

# **Aplicación de ciencia de datos en el sector de producción animal para la predicción y explicación de óptimos en ganado porcino**

03 Abril 2023



**Universidad  
Internacional  
de Valencia**

Titulación:

Máster Big Data y Data Science  
2022 – 2023

Alumno/a:

Cámara Gómez, Jose Eduardo  
D.N.I: 48393337A

Convocatoria:

Primera

De:

Planeta Formación y Universidades

Director/a de TFM: Benjamin Arroquia-Cuadros

## Índice

Resumen .....	6
1. Introducción.....	7
2. Objetivos.....	9
3. Estado del Arte y Marco teórico .....	10
4. Desarrollo del proyecto y resultados .....	11
4.1. Metodología.....	12
4.2. Planteamiento del problema .....	13
4.3. Desarrollo del proyecto.....	14
1.1.1. Resumen del proceso para cada modelo.....	14
1.1.2. Preparación de Datos para regresión .....	42
1.1.3. RandomForest.....	46
1.1.4. LazyPredict.....	50
1.1.5. HistGradientBoostingRegressor.....	52
1.1.6. KNeighborsRegressor .....	54
1.1.7. Evaluación modelo final.....	56
1.1.8. Implementación del modelo para producción.....	57
4.4. Resultados .....	58
5. Conclusión y trabajos futuros.....	59
7. Referencias .....	60
Glosario .....	63
Apéndice I.....	64
1.1. GMD profiling Dataset 01.....	64
1.2. TFM_Preparar_Dataset.....	64
1.3. Regresión RandomForest.....	90
1.4. HistGradientBoostingRegressor.....	111
1.5. KNeighborsRegressor .....	131
Anexos I.....	141

# Índice de ilustraciones

Ilustración 1. Esquema de modelo CRISP-DM. Tomada de: (Ncr & Clinton, 1999).....	11
Ilustración 2. Puntos principales de CRISP-DM. Tomada de (Ncr & Clinton, 1999) ....	12
Ilustración 3. Cronograma de desarrollo del TFM. Elaboración propia .....	13
Ilustración 4. Curva de Crecimiento del Cerdo. Tomada de (www.3tres3.com, 2010).15	
Ilustración 5. Estadísticas generales de dataset inicial. Generado con ydata-profiling.	24
Ilustración 6. Análisis preliminar de variable GMD. Generado con ydata-profiling.....	24
Ilustración 7. Diagrama “boxplot-wiskers” de GMD por tipo ganado. Elaboración propia. .....	25
Ilustración 8. Boxplot para comportamiento de ctos por meses. Elaboración propia ..	26
Ilustración 9. Boxplot para ctos agrupado por años. Elaboracion propia. ....	27
Ilustración 10. Evolución de GMD por meses para los últimos años. Elaboración propia. ....	27
Ilustración 11. Descomposición de la serie temporal. Elaboración propia. ....	28
Ilustración 12. Boxplot con Outliers de GMD. Elaboración propia. ....	30
Ilustración 13. Estadísticas generales de dataset inicial. Generado con ydata-profiling. .....	31
Ilustración 14 - Quitar atributos innecesarios del Dataset Inicial. Elaboración propia ..	32
Ilustración 15. Distribución de valores de GMD. Elaboración propia. ....	34
Ilustración 16. Eliminar outliers de GMD. Elaboración propia .....	34
Ilustración 17. Outliers de DiasMedios en Dataframe inicial. Elaboración propia. ....	35
Ilustración 18. Eliminar outliers de DiasMedios. Elaboración propia.....	35
Ilustración 19. Rellenar valores perdidos de localización GPS de granjas. Elaboración propia. ....	36
Ilustración 20. Rellenar valores perdidos de código postal. Elaboración propia. ....	36
Ilustración 21. Función para simplificar nombres de granjas. Elaboración propia.....	37
Ilustración 22. Corregir nombre de granjas. Elaboración propia.....	37
Ilustración 23. Mostrar reducción de ocurrencias duplicadas de granjas. Elaboración propia. ....	37
Ilustración 24. Eliminar columna de nombre de granja no corregida. Elaboración propia. ....	37
Ilustración 25. Calcular porcentaje de cada sexo en contratos. Elaboración propia. ...	38
Ilustración 26. Añadir campos de semana y año de entrada en contrato. Elaboración propia. ....	38
Ilustración 27. Calcular consumo de pienso por cerda y día. Elaboración propia. ....	39
Ilustración 28. Función para Graficar diferencias del modelo y la realidad. Elaboración propia. ....	42
Ilustración 29. Comvertir en One Hot Encodig. Elaboración propia.....	43
Ilustración 30. Agrupar razas en 8 categorías. Elaboración propia.....	43
Ilustración 31. Selección de variables para regresión. Elaboración propia. ....	43
Ilustración 32. Matriz de dispersión de las variables seleccionadas para regresión. Elaboración propia. ....	45
Ilustración 33. Matriz de correlaciones. Elaboración propia.....	45



Ilustración 34. Correlación de atributos con GMD. Elaboración propia.....	46
Ilustración 35. Diferencias RandomForest v1. Elaboración propia. ....	47
Ilustración 36. Hiperparámetros a optimizar en RandomForest. Elaboración propia. ..	48
Ilustración 37. Búsqueda de hiperparámetros RandomForest. Elaboración propia. ....	49
Ilustración 38. Errores RandomForest con mejores parámetros. Elaboración propia. .	50
Ilustración 39. Resultado de HistGradientBoostingRegressor en dataset del proyecto. Elaboración propia.....	52
Ilustración 40. Rangos de parámetros para la búsqueda aleatoria de la mejor combinación. Elaboración propia.....	53
Ilustración 41. Errores de KNeighborsRegressor. Elaboración propia.....	55
Ilustración 42. Comparativa de valor de R2 para KNN según nº vecinos. Elaboración propia. ....	55
Ilustración 43. Comparativa de R2 para KNN según número de variables usadas. Elaboración propia.....	56

