no se ha indagado en seleccionar las variables más influyentes del modelo y realizar modelos con menos variables, comparando los resultados. Al tener 64 variables, se ha optado por incluir la mayor información posible y, viendo que los modelos reflejan resultados interesantes, no se ha optado por el análisis de selección de características. Es habitual en la práctica realizar modelos únicamente con las características más relevantes y, de esta manera, simplificar los cálculos de los algoritmos, cosa que se ha descartado en este proyecto, ya que simplemente con el escalado de variables al intervalo [0,1] se ha obtenido una gran eficiencia en los modelos.

En cuanto al intervalo de tiempo del estudio, una línea de mejoras destacable podría ser la extracción de datos en un periodo más largo o por temporadas. Este tipo de análisis podrían hacer relucir patrones temporales de venta de vehículos y valorar si la fecha de venta es un indicador estacional de los precios. Además, con más extracciones a lo largo del tiempo, aumentaría el número de datos con los que se puede entrenar el modelo y puede solucionar problemas como el de la inclusión de un coche con características no modeladas.

Añadiendo al tema de los datos, se podrían añadir nuevas fuentes, con datos provenientes de diferentes páginas de ventas de coches de segunda mano, incluyendo páginas de otros países. Esto puede enriquecer los datos y permitir elaborar modelos más interesantes para un público global.

Por último, la idea final de un proyecto de este estilo sería terminar con en el desarrollo de una aplicación web que permita incluir las características de un coche. De esta manera, usuarios de distintas partes de España, podrían valorar sus coches antes de ponerlos en el mercado, y como comprador se podría valorar cuánto es necesario gastar para comprar un coche con las características deseadas.

En resumen, el trabajo realizado propone una base estructurada y sólida para predecir el precio de los coches de segunda mano, pero está sujeta a refinamientos y mejoras. Entre otras cosas, destacan la implementación de diferentes algoritmos, el enriquecimiento de los datos o el despliegue de una aplicación final que facilite la utilización del modelo. Con estas mejoras se podría enriquecer el proyecto y mejoraría la capacidad de decisión de los interesados en esta herramienta predictora.

# bibliografía

Autohero. (2024). *Autohero*. Obtenido de https://www.autohero.com/es

Awad, M., y Khanna, R. (2015). Support Vector Regression. En *Efficient Learning Machines: Theories, Concepts, and Applications for Engineers and System Designers* (p. 67–80). Apress. doi:10.1007/978-1-4302-5990-9\_4

Biau, G., y Scornet, E. (2016). A random forest guided tour. *Test, 25*, 197-227. doi:10.1007/s11749-016-0481-7

Buitinck, L., Louppe, G., Blondel, M., Pedregosa, F., Mueller, A., Grisel, O., Niculae, V., Prettenhofer, P., Gramfort, A., Grobler, J., Layton, R., VanderPlas, J., Joly, A., Holt, B. Varoquaux, G. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research, 12*, 2825-2830. Obtenido de https://scikit-learn.org/stable/

Chandak, A., Ganorkar, P., Sharma, S., Bagmar, A., y Tiwari, S. (2019). Car price prediction using machine learning. *International Journal of Computer Sciences and Engineering (IJCSE), 7*(5), 444-450. doi:10.26438/ijcse/v7i5.444450

Chapagain, A. (2019). *Hands-On Web Scraping with Python: Perform advanced scraping operations using various Python libraries and tools such as Selenium, Regex, and others.* Packt Publishing Ltd.

Coadou, Y. (2013). Boosted decision trees and applications. *EPJ Web of conferences.* doi:10.1051/epjconf/20135502004

De Ville, B. (2013). Decision trees. *Wiley Interdisciplinary Reviews: Computational Statistics*, 448-455. doi:10.1002/wics.1278

Figueras, M. (06 de Marzo de 2023). El precio de un coche de segunda mano se dispara un 33% en tres años y ya cuesta 18.000 euros de media. *La Vanguardia*.

Hunter, J. (2007). Matplotlib: A 2D graphics environment. *Computing in Science & Engineering, 9*(3), 90-95. doi:10.1109/MCSE.2007.55

James, G., Witten, D., Hastie, T., Tibshirani, R., y Taylor, J. (2023). En *An introduction to statistical learning: With applications in python.* Springer. doi:10.1007/978-3-031-38747-0\_3

McKinney, W. (2010). Data Structures for Statistical Computing in Python. (S. van der Walt, & J. Millman, Edits.) *Proceedings of the 9th Python in Science Conference, 445*, 56 - 61. doi:10.25080/Majora-92bf1922-00a

McKinney, W. (2022). *Python for Data Analysis* (3ª ed.). O'Reilly Media, Inc.

Metrópoli. (14 de Septiembre de 2023). Los coches de segunda mano se posicionan en lo alto del mercado de la automoción. *El Español*.

Muti, S., y Kazum, Y. (2023). Using linear regression for used car price prediction. (I. Akkurt, Ed.) *International Journal of Computational and Experimental Science and ENgineering (IJCESEN), 9*(1), 11-16. doi:10.22399/ijcesen.1070505

Oliphant, T. (2006). *Guide to numpy* (Vol. 1). Trelgol Publishing USA.

Probst, P., Boulesteix, A.-L., y Bischl, B. (2019). Tunability: Importance of hyperparameters of machine learning algorithms. *Journal of Machine Learning Research, 20*(53), 1-32. Obtenido de http://jmlr.org/papers/v20/18-444.html

Ramani, V., Ghosh, D., y Sodhi, M. (2022). Understanding systemic disruption from the Covid-19-induced semiconductor shortage for the auto industry. *Omega, 113*(102720).

Richardson, L. (2023). *Beautiful Soup 4.12.0 documentation*. Obtenido de https://www.crummy.com/software/BeautifulSoup/bs4/doc/

Rolon-Mérette, D., Ross, M., Rolon-Mérette, T., y Church, K. (2020). Introduction to Anaconda and Python: Installation and setup. *The Quantitative Methods for Psychology, 16*(5), S3-S11. doi:10.20982/tqmp.16.5.S003

Software Freedom Conservancy. (2024). *Selenium 4.21.0 documentation*. Obtenido de https://www.selenium.dev/selenium/docs/api/py/api.html

Soper, D. (2021). Greed is good: Rapid hyperparameter optimization and model selection using greedy k-fold cross validation. *Electronics, 10*(16), 1973. doi:10.3390/electronics10161973

# anexo

## CÓDIGO DEL WEB SCRAPING

|  |
| --- |
| from selenium import webdriver  from selenium.webdriver.common.by import By  import time  import requests  from bs4 import BeautifulSoup  from datetime import datetime  import pandas as pd  import numpy as np  # Inicializa el navegador  driver = webdriver.Chrome(executable\_path='C:\Windows\System32\chromedriver.exe')  # Maximiza la ventana del navegador (abre en pantalla completa)  driver.maximize\_window()  # Abre la página web  driver.get("https://www.autohero.com/es/search/")  # Lista para almacenar los enlaces  links = []  # Simula hacer scroll y carga los enlaces en cada scroll  for \_ in range(100):  # Ajusta el número de scrolls según sea necesario      # Hace scroll hasta el final de la página      driver.execute\_script("window.scrollTo(0, document.body.scrollHeight);")      # Espera a que la página se cargue después del scroll      time.sleep(2)      # Extrae los enlaces después de cada scroll y agrega a la lista      enlaces = driver.find\_elements(By.CSS\_SELECTOR, 'a.link\_\_\_2Maxt')      for enlace in enlaces:          links.append(enlace.get\_attribute('href'))  # Elimina duplicados de la lista de enlaces  links = list(set(links))  # Cierra el navegador  driver.quit()  #Crea el dataframe con los datos de cada link  datos = pd.DataFrame()  for pagina in links[650:]:      #si hacemos muchas consultas a la página nos echa así que vamos a poner una espera      time.sleep(2)      url = pagina      r = requests.get(url)      time.sleep(2)      html\_contents = r.text      html\_soup = BeautifulSoup(html\_contents, 'html.parser')        time.sleep(2)      #Estos valores siempre aparecen en cada página      #Vamos a sacar los elementos por su clase      tit = html\_soup.select\_one('.titleText\_\_\_edZSG')      try:          nombre = tit.text          marca = nombre.split(' ', 1)[0]          modelo = nombre.split(' ', 1)[1]          precio = float(html\_soup.select\_one('.vehiclePrice\_\_\_1uUmJ').text.strip('\xa0€').replace('.',''))          version = html\_soup.select\_one('.subtitleText\_\_\_2wcYx').text          listaValores = [marca, modelo, precio, version]          listaNombres = ['Marca', 'Modelo', 'Precio', 'Versión']          #El resto de valores pueden o no aparecer en cada página          #primero tratamos las etiquetas grandes de características          carac1 = html\_soup.select('.listItemValue\_\_\_1IWSE')          caracteristicas1 = [i.text for i in carac1]          nombrecarac1 = html\_soup.select('.listItemTitle\_\_\_2CQBv')          nombrecaracteristicas1 = [i.text for i in nombrecarac1]          listaValores.extend(caracteristicas1)          listaNombres.extend(nombrecaracteristicas1)          #ahora se extrae la información de las tablas inferiores          carac2 =html\_soup.select('.body\_\_\_2uId6')          caracteristicas2 = [i.text for i in carac2]          listaValores.extend(caracteristicas2)          listaVal = [listaValores]          nombrecarac2 =html\_soup.select('.itemTitle\_\_\_3GH8k')          nombrecaracteristicas2 = [i.text for i in nombrecarac2]          listaNombres.extend(nombrecaracteristicas2)          datos\_pagina = pd.DataFrame(listaVal,columns=listaNombres)          datos = pd.concat([datos, datos\_pagina], ignore\_index=True)      except:          datos = datos  datos.to\_excel('coches1.xlsx',index=False) |