# Índice de tablas

Tabla 1. Columnas con datos faltantes. Elaboración propia.....	29
Tabla 2. Columnas del Dataset tras quitar atributos no necesarios. Elaboración propia. .....	33
Tabla 3. Columnas Seleccionadas para regresión. Elaboración propia.....	44
Tabla 4. Errores de estimación para RandomForest v1. Elaboración propia.....	47
Tabla 5. Características más relevantes según RandomForest. Elaboración propia... 48	
Tabla 6. Mejores parámetros para RandomForest. Elaboración propia.....	49
Tabla 7. Errores de mejores parámetros para RandomForest. Elaboración propia. ....	49
Tabla 8. Ranking de modelos segun LazyPredict. Elaboración propia. ....	51
Tabla 9. Errores de HistGradientBoostingRegressor tras optimizar parámetros. Elaboración propia.....	53
Tabla 10. HistGradientBoostingRegressor mediante earlystopping traas búsqueda de parámetros. Elaboración propia.....	54

# Resumen

El objetivo del presente trabajo busca mostrar la aplicabilidad de las técnicas de la ciencia de datos sobre el proceso de engorde porcino. Para ello se aplica el proceso completo CRISP-DM sobre los datos históricos de los últimos cinco años del engorde de la empresa Cefu S.A. Se pretende estimar el número de animales por semana a varios meses vista que estarán en un determinado peso específico. Se parte desde la definición del objetivo, el estudio de como se calcula actualmente, se analizan los atributos disponibles, su lleva a cabo su selección, limpieza, definición de nuevos campos calculados que puedan ayudar en el problema, su adaptación para ser usados en modelos de regresión, la selección y optimización de los modelos, con los mejores parámetros para estimar el valor a modelar. Finalmente se comparará el mejor modelo obtenido con los valores estimados previamente, se implementará el modelo seleccionado y se mostrará como usar esta previsión en el modelado de KPIs para la toma de decisiones por parte de la empresa. El proyecto pretende demostrar las bondades de la ciencia de datos en la toma de decisiones, de su adaptabilidad y posibilidades para anticipar oportunidades y problemas, de la mejora en el conocimiento objetivo de las características del negocio y su relevancia, y servir para que cada vez se integre en más procesos y ayude a una estrategia basada en datos que permita huir de la arbitrariedad de las decisiones de negocio.

## Abstract

The objective of this paper is to show the applicability of data science techniques on the pig fattening process. For this, the complete CRISP-DM process is applied on the historical data of the last five years for Cefu S.A.'s data. It is intended to estimate the number of animals per week several months ahead that will be at a certain specific weight. It starts from the definition of the objective, the study of how it is currently calculated, the available attributes are analyzed, their selection, cleaning, definition of new calculated fields that can help in the problem, their adaptation to be used in regression models, the selection and optimization of the models, with the best parameters to estimate the value to be modeled. Finally, the best model obtained will be compared with the previously estimated values, the selected model will be implemented, and it will be shown how to use this forecast in the modeling of KPIs for decision making by the company. The project aims to demonstrate the benefits of data science in decision making, its adaptability and possibilities to anticipate opportunities and complications, the improvement in the objective knowledge of the characteristics of the business and its relevance and serve to integrate it into more and more processes and help a data-based strategy that allows to avoid from the arbitrariness of business decisions.

**Palabras clave:** CRISP-DM, engorde porcino, GMD, regresión.

# 1. Introducción

Este trabajo trata sobre la mejora en la predicción del número de animales disponibles a un peso objetivo en el marco de La empresa Cefu S.A., que se dedica a la cría y engorde de Ganado Porcino y Vacuno. Toda su producción va destinada a elPozo Alimentación, empresa del mismo grupo empresarial. La empresa además de la cría de animales también fabrica el pienso con el que los alimenta y dispone de un equipo de veterinarios, ingenieros, expertos en Medio Ambiente, etc. Dispone de todo el personal de apoyo necesario para la gestión integral, el bienestar animal, la trazabilidad y el crecimiento. Está muy comprometida con la implantación de la granja del futuro basada en la Sostenibilidad, Bienestar Animal, Solidez, Seguridad Alimentaria y generación de Empleo Femenino y Rural. Controla la trazabilidad de todo el proceso y promueve líneas de granjas en un Plan Libre de Antibióticos.

Es una empresa estratégica dentro del grupo empresarial, para asegurar el suministro y el control integral de la trazabilidad. Si bien Cefu S.A. no es la única proveedora de ganado porcino y vacuno para la empresa de elPozo Alimentación, que constituye el núcleo y empresa matriz del Grupo Fuertes, sí es la responsable de más de 60% del total del ganado utilizado. La empresa dispone de tres tipos principales de granjas:

- **Granjas de Madres:** Son granjas de cerdas reproductoras y de sus lechones hasta el destete. La mayor parte de estas son granjas propias (39), aunque también hay granjas integradas (11), siendo las de mayor tamaño las propias.
- **Granjas de Transición:** Son granjas de lechoneras, que reciben los lechones recién destetados y los crían durante un corto período de tiempo, hasta recolocarlos en las granjas definitivas. La mayoría son igualmente propias. En algunas granjas de madres tienen una zona de la granja dedicada a Transición.
- **Granjas de Engorde:** Estas granjas son las que reciben los animales hasta alcanzar su peso objetivo y trasladarlas a su destino de procesamiento. Son mayoritariamente granjas integradas. Se entiende por granja integrada la que la granja y el personal de esta no es propiedad de Cefu S.A., pero sí lo son los animales, el pienso, medicamentos y otros envíos de material necesario, personal veterinario, etc. Al integrado se le paga en función de unos indicadores técnicos por alojar los animales, pero todos los datos de la crianza de estos contratos se registran en los sistemas de Cefu S.A. exactamente igual que en las granjas propias. Actualmente hay 489 granjas de engorde integradas.