## Carga de datos

|  |
| --- |
| import pandas as pd  import numpy as np  import matplotlib.pyplot as plt  import seaborn as sns  import timeit  from sklearn.model\_selection import train\_test\_split,GridSearchCV  from sklearn.feature\_selection import f\_regression, mutual\_info\_regression  from sklearn.linear\_model import LinearRegression, Ridge, Lasso  from sklearn.metrics import r2\_score,mean\_squared\_error  from sklearn.tree import DecisionTreeRegressor  from sklearn.ensemble import RandomForestRegressor  from sklearn.preprocessing import MinMaxScaler, StandardScaler  from sklearn.svm import SVR  from sklearn.ensemble import GradientBoostingRegressor  from scipy.stats import shapiro  pd.set\_option('display.max\_rows', None)  # Muestra todas las filas  datos1=pd.read\_excel('datos1.xlsx')  datos2=pd.read\_excel('datos2.xlsx')  datos3=pd.read\_excel('datos3.xlsx')  datos4=pd.read\_excel('datos4.xlsx')  datos5=pd.read\_excel('datos5.xlsx')  datosVal=pd.read\_excel('datosVal.xlsx')  datospre = pd.concat([datos1,                     datos2,                     datos3,                     datos4,                     datos5]).drop\_duplicates('Matricula',  keep = 'last').reset\_index(drop=True).drop(      ['Versión','Distintivo Ambiental','Historial de revisiones',  'Estándar de calidad'], axis=1)  #Carga de datos de validación para unificar transformaciones  datos = pd.concat([datospre,                    datosVal]).drop\_duplicates('Matricula',  keep = 'first').reset\_index(drop=True).drop(      ['Versión','Distintivo Ambiental','Historial de revisiones',  'Estándar de calidad'], axis=1) |