La empresa cuenta además con un personal propio en las áreas de Veterinaria, Ingeniería, Medio Ambiente, Administración, Fábrica de Piensos, Báscula, Almacén, etc. La parte de los camiones para el transporte del pienso y los animales, están externalizadas a unas empresas que trabajan en exclusiva para Cefu S.A. y cuyos datos están completamente integrados en la empresa, estando dotadas de localizador GPS en tiempo real, para poder controlar aspectos del correcto funcionamiento y ejecución de las planificaciones, que se cumplen los controles sanitarios exigidos, etc. La organización de los viajes en cuanto a las cantidades y horarios de estos camiones

corre a cargo de Cefu S.A. y traslada a las empresas de transporte la orden trabajo que deben cumplir cada día.

El proyecto que se aborda en el actual Trabajo de Fin de Máster consiste en recopilar los datos disponibles de los engordes de ganado de un período de los últimos cinco años, para realizar sobre ellos todo el proceso de limpieza y preparación de estos datos, hasta llegar a aplicar modelos de regresión con el fin de ver si podemos modelizar y prever el valor de la Ganancia Media Diaria de los futuros engordes, para poder adelantar las existencias de animales disponibles a corto y medio plazo. Se pretende también aprender sobre las distintas variables tratadas, su importancia relativa, los fallos en la captura de estos datos y si es necesario o conveniente la incorporación de nuevos datos. También se pretende saber el porcentaje de explicación de la variable objetivo conseguido mediante los métodos aplicados y la fiabilidad de los resultados. El resultado perseguido buscará conseguir una mejor compresión de los datos y un método ya entrenado (aunque de revisión y actualización continua) que permita predecir los valores de la variable de Ganancia Media Diaria.

En este documento se sigue la siguiente estructura, primero se muestran los objetivos a conseguir, el estado del arte del problema y tras este se aplica la metodología CRIP-DM para ir desarrollando los distintos puntos que nos lleven desde la recopilación de los datos disponibles hasta su modelado y entrenamiento de un método de regresión que permita obtener los objetivos marcados, y con este modelizar informes y KPIs para que sean de utilidad en la toma de decisiones estratégicas a nivel de negocio.

## 2. Objetivos

El objetivo principal de este proyecto consiste en mejorar la estimación del crecimiento de los animales en granjas de porcino, con el fin de mejorar la logística y poder planificar a medio plazo los animales disponibles en un peso objetivo.

Se espera que el desarrollo del proyecto también ayude a mejorar la comprensión del negocio, identificando las variables más relevantes, las que producen mayores desviaciones y problemas, etc.

Como objetivos secundarios esperados del proyecto podríamos citar:

- Previsión a medio plazo de las existencias de ganado por semanas.
- Previsión de localización del ganado con el peso objetivo deseado en las semanas siguientes.
- Comprensión de los parámetros que pueden retrasar o presentar problemas en el desarrollo del engorde, para detectar de forma temprana sus desviaciones e intentar corregirlas.
- Presentación de resultados empresariales de los engordes alineados con la misión y visión empresarial.
- Poder estimar la curva de Ganancia Media Diaria de peso corporal de los animales según las características a priori del contrato.

### 3. Estado del Arte y Marco teórico

Para la estimación de los animales en peso objetivo a futuro, necesitamos poder estimar la Ganancia Media Diaria de estos animales, de forma que podamos proyectar la progresión de su peso corporal y finalmente establecer la fecha en la que alcanzarán el peso objetivo. Con este dato sería muy sencillo poder estimar para cada grupo de animales estimar su fecha objetivo y finalmente agruparlos y totalizarlos como se pretendía.

La estimación de la Ganancia Media Diaria (GMD) en ganado porcino es una tarea importante en la producción animal, ya que permite medir el crecimiento y desarrollo de los animales, posibilitando planificar la fecha en la que estarán en un peso objetivo. En los últimos años, se han desarrollado varias técnicas de regresión para estimar la GMD en función de diferentes variables, como el peso inicial, la edad, la raza (Lázaro et al., 2017), la dieta, localización, condiciones ambientales, etc. (Agostini et al., s. f.).

Para realizar esta regresión hay numerosos trabajos que han probado a estimarla haciendo uso de Regresión lineal, polinómica, con árboles de decisión, redes neuronales (Wang et al., s. f.), análisis bayesiano (Lázaro et al., 2017), etc. (Tolosa et al., 2021).

La estimación del GMD no es lineal, sigue una curva de crecimiento también ampliamente estudiada en “papers” científicos del ámbito veterinario (Moughan & Verstegen, 1988) (Campos Benvenga et al., 2022), aunque en el ámbito de este proyecto no es el principal problema, como se detallará más adelante, ya que no se necesita saber el crecimiento de los animales en cada momento de su desarrollo, sino el momento en el que alcanzarán el peso objetivo. Como el peso objetivo es un valor que se mueve en un intervalo muy similar a lo largo del tiempo (dependiendo del tipo de ganado, blanco o ibérico), y también lo es el peso de entrada (si bien hay varios valores posibles dependiendo de la fase del contrato), podemos asumir un GMD lineal para todo el período e intentar en base a una serie de parámetros realizar una regresión que nos ayude a comprender y predecir los valores esperados, y con qué intervalo de confianza.

## 4. Desarrollo del proyecto y resultados

El proyecto se desarrollará usando la metodología CRISP-DM (Ncr & Clinton, 1999), (Wirth & Hipp, 2000) y (Ayele, 2020), de forma que usaremos la plantilla general del método para asegurarnos avanzar sin perder en ningún momento los objetivos tanto del proyecto como empresariales fijados, y armonizar correctamente con la aplicación de métodos de ciencia de datos, a fin de no perdernos, y avanzar de forma predecible, siempre focalizados en los objetivos a conseguir. En la Ilustración 1 se muestra el esquema del funcionamiento general del método y cuáles son sus principales fases, habiendo como se puede apreciar la posibilidad de que al desarrollar una fase descubramos un nuevo requerimiento que no habíamos tenido en cuenta o desarrollado suficientemente y tengamos que volver a una fase anterior que retroalimentaremos y completaremos. De hecho, el método no es un método lineal y que se complete en una única pasada, en la propia vida del proyecto es normal que iterativamente se mejore y se necesite volver una y otra vez a plantear nuevas mejoras y ampliaciones, de forma que permite evolucionar durante la vida del proyecto.

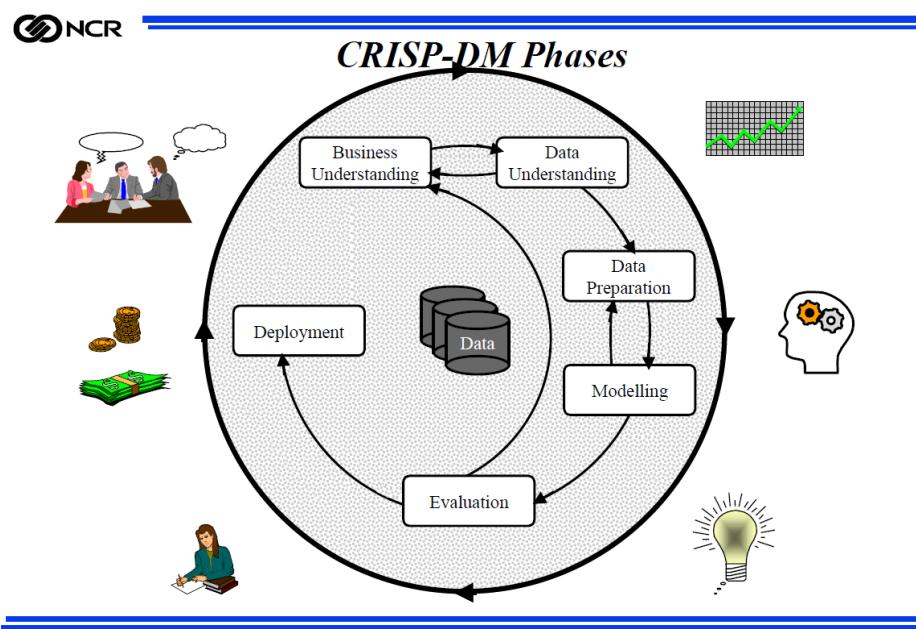


Ilustración 1. Esquema de modelo CRISP-DM. Tomada de: (Ncr & Clinton, 1999)

Para este proyecto seguiremos los puntos citados en el artículo(Ncr & Clinton, 1999), y cuyo resumen esquemático de principales fases y tareas se muestra en la Ilustración 2, del mismo artículo.:

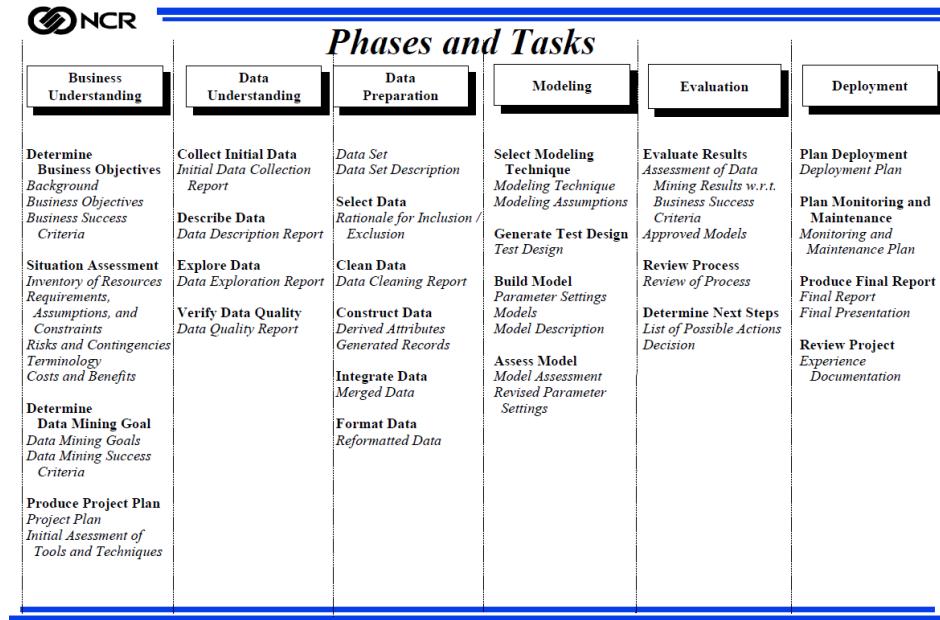


Ilustración 2. Puntos principales de CRISP-DM. Tomada de (Ncr & Clinton, 1999)

En los siguientes apartados del proyecto desarrollaremos cada uno de estos puntos de acuerdo a la idea de negocio y objetivos planteados en el marco del Trabajo Fin de Master..