## transformación de datos

|  |
| --- |
| #Modificación marcas  mapeo\_marcas = {'Alfa': 'Alfa Romeo', 'Land': 'Land Rover'}  def transformar\_marca(marca):      return mapeo\_marcas.get(marca, marca)  datos['Marca'] = datos['Marca'].apply(transformar\_marca)  def transformar\_modelo(marca, modelo):      if marca == 'Alfa Romeo':          return modelo.split(' ', 1)[1] if modelo.startswith('Romeo') else modelo      elif marca == 'Land Rover':          return modelo.split(' ', 1)[1] if modelo.startswith('Rover') else modelo      else:          return modelo  datos['Modelo'] = datos.apply(lambda row: transformar\_modelo(row['Marca'], row['Modelo']), axis=1)  #Para evitar modelo iguales en dos marcas se van a unir marca y modelo  datos['Modelo'] = datos['Marca'] + ' ' + datos['Modelo']  #Eliminación de nulos  datos = datos.dropna()  datos.reset\_index(drop=True, inplace=True)  #Modificar variables para poder tratarlas  #las fechas se van a convertir en fechas de pandas y luego en días hasta el actual, para que sean números y se puedan comparar  datos['Primera matriculación'] = pd.to\_datetime(datos['Primera matriculación'], format = '%d.%m.%Y')  datos['Días desde matriculación'] = (pd.Timestamp.now() - datos['Primera matriculación']).dt.days  datos['La última revisión se realizó el'] = pd.to\_datetime(datos['La última revisión se realizó el'], format = '%d.%m.%Y')  datos['Días desde revisión'] = (pd.Timestamp.now() - datos['La última revisión se realizó el']).dt.days  datos['ITV válida hasta'] = pd.to\_datetime(datos['ITV válida hasta'], format = '%d.%m.%Y')  datos['Días hasta ITV'] = (datos['ITV válida hasta'] - pd.Timestamp.now()).dt.days  #las variables numéricas  datos['Kilometraje'] = datos['Kilometraje'].str.replace(' km', '').str.replace('.','').astype(float)  datos['Potencia CV'] = datos['Potencia'].str.extract(r'(\d+) CV \/ (\d+) kW').iloc[:,0].astype(float)  datos['Cilindrada']  = datos['Cilindrada'].str.replace(' ccm', '').astype(float)  datos['Emisiones de CO2'] = datos['Emisiones de CO2'].str.replace(' g/km', '').astype(float)  datos['Puertas'] = datos['Puertas'].astype(int)  datos['Precio'] = datos['Precio'].astype(float)  #El consumo hay que partirlo en 3  partes = datos['Consumo'].str.split(')', expand=True)  datos['Consumo ciudad'] = partes.iloc[:,0].str.extract(r'(\d+(?:\.\d+)?) l\/100 km \(En la ciudad').astype(float)  datos['Consumo combinado'] = partes.iloc[:,1].str.extract(r'(\d+(?:\.\d+)?) l\/100 km \(Combinado').astype(float)  #arreglo de combinados que aparecen en la primera columna  incluir = partes.iloc[:,0].str.extract(r'(\d+(?:\.\d+)?) l\/100 km \(Combinado').astype(float)[0]  datos['Consumo combinado'] = datos['Consumo combinado'].combine\_first(incluir)  datos['Consumo fuera'] = partes.iloc[:,2].str.extract(r'(\d+(?:\.\d+)?) l\/100 km \(Fuera de la ciudad').astype(float)  #Eliminación de campos que no se utilizarán  datos = datos.drop(['Consumo ciudad','Consumo fuera','Primera matriculación',                      'La última revisión se realizó el','ITV válida hasta','Número de inventario'], axis=1)  #Eliminación outliers de llaves y precio  datos = datos[~((datos['Número de llaves'] == 4) & (datos['Precio'] > 40000))]  #Eliminación marcas con menos de 4 covhes  datos = datos[~((datos['Marca'] == 'Chevrolet') |                  (datos['Marca'] == 'Jaguar') |                  (datos['Marca'] == 'Lancia') |                  (datos['Marca'] == 'Abarth') |                  (datos['Marca'] == 'Ssangyong') |                  (datos['Marca'] == 'Land Rover') )]  # Para algunas variables se pueden hacer agrupaciones para reducir los diferentes valores de cada una, por ejemplo,  #en el tipo de cambio para eliminar el semi-automático y el  #doble embrague. En la variable tapicería es necesario eliminar el texto entre paréntesis.  #Eliminamos el pais, ya que solo tiene un valor.  # El coche con ruedas de emergencia no tiene sentido y se va a eliminar  # Se va a eliminar el coche de gas natural  # Se va a eliminar el de tipo Pickup  # Se va a eliminar la clase de eficiencia EURO 6c  # Todo esto va a hacer que tengamos menos variables dummies, cosa que beneficiará los procesos de búsqueda de los modelos  datos['Tapicería'] = [texto.split('(')[0].strip() for texto in datos['Tapicería']]  mapeo\_transmision = {'Semi-automático': 'Automático', 'Doble embrague': 'Automático',                      'Cambio tipo manual': 'Manual', 'Cambio tipo automático': 'Automático'}  def transformar\_transmision(transmision):      return mapeo\_transmision.get(transmision, transmision)  datos['Transmisión'] = datos['Transmisión'].apply(transformar\_transmision)  datos = datos[~((datos['Tipo de ruedas'] == 'Emergencia') |                  (datos['Tipo de vehículo'] == 'Pickup') |                  (datos['Carburante'] == 'Gas natural') |                  (datos['Clase de eficiencia CO2'] == 'EURO 6c'))]  datos = datos.drop('País de origen', axis=1)  #Binarias  datos['Motor original'] = datos['Motor original'].map({'Sí': 1, 'No': 0})  datos['Coche accidentado y reparado'] = datos['Coche accidentado y reparado'].map({'Sí': 1, 'No': 0})  datos['Tipo de IVA'] = datos['Tipo de IVA'].map({'IVA no deducible': 0, 'IVA deducible': 1})  #Categóricas  datos = pd.get\_dummies(datos, columns=['Marca','Modelo',                                           'Carburante','Transmisión','Tracción','Tipo de vehículo',                                           'Color','Tapicería','Tipo de ruedas','Clase de eficiencia CO2'],                                           dtype=int,drop\_first=True)  datos = datos.select\_dtypes(exclude=['object'])  columnas = datos.filter(regex='^(Color\_|Modelo\_)').columns  # Eliminar las columnas seleccionadas  datos = datos.drop(columns=columnas)  #Separar datos de validación  datosVal = datos[datos['Validacion'] == 1].drop('Validacion',axis = 1)  datos = datos[datos['Validacion'] == 0].drop('Validacion',axis = 1)  X=datos.drop(['Precio'],axis=1)  y=datos['Precio']  #split de entrenamiento y test  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y,test\_size=0.3,random\_state=0) |