## 4.1. Metodología

El desarrollo del Trabajo de Fin de Máster (en adelante TFM) se compone de varias fases, como se muestran en el cronograma de la Ilustración 3. Serían:

- Una primera fase de Inicio del TFM. De selección del Tema y del Tutor, y de comprensión de la estructura y plazos del proyecto. En esta fase se investigará sobre el formato, estructura y alcance del proyecto. También se definirán los plazos.
- La siguiente fase buscará terminar de definir el proyecto definiendo de forma consensuada con el Tutor el Título del proyecto y haciéndolo oficial, al subirlo en el Campus de la asignatura.
- La tercera fase, que en realidad comienza antes de terminar la anterior, comienza el desarrollo progresivo de la memoria del TFM, junto con la implementación de toda la recopilación de datos, fuentes y pruebas realizadas.
- Las cuarta y última fase recoge los plazos para presentar el proyecto y finalmente proceder a su defensa pública.

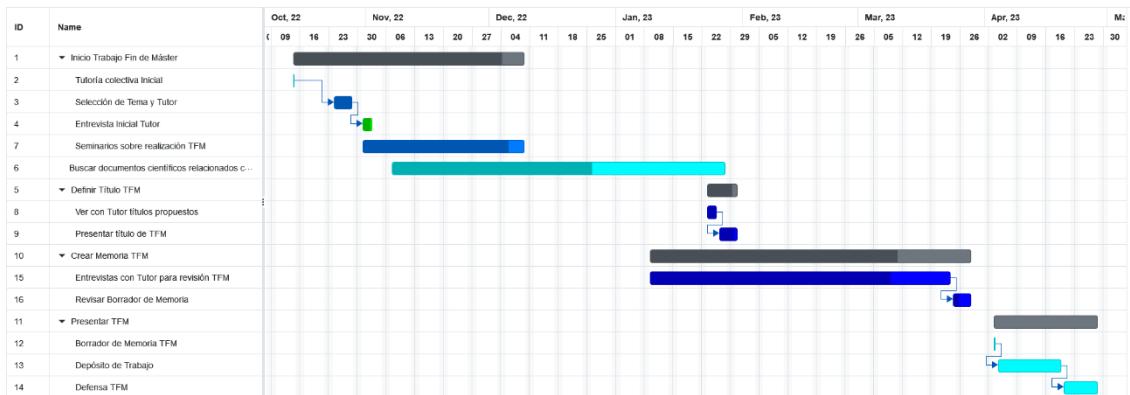


Ilustración 3. Cronograma de desarrollo del TFM. Elaboración propia

## 4.2. Planteamiento del problema

En base al objetivo definido para este proyecto, se establecen otros más específicos como es el de obtener un estimador del valor de la Ganancia Media Diaria de peso en ganado porcino en contratos de la empresa Cefu S.A., mediante el uso de los datos históricos de contratos anteriores y de técnicas de ciencia de datos.

Actualmente en la empresa ya se intenta estimar los valores de GMD en función de los datos históricos, pero se hace de una forma muy básica, con uso de una hoja de Excel que simplemente busca en el valor medio de engordes de los dos últimos años agrupados por raza. Se obtiene un valor predicho con esa aproximación, pero tampoco se conoce ni mide la desviación que se obtiene con el mismo. Con esos datos se intenta hacer una planificación de los contratos en curso para estimar el número de animales disponibles por semana hasta 8 o 10 semanas vista desde la fecha de cálculo, y cada dos semanas se van actualizando los datos con los nuevos datos productivos. Un equipo humano de visitadores va revisando las granjas que están “teóricamente” cerca de alcanzar su peso objetivo, para verificar que esto es así y planificar su recogida.

El departamento que elabora los cálculos y previsiones citadas forma parte de dirección de Cefu S.A. y se encarga de planificar junto a elPozo las recogidas previstas para las siguientes semanas, unas cuatro semanas actualmente, aunque se pretende poder aumentar el período estimado.

Pertenezco al departamento de Ingeniería de la empresa, y he participado activamente en el diseño de la base de datos y programas de captura para la información recogida en los contratos de Engorde, junto con un equipo de 3 desarrolladores del Grupo corporativo Grupo Fuertes. Este proyecto se enmarca en la utilización de la ciencia de datos para la ayuda en la toma de decisiones estratégicas basadas en datos.

La propuesta del proyecto consiste en abordar todo el proceso desde la recopilación de los datos, su comprensión, idoneidad y preparación, hasta la modelización del

problema y sus resultados. Finalmente, la implementación del modelo y la generación de informes que lo aprovechen.

Uno de los objetivos que se persiguen serían los de aprovechar la gran cantidad de datos históricos para optimizar los parámetros productivos de la granja, de forma que se tenga un mejor conocimiento de los factores que afectan a la crianza de los animales, a su crecimiento, a posibles problemas, etc. Es por este punto por el que se pensó la aplicación de ciencia de datos, se planteó a la dirección de la empresa, como un objetivo sencillo que puede mostrar la potencialidad del uso de este tipo de métodos para solucionar problemas, mejorar el conocimiento del negocio y las decisiones tomadas de acuerdo a datos objetivos y medibles. Se pretende que este proyecto sea el primero de otros muchos que ayuden a que las decisiones estratégicas cada vez estén más focalizadas en el uso de la información y permitan anticipar tendencias, crear productos innovadores basados en lo que nos revelan los datos que realmente está funcionando, reaccionar más rápido a los cambios y detectar oportunidades y problemas antes que estos terminen de hacerse evidentes.

## 4.3. Desarrollo del proyecto

### 1.1.1. Resumen del proceso para cada modelo

#### *Conocimiento del negocio*

#### **Objetivos de Negocio**

Se desea mejorar la previsión de número de animales disponibles en fechas futuras para un peso objetivo dado, para poder anticipar la oferta de animales a servir a elPozo, así como poder organizar la carga de estos animales, pues con la previsión deseada se sabría de antemano las ubicaciones en las que deben estar disponibles y se puede organizar mejor la carga de camiones para su traslado. En el proceso de carga interviene cierto personal de la empresa a organizar, como son los marcadores (personas que seleccionan los animales de entre el total del contrato que se llevarán), los cargadores (ayudan a subir al camión los animales), camioneros, personal de la granja, etc. Organizando y optimizando las cargas se puede obtener grandes mejoras en los tiempos de trabajo, así como ajustar mejor los animales seleccionados y no llevarlos pasados o faltos de peso, por no haber calculado correctamente el peso al que estarían.

Como parámetro crucial para poder estimar la fecha a la que unos animales concretos estarán en el peso deseado aparece el índice de Ganancia Media Diaria, que es un indicador de cuanto se incrementa el peso de los animales cada día. Conocer este parámetro permite proyectar los animales a la fecha que tendrán el peso deseado. Este parámetro no es lineal, y sigue una curva de crecimiento promedio como la de la Ilustración 4, no obstante, y dado que no estamos en realidad interesados en saber

exactamente el peso de los animales en cualquier momento de su crianza, si no del peso a la salida y siempre por los mismos rangos de peso (dependiendo del tipo de ganado), no habría problema en reducirlo a un valor lineal para el total del engorde que nos daría una muy buena aproximación.

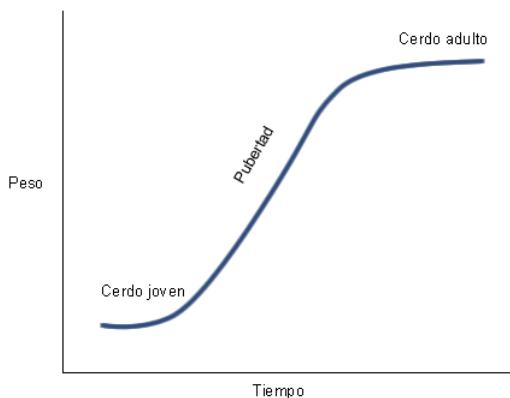


Ilustración 4. Curva de Crecimiento del Cerdo. Tomada de ([www.3tres3.com](http://www.3tres3.com), 2010)

Tras saber la fecha a la que cada grupo de animales estaría en peso objetivo podemos saber las existencias disponibles semana a semana, y adicionalmente las localizaciones de estos, de cara a poder organizar las cargas en pro de optimizar el tiempo de desplazamiento para realizar el trabajo.

Para valorar si son correctos los animales predichos por contrato, previo a la carga, unos 15 días antes se enviará a personal de la empresa a supervisar a los animales y corroborar que se encuentran en el peso objetivo indicado, así mismo entre 1 o 2 días antes de la carga volverán a visitar la granja para marcar los animales seleccionados de entre los disponibles en el contrato. En estos procesos se pueden detectar errores en la previsión e intentar solucionarlos seleccionando los siguientes contratos de la siguiente semana, por si alguno se hubiera adelantado. En estos casos, tener la planificación de la semana actual y de las siguientes resulta una gran ventaja de organización y trabajo, permitiendo una rápida respuesta ante incidencias. Un ejemplo son los problemas de accesos de los camiones a la granja a causa de contingencias meteorológicas, como nevadas o fuertes lluvias que hagan impracticable el acceso durante algún día.

### Criterios de éxito

Para medir el éxito de los objetivos anteriormente descritos, habría que valorar a la finalización de los contratos de engorde el índice de Ganancia Media Diaria real y el que se había predicho. Se considerará un éxito si hay una diferencia menor a 2 centésimas en valor absoluto del GMD estimado frente al real, para al menos un 95% de los contratos. La diferencia de 2 centésimas en el valor de GMD para un valor promedio de 0.85 equivale a un error del 2,35%.

En cuanto al número de animales por semana se espera obtener una precisión del 98%, aplicando la proyección de los días del contrato al GMD estimado. Con ese GMD y la fecha y peso de entrada se podrá saber a qué fecha está en peso objetivo y acumular por semanas los animales de los distintos contratos que cumplan la condición. Se desea mantener una precisión en el número de animales cercana al 98% para contratos a recoger entre 4 y 6 semanas desde la fecha actual (con la que se obtuvieron los datos a pasar al modelo). Es tolerable un error superior para previsiones a más largo plazo (en las que todavía los datos recopilados no son completos).

Para medir la desviación del número de animales a nivel de granja, podemos hacer uso de los informes recogidos por los visitadores de los animales en los días previos a la carga, que indiquen si los animales están por encima o debajo del peso predicho. Se desea en este porcentaje de animales fuera del intervalo de peso deseado no sea superior al 1%. Estos animales se refieren a aquellos contratos que ya están muy cerca de recogerse, es decir a tan sólo dos o tres semanas de su peso objetivo.

### **Objetivos de Minería de Datos**

El principal objetivo que nos debería llevar a acercarnos a los objetivos de negocio pasa por estimar el valor de la Ganancia Media Diaria para cada contrato de los disponibles y los que se abran en el futuro. Esta medida se expresa en Kilos por día, y su valor promedio para un cerdo blanco podría estar en torno a 0,85 Kg/día. La función de pérdida a utilizar para estimar la mejor regresión será la de la raíz del error cuadrático medio, y se buscará minimizarlo. Se buscará usar este indicador del error por expresarse en las mismas unidades y magnitud que la variable a predecir (GMD), y porque permite minimizar todos los errores, pero penalizando más a los valores más alejados. Para el método seleccionado también se calcularán otros estimadores como el error máximo, para estudiar su casuística y ver por qué se produce. Se está muy interesado en poder predecir con la mayor exactitud posible este valor, pero también en el proceso se desea poder ser consciente de la exactitud esperada de esta medida y del intervalo de confianza que nos pueda acercar a la certidumbre deseada.

Se espera del análisis de los datos poder comprender mejor los factores que afectan al crecimiento de los animales y así también poder sacar conclusiones de qué puede influir positivamente en este aspecto para buscar potenciarlo, o intentar evitar los aspectos que afecten negativamente al proceso.