## Gráficos y tablas

|  |
| --- |
| datos.describe().T  #boxplots  fig, axes = plt.subplots(1, 3, figsize=(12,4))  plot1 = sns.boxplot(x='Puertas', y='Precio', data=datos, color='#28b84f', ax=axes[0])  plot1.set(xlabel='Nº puertas')  plot2 = sns.boxplot(x='Número de asientos', y='Precio', data=datos, color='#28b84f', ax=axes[1])  plot2.set(xlabel='Nº asientos')  plot2.set\_ylabel('')  plot2.set\_yticklabels([])  plot3 = sns.boxplot(x='Número de llaves', y='Precio', data=datos, color='#28b84f', ax=axes[2])  plot3.set(xlabel='Nº llaves')  plot3.set\_ylabel('')  plot3.set\_yticklabels([])  plt.tight\_layout()  plt.show()  fig, axes = plt.subplots(2, 3, figsize=(16,16))  k = 0  for i in datos.select\_dtypes(exclude=['object', 'datetime64', 'int']).columns:      sns.boxplot(datos[i],ax = axes.flatten()[k], color = '#28b84f').set\_title(i)      k = k+1  plt.tight\_layout()  plt.show()  plt.subplots(figsize=(12,4))  mediana\_precios\_por\_marca = datos.groupby('Marca')['Precio'].median().sort\_values()  marca\_order = mediana\_precios\_por\_marca.index  plot = sns.boxplot(x='Marca', y='Precio', data=datos, color='#28b84f',order=marca\_order)  plot.set(xlabel='Marca')  locs, labels = plt.xticks(rotation=50, ha='right')  plt.show()  for i in marca\_order:      plt.subplots(figsize=(12,4))      mediana\_precios\_por\_modelo = datos[datos['Marca'] == i].groupby('Modelo')['Precio'].median().sort\_values()      modelo\_order = mediana\_precios\_por\_modelo.index      plot = sns.boxplot(x='Modelo', y='Precio', data=datos[datos['Marca'] == i], color='#28b84f',order=modelo\_order)      plot.set(xlabel='Modelo')      locs, labels = plt.xticks(rotation=50, ha='right')      plot.set\_title(i)  for i in ['Carburante','Transmisión','Tracción','Tipo de vehículo',            'Color','Tapicería','Tipo de ruedas','Clase de eficiencia CO2',            'País de origen','Motor original','Coche accidentado y reparado',           'Tipo de IVA']:      plt.subplots(figsize=(12,4))      mediana\_precios\_por\_carac = datos.groupby(i)['Precio'].median().sort\_values()      carac\_order = mediana\_precios\_por\_carac.index      plot = sns.boxplot(x=i, y='Precio', data=datos, color='#28b84f',order=carac\_order)      plot.set(xlabel='')      plot.set\_title(i)      locs, labels = plt.xticks(rotation=40, ha='right')    #scatter plot  from pandas.plotting import scatter\_matrix  # Crear matriz de diagramas de dispersión  scatter\_matrix(datos.select\_dtypes(exclude=['object','int']), figsize=(12, 12),                 color='#28b84f') # Esto no cambia el color de las diagonales  # Cambiar color de las barras en los gráficos de la diagonal  for ax in plt.gcf().get\_axes():      if hasattr(ax, 'hist'):          for \_, spine in ax.spines.items():              spine.set\_visible(True)  # Mostrar los bordes del gráfico          for bar in ax.patches:              bar.set\_facecolor('#28b84f')  # Color de las barras del histograma              bar.set\_edgecolor('black') # Borde de las barras              bar.set\_linewidth(0.5) # Grosor del borden de las barras  plt.show()  #correlación  correlacion = datos.select\_dtypes(exclude=['object']).corr()  mask = np.zeros\_like(correlacion)  plt.figure(figsize = (10,8))  mask[np.triu\_indices\_from(mask)] = True  sns.heatmap(correlacion,mask=mask,vmin = -1, vmax=1, center=0, cmap="PiYG",annot=True,fmt=".2f")  plt.show()correlacion = datos.select\_dtypes(exclude=['object']).corr()  mask = np.zeros\_like(correlacion)  plt.figure(figsize = (10,8))  mask[np.triu\_indices\_from(mask)] = True  sns.heatmap(correlacion,mask=mask,vmin = -1, vmax=1, center=0, cmap="PiYG",annot=True,fmt=".2f")  plt.show()  pd.DataFrame(abs(correlacion['Precio']).sort\_values(ascending=False).head(20)) |

## modelos

|  |
| --- |
| #lineal  lm = LinearRegression()  lm.fit(X\_train, y\_train)  y\_pred = lm.predict(X\_test)  mse\_test = np.mean((y\_test-y\_pred)\*\*2)  print('El ECM es {:.6f}'.format(mse\_test))  score = lm.score(X\_test, y\_test)  print('El R^2 es {:.6f}'.format(score))  # Escalamos todas las variables al intervalo [0,1]  X=MinMaxScaler(feature\_range=(0, 1)).fit\_transform(datos.drop(['Precio'],axis=1))  X = pd.DataFrame(X, columns=datos.drop(['Precio'],axis=1).columns)  y=MinMaxScaler(feature\_range=(0, 1)).fit\_transform(datos['Precio'].array.reshape(-1, 1))  y = pd.DataFrame(y, columns=['Precio'])  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y,test\_size=0.3,random\_state=0)  lm = LinearRegression()  lm.fit(X\_train, y\_train)  y\_pred = lm.predict(X\_test)  mse\_test = np.mean((y\_test-y\_pred)\*\*2)  print('El ECM es {:.6f}'.format(mse\_test))  score = lm.score(X\_test, y\_test)  print('El R^2 es {:.6f}'.format(score))  # Probamos a quitar los coches caros de 2 puertas  datos = datos[~((datos['Puertas'] == 2) & (datos['Precio'] > 20000))]  X=MinMaxScaler(feature\_range=(0, 1)).fit\_transform(datos.drop(['Precio'],axis=1))  X = pd.DataFrame(X, columns=datos.drop(['Precio'],axis=1).columns)  y=MinMaxScaler(feature\_range=(0, 1)).fit\_transform(datos['Precio'].array.reshape(-1, 1))  y = pd.DataFrame(y, columns=['Precio'])  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y,test\_size=0.3,random\_state=0)  y\_train = y\_train.values.ravel()  y\_test = y\_test.values.ravel()  lm = LinearRegression()  lm.fit(X\_train, y\_train)  y\_pred = lm.predict(X\_test)  mse\_test = np.mean((y\_test-y\_pred)\*\*2)  print('El ECM es {:.6f}'.format(mse\_test))  score = lm.score(X\_test, y\_test)  print('El R^2 es {:.6f}'.format(score))  ##hipótesis  y\_pred\_train = lm.predict(X\_train)  resid = y\_pred\_train - y\_train  plt.figure(figsize=(10,5))  plt.scatter(y\_pred\_train, resid,color = '#28b84f')  plt.xlabel('Valores Predicción')  plt.ylabel('Residuos')  plt.axhline(y=0, color='black', linestyle='--')  plt.show()  shapiro\_stat, shapiro\_pvalue = shapiro(resid)  shapiro\_pvalue  #Ridge  lambda\_vector = np.arange(0.46, 0.5, 0.001)  param\_grid = {'alpha': lambda\_vector }  grid = GridSearchCV(Ridge(random\_state=0), param\_grid=param\_grid, cv=5)  grid.fit(X\_train, y\_train)  print("best mean cross-validation score: {:.6f}".format(grid.best\_score\_))  print("best parameters: {}".format(grid.best\_params\_))  scores = np.array(grid.cv\_results\_['mean\_test\_score'])  plt.figure(figsize=(9, 7))  plt.semilogx(lambda\_vector, scores, '-o',color='#28b84f')  plt.xlabel('$\lambda$', fontsize=10)  plt.ylabel('5-Fold $R^2$')  plt.show()  alpha\_optimo = grid.best\_params\_['alpha']  ridge = Ridge(random\_state=0,\*\*grid.best\_params\_).fit(X\_train, y\_train)  y\_pred = ridge.predict(X\_test)  mse\_test = np.mean((y\_test-y\_pred)\*\*2)  print('El ECM es {:.6f}'.format(mse\_test))  score = ridge.score(X\_test, y\_test)  print('El R^2 es {:.6f}'.format(score))  #Lasso  alpha\_vector = np.arange(0.00001, 0.0001, 0.000001)  param\_grid = {'alpha': alpha\_vector }  grid = GridSearchCV(Lasso(random\_state=0), param\_grid=param\_grid, cv=5)  grid.fit(X\_train, y\_train)  print("best mean cross-validation score: {:.6f}".format(grid.best\_score\_))  print("best parameters: {}".format(grid.best\_params\_))  scores = np.array(grid.cv\_results\_['mean\_test\_score'])  plt.figure(figsize=(9, 7))  plt.semilogx(alpha\_vector, scores, '-o',color='#28b84f')  plt.xlabel('$\lambda$', fontsize=10)  plt.ylabel('5-Fold $R^2$')  plt.show()  alpha\_optimo = grid.best\_params\_['alpha']  lasso = Lasso(random\_state=0,\*\*grid.best\_params\_).fit(X\_train, y\_train)  y\_pred = lasso.predict(X\_test)  mse\_test = np.mean((y\_test-y\_pred)\*\*2)  print('El ECM es {:.6f}'.format(mse\_test))  score = lasso.score(X\_test, y\_test)  print('El R^2 es {:.6f}'.format(score))  #Árbol de regresión  max\_depth = np.arange(9, 16, 1)  min\_samples\_leaf = np.arange(16, 21, 1)  # rejilla de valores posibles que queremos explorar  param\_grid = {'max\_depth': max\_depth, 'min\_samples\_leaf': min\_samples\_leaf}  # Validaremos el modelo mediante validación cruzada en n\_folds  n\_folds = 5  grid = GridSearchCV(DecisionTreeRegressor(random\_state=0),                      param\_grid=param\_grid, cv=5, return\_train\_score=True)  grid.fit(X\_train, y\_train)  print("best mean cross-validation score: {:.6f}".format(grid.best\_score\_))  print("best parameters: {}".format(grid.best\_params\_))  