## Planes de Proyecto

Para conseguir los objetivos marcados se seguirá la metodología CRISP-DM, se utilizarán principalmente los datos del ERP de la empresa, se hará un proceso ETL de los mismos, un análisis descriptivo y se buscará una regresión por distintos métodos, con el fin de encontrar aquellos que nos acerquen con mayor certidumbre al valor real del parámetro. Finalmente se recogerán los hallazgos del estudio, se creará un despliegue que permita utilizar los resultados obtenidos para nuevos contratos, y se generarán gráficos que hagan uso de estos valores para mostrar los resultados obtenidos y que ayuden a alcanzar los objetivos de negocio.

### *Comprendión de los Datos*

#### **Recolección inicial de datos**

Los datos usados para este proyecto se sacarán de la base de datos del ERP de la empresa. Es una base de datos SQL Server con históricos de los contratos de la empresa desde el año 1996. Para este trabajo se han usado datos de los contratos finalizados en los últimos 5 años, ya que se ha considerado más veraz hacer uso únicamente de estos y no remontarnos más atrás dada la evolución que han sufrido los procesos y tecnologías en las granjas en estos últimos años, con la inclusión de nuevas razas, de nuevas medidas de bienestar animal, con tipos de piensos más adaptados, con mejores sistemas de alimentación, control de clima, mejor control veterinario, contratos libres de antibióticos, etc.

Los datos que vamos a utilizar en la empresa están referenciados por un código de contrato. Un Contrato hace referencia a un grupo de animales con las mismas características de tipo de ganado, raza, edad y sexo, que se crían en un mismo período de tiempo, en una misma granja y con unas condiciones sanitarias, de alimentación, de instalaciones o personal que los atiende similares. Un contrato es único en la empresa e identifica a estos animales desde su entrada hasta su salida de la granja. Todos los movimientos de los animales, del pienso, medicamentos, incidencias, controles, etc. vendrán referenciados por este contrato con distinta granularidad. Al cierre del contrato y atendiendo a todos los datos grabados durante su desarrollo se puede concretar los datos reales del mismo, incluido el de Ganancia Media Diaria, los costes económicos, los beneficios obtenidos por animal, y una gran cantidad de indicadores de la marcha del contrato a poder usar para la mejora y optimización de procesos y resultados, así como para medir el desempeño de los KPIs que nos deberían acercar a los objetivos estratégicos marcados.

Los datos disponibles son:

- Contratos: Para la apertura de cada nuevo contrato se recopilan los principales datos que definen a los animales de este, tales como son el tipo de Ganado, Sexo de los animales, raza, tipo de alimentación, granja e integrador a los que están asociados, etc.
- Granjas y Naves: Identifican las características de las instalaciones en las que se crían los animales, tales como su código REGA, localización geográfica, dirección, la capacidad de animales que tiene autorizada, el número de naves, veterinario principal asociado, etc.
- Propietario: Identifica si la granja es propiedad de Cefu S.A. o de algún Integrado. Aglutina los principales datos del propietario y es referenciada por los contratos, facturas, etc.
- Historial de los animales: Se lleva un detalle de las principales eventualidades diarias del contrato, siendo los principales tipos de movimientos registrados: las entradas de animales al contrato, los envíos de pienso, medicamentos u otros materiales, las visitas de veterinarios o personal de Cefu S.A., los chequeos de bienestar o de instalaciones, grabación de alguna incidencia, las bajas de algún animal y las recogidas de animales (para sacarlos de la granja).
- Alimentación de los animales: Se lleva un registro de todos los piensos enviados a cada contrato. El pienso es fabricado en las fábricas de la propia empresa y se tiene una trazabilidad total de su composición, fabricación, reparto y consumo del mismo. En algunas granjas se dispone de alimentación inteligente que permite incluso controlar y llevar detalle de la cantidad consumida diariamente por cada animal (identificado por un chip único alojado en un crotal).
- Incidencias: Se lleva registro de las posibles incidencias que acontecen en el contrato, tales como algún problema sanitario o necesidad de aplicar algún tratamiento, los tipos y cantidades de piensos usados en el contrato, si hay algún tipo de traslado o cambio en el contrato etc.
- Medicamentos: Se lleva un control detallado de los medicamentos utilizados, tanto internamente como notificado a las distintas autoridades de cada comunidad autónoma. Hay contratos adscritos a un Plan Libre de Antibióticos. La medicación se puede aplicar en pienso medicado, o por otras vías como puede ser inyecciones, bebederos o tópica.
- Envíos de Materiales Varios: Se registra también los materiales usados en la granja ya sean para los trabajadores de las mismas (como pueden ser monos, botas, ...), para las instalaciones (detergente, papel higiénico, bombillas, ...), para los animales (juguetes antiestrés, agujas, ...). Todo esto para llevar un control de gasto o poder sacar conclusiones si determinados materiales mejoran significativamente la marcha del contrato.
- Datos de Sensores: Cefu S.A. está cada vez más comprometida en la creación de la granja del futuro y esto también pasa por la inclusión de tecnología, que mejore las condiciones, control y conocimiento de la granja en tiempo real. La introducción de las nuevas tecnologías está siendo gradual y hay abiertas muchas líneas de investigación de cara a poder ir extendiendo las más provechosas al resto de las granjas. Las granjas de madres propias son las que más proyectos albergan, pero la intención es que muchos de estos avances se ter-

minen llevando a todas las granjas. Ejemplos de tecnologías serían: sensores de clima, con control inteligente de apertura de ventanas, humidificadores o ventiladores, control de apertura de puertas, estocaje de silos de pienso, contadores de agua o gas, cámaras de conteo de animales, de estimación de peso, de averiguar estado de la cerda (si está en celo, si está gestante) por sus movimientos y comportamientos mediante reconocimiento de imágenes, animales identificados por chip para su trazabilidad individualizada, etc.

## Descripción de los Datos

De los datos indicados anteriormente se han seleccionado, basado en conocimiento experto por parte de personal de la empresa, aquellos datos que a priori parecían más relevantes para poder estimar los parámetros indicados en los objetivos empresariales, básicamente se fija como primer objetivo que posibilitará desarrollar el resto, la estimación de la Ganancia Media Diaria. Se dispone de los datos históricos de una enorme cantidad de contratos desde hace más de 25 años. No obstante, incluir datos tan antiguos puede influir negativamente en la estimación, ya que los métodos, tecnología, conocimiento, control y alimentación actual difiere mucho a la que se utilizaba anteriormente, y en consecuencia los índices obtenidos. Sería más adecuado usar los últimos 5 años, que además son los que se refieren a la mayoría de las razas y genéticas utilizadas actualmente. Los datos serán agrupados y resumidos a nivel de contrato, integrando las distintas tablas de movimientos detallados en un único origen de datos con el que poder trabajar.

Los datos con los que contaremos son:

- **Relativos a la apertura del contrato.** Un contrato se define como un grupo de animales que se tratan como una unidad. Todos los animales del contrato comparten características similares de genética, sexo, peso, tipo de alimentación, etc. Todos entran en una misma granja, en fechas similares (la entrada de los animales del contrato no difiere más de 2 o 3 semanas, para contratos de un tamaño mediano y no más de 1 mes para contratos más grandes. Los animales se van recogiendo del contrato a una fecha determinada y con unas semanas de plazo para ir sacando primero a los animales con un peso más adelantado y los últimos los que han ido engordando más lentamente, pues hablamos de biología y aun con las mismas características ambientales y animales de similares características, no todos se comportan exactamente igual. Los principales datos del contrato a tratar son:
  - Número de Contrato. Es el identificador del contrato, de tipo numérico, se usa para referenciar el resto de los movimientos asociados al contrato. Referentes a los animales: entradas, bajas, recogidas. O referentes a envíos de pienso, de medicamentos, a informes de la granja o los animales, etc. En nuestro dataset dispondremos de una fila para cada contrato, agrupando los datos en virtud de este.

- Fecha de entrada de los primeros animales. Es la fecha en la que entran los primeros animales, de tipo fecha.
- Fecha de entrada de los últimos animales. Para poder distinguir los contratos que por algún motivo se demoran más de la cuenta en su llenado, y si esto puede indicar alguna diferencia en los animales. Por entrar con pesos distintos o estar unos un tiempo significativamente mayor que otros, etc.
- Número de animales del contrato. Por si los datos reflejan diferencias por el tamaño de del contrato, o que las granjas de mayor capacidad difieran de las más pequeñas.
- Integrador. Identificador del encargado de llevar la granja, por si los datos pudieran identificar diferencias entre distintos encargados y el manejo que hacen de los animales.
- Granja. Identificador de la granja, para poder agrupar y valorar más los datos de otros contratos en la misma granja, frente a otras con distinta climatología, por ejemplo, o con distinto tamaño, condiciones, etc.
- Tipo de Ganado. Hay dos grandes tipos de ganado, el cerdo blanco y el ibérico, con datos muy distintos de GMD y tiempos de engorde. Por lo que podría llegar a ser una buena idea el separar ambos si el algoritmo no trata adecuadamente esta diferencia. Además, está desbalanceado el número de contratos y de animales de ambos tipos, siendo más del 90% de los mismos de cerdo blanco, por lo que en el caso de optar por un método conjunto habría que trabajar en esto para que no tienda a ajustarse al cerdo blanco mejor que al ibérico, por el tamaño de la muestra. Este dato sería un enumerado con los 2 posibles valores.
- Sexo de los animales. Los contratos pueden ser de animales separados o no por sexo. Disponemos de contratos de sólo hembras, sólo machos, o mixtos, además de contratos de machos castrados, que se comportan como las hembras. Pudiendo la proporción de machos y hembras, pura, al 50% o al 33% y 66%. Este dato sería un enumerado. Es un dato que se empezó a dar importancia recientemente y hay muchos contratos históricos a revisar su dato atendiendo a los datos de facturación, que sí vienen desglosados. Habría que corregir este valor en los contratos antiguos al realizar la carga del dataset. El GMD de los distintos sexos difiere y por eso es relevante indicarlo.
- Raza. La raza se refiere a la genética de los animales, hay varias genéticas y cada cierto tiempo se va probando con alguna nueva genética. Las distintas razas tienen parámetros distintos con respecto al objetivo a buscar, no tan diferentes como en el caso del tipo de ganado, pero sí relevantes. Por lo tanto, es de suponer que el algoritmo valorará este dato para ver como ponderar el GMD esperado. Es una enumeración de los distintos valores que hay actualmente.

Tipo de alimentación. Hay 2 tipos principales de alimentación, la alimentación líquida y la normal.

- Relativos a la marcha del contrato. A lo largo de la vida del contrato, se van produciendo movimientos en el mismo, referentes a los animales. Estos datos no están disponibles al abrir el contrato y no están completos hasta la finalización de este. Por lo tanto, no se pueden utilizar para contratos recién abiertos, pero sí pueden ser un buen corrector para mejorar la precisión de los cálculos conforme vamos teniendo más datos de los mismos. Por ejemplo, si un contrato tiene más o menos bajas, puede aflorar un problema sanitario que nos haga pensar que no seguirá un crecimiento normal, y que este se verá retrasado mientras se resuelve el mismo, sería lógico comparar el GMD esperado con los contratos que han tenido un comportamiento similar de estas, y no con los que se han comportado de forma diametralmente opuesta.
  - Peso de entrada. No todos los contratos comienzan con