arbol1 = DecisionTreeRegressor(random\_state=0,\*\*grid.best\_params\_).fit(X\_train, y\_train)  error = grid.cv\_results\_['mean\_test\_score'].reshape(len(max\_depth),len(min\_samples\_leaf))  for i, lr in enumerate(list(max\_depth)):      plt.plot(min\_samples\_leaf, error[i,],'--o', label='max\_depth = %g'%lr)  plt.legend()  plt.xlabel('min\_samples\_leaf')  plt.ylabel('5-Fold $R^2$')  plt.title('train: %0.3f\ntest: %0.3f'%(arbol1.score(X\_train, y\_train), arbol1.score(X\_test, y\_test)))  plt.grid()  plt.show()  y\_pred = arbol1.predict(X\_test)  mse\_test = np.mean((y\_test-y\_pred)\*\*2)  print('El ECM es {:.6f}'.format(mse\_test))  score = arbol1.score(X\_test, y\_test)  print('El R^2 es {:.6f}'.format(score))  #Random forest  start = timeit.default\_timer()  # SELECCIÓN DEL MODELO (model selection)  max\_depth = np.arange(27, 50, 1)  #min\_samples\_leaf = np.arange(2, 30, 1) SALE 2  n\_estimators = [850,851,852]  # rejilla de valores posibles que queremos explorar  # param\_grid = {'max\_depth': max\_depth, 'min\_samples\_leaf': min\_samples\_leaf, 'n\_estimators': n\_estimators}  param\_grid = {'max\_depth': max\_depth, 'n\_estimators': n\_estimators}  # Validaremos el modelo mediante validación cruzada en n\_folds (este tarda bastante)  n\_folds = 5  grid = GridSearchCV(RandomForestRegressor(random\_state=0),                      param\_grid=param\_grid, cv=5, return\_train\_score=True)  grid.fit(X\_train, y\_train)  print("best mean cross-validation score: {:.4f}".format(grid.best\_score\_))  print("best parameters: {}".format(grid.best\_params\_))  stop = timeit.default\_timer()  print('Time: ', stop - start)  start = timeit.default\_timer()  # SELECCIÓN DEL MODELO (model selection)  # max\_depth = np.arange(25, 70, 1)  min\_samples\_leaf = np.arange(2, 30, 1)  n\_estimators = [852,853,854]  # rejilla de valores posibles que queremos explorar  # param\_grid = {'max\_depth': max\_depth, 'min\_samples\_leaf': min\_samples\_leaf, 'n\_estimators': n\_estimators}  param\_grid = {'min\_samples\_leaf': min\_samples\_leaf, 'n\_estimators': n\_estimators}  # Validaremos el modelo mediante validación cruzada en n\_folds (este tarda bastante)  n\_folds = 5  grid = GridSearchCV(RandomForestRegressor(random\_state=0,max\_depth=28),                      param\_grid=param\_grid, cv=5, return\_train\_score=True)  grid.fit(X\_train, y\_train  print("best mean cross-validation score: {:.4f}".format(grid.best\_score\_))  print("best parameters: {}".format(grid.best\_params\_))  stop = timeit.default\_timer()  print('Time: ', stop - start)  arbol2 = RandomForestRegressor(random\_state=0, max\_depth= 28,                                 n\_estimators = 852).fit(X\_train, y\_train)  y\_pred = arbol2.predict(X\_test)  mse\_test = np.mean((y\_test-y\_pred)\*\*2)  print('El ECM es {:.6f}'.format(mse\_test))  score = arbol2.score(X\_test, y\_test)  print('El R^2 es {:.6f}'.format(score))  #Boosted trees  start = timeit.default\_timer()  # SELECCIÓN DEL MODELO (model selection)  n\_estimators = [1800, 1900,2000,2500]  learning\_rate = [0.025,0.029,0.03, 0.035,0.032]  # rejilla de valores posibles que queremos explorar  param\_grid = {'n\_estimators': n\_estimators,'learning\_rate': learning\_rate}  # Validaremos el modelo mediante validación cruzada en n\_folds (este tarda bastante)  n\_folds = 5  grid = GridSearchCV(GradientBoostingRegressor(random\_state=0),                      param\_grid=param\_grid, cv=5, return\_train\_score=True)  grid.fit(X\_train, y\_train)  print("best mean cross-validation score: {:.4f}".format(grid.best\_score\_))  print("best parameters: {}".format(grid.best\_params\_))  stop = timeit.default\_timer()  print('Time: ', stop - start)  arbol3 = GradientBoostingRegressor(random\_state=0,\*\*grid.best\_params\_).fit(X\_train, y\_train)  error = grid.cv\_results\_['mean\_test\_score'].reshape(len(learning\_rate),len(n\_estimators))  for i, lr in enumerate(list(learning\_rate)):      plt.plot(n\_estimators, error[i],'--o', label='learning\_rate = %g'%lr)  plt.legend()  plt.xlabel('n\_estimators')  plt.ylabel('{}-fold CV R2'.format(n\_folds))  plt.title('train: %0.3f\ntest: %0.3f'%(arbol3.score(X\_train, y\_train), arbol3.score(X\_test, y\_test)))  plt.grid()  plt.show()  start = timeit.default\_timer()  # SELECCIÓN DEL MODELO (model selection)  # n\_estimators = [1800, 1900,2000,2500]  learning\_rate = np.arange(0.03, 0.04, 0.001)  max\_depth = np.arange(2, 10, 1)  # rejilla de valores posibles que queremos explorar  param\_grid = {'max\_depth': max\_depth,'learning\_rate': learning\_rate}  # Validaremos el modelo mediante validación cruzada en n\_folds (este tarda bastante)  n\_folds = 5  grid = GridSearchCV(GradientBoostingRegressor(random\_state=0, n\_estimators = 1800),                      param\_grid=param\_grid, cv=5, return\_train\_score=True)  grid.fit(X\_train, y\_train)  print("best mean cross-validation score: {:.4f}".format(grid.best\_score\_))  print("best parameters: {}".format(grid.best\_params\_))  stop = timeit.default\_timer()  print('Time: ', stop - start)  arbol3 = GradientBoostingRegressor(random\_state=0, learning\_rate = 0.037000000000000005,                                     max\_depth=2,n\_estimators = 1800).fit(X\_train, y\_train)  importances = arbol3.feature\_importances\_  importances = importances / np.max(importances)  # pos = np.where(importances>0.01)  # importances = importances[pos]  indices = np.argsort(importances)[::-1]  indices = indices[:10]  plt.figure(figsize=(10,10))  plt.barh(range(X\_train[X\_train.columns[indices]].shape[1]),importances[indices],color='#28b84f')  plt.yticks(range(X\_train[X\_train.columns[indices]].shape[1]),X.columns[indices])  plt.show()  y\_pred = arbol3.predict(X\_test)  mse\_test = np.mean((y\_test-y\_pred)\*\*2)  print('El ECM es {:.6f}'.format(mse\_test))  score = arbol3.score(X\_test, y\_test)  print('El R^2 es {:.6f}'.format(score))  ##Validación  y\_pred = arbol3.predict(Xval)  mse\_test = np.mean((yval-y\_pred)\*\*2)  score = arbol3.score(Xval, yval)  plt.figure(figsize=(10,5))  plt.plot(yval.ravel(), y\_pred, marker='o', linestyle = '', color = '#28b84f')  plt.plot((0, 1), (0, 1), linestyle='-', color = 'black')  plt.plot((0, 1), (0, 0.85), linestyle='--', color = 'grey')  plt.plot((0, 1), (0, 1.15), linestyle='--', color = 'grey')  plt.xlabel('Valor real')  plt.ylabel('Predicción')  plt.show()  yval = yval.ravel()  error = yval-y\_pred  acierto = abs(error/yval)<= 0.15  plt.figure(figsize=(10, 6))  plt.hist(yval[acierto], bins=25, color='green', alpha=0.7, label='Aciertos')  plt.hist(yval[~acierto], bins=25, color='red', alpha=0.7, label='Errores')  plt.xlabel('Valor real')  plt.ylabel('Frecuencia')  plt.legend()  plt.show()  #SVM  start = timeit.default\_timer()  C = np.arange(197499, 197509, 1)  gamma = np.arange(0.00002, 0.000032, 0.000002)  grid = GridSearchCV(SVR(kernel = 'rbf'),                     param\_grid={'C': C, 'gamma': gamma},                     cv = 5, return\_train\_score=True)  grid.fit(X\_train, y\_train)  print("best mean cross-validation score: {:.6f}".format(grid.best\_score\_))  print("best parameters: {}".format(grid.best\_params\_))  stop = timeit.default\_timer()  print('Time: ', stop - start)  svm = SVR(kernel = 'rbf', \*\*grid.best\_params\_).fit(X\_train, y\_train)  error = grid.cv\_results\_['mean\_test\_score'].reshape(len(C),len(gamma))  plt.figure(figsize=(10,10))  for i, lr in enumerate(list(C)):      plt.plot(gamma, error[i],'--o', label='C = %g'%lr)  plt.legend()  plt.xlabel('gamma')  plt.ylabel('{}-fold R2'.format(5))  plt.title('train: %0.3f\ntest: %0.3f'%(svm.score(X\_train, y\_train), svm.score(X\_test, y\_test)))  plt.grid()  plt.show()  y\_pred = svm.predict(X\_test)  mse\_test = np.mean((y\_test-y\_pred)\*\*2)  print('El ECM es {:.6f}'.format(mse\_test))  score = svm.score(X\_test, y\_test)  print('El R^2 es {:.6f}'.format(score))  start = timeit.default\_timer()  C = np.arange(32, 50, 1)  gamma = np.arange(0.0080, 0.0091, 0.0001)  grid = GridSearchCV(SVR(kernel = 'sigmoid'),                     param\_grid={'C': C, 'gamma': gamma},                     cv = 5, return\_train\_score=True)  grid.fit(X\_train, y\_train)  print("best mean cross-validation score: {:.6f}".format(grid.best\_score\_))  print("best parameters: {}".format(grid.best\_params\_))  stop = timeit.default\_timer()  print('Time: ', stop - start)  svm = SVR(kernel = 'sigmoid', \*\*grid.best\_params\_).fit(X\_train, y\_train)  error = grid.cv\_results\_['mean\_test\_score'].reshape(len(C),len(gamma))  plt.figure(figsize=(10,10))  for i, lr in enumerate(list(C)):      plt.plot(gamma, error[i],'--o', label='C = %g'%lr)  plt.legend()  plt.xlabel('gamma')  plt.ylabel('{}-fold R2'.format(5))  plt.title('train: %0.3f\ntest: %0.3f'%(svm.score(X\_train, y\_train), svm.score(X\_test, y\_test)))  plt.grid()  plt.show()  y\_pred = svm.predict(X\_test)  mse\_test = np.mean((y\_test-y\_pred)\*\*2)  print('El ECM es {:.6f}'.format(mse\_test))  score = svm.score(X\_test, y\_test)  print('El R^2 es {:.6f}'.format(score)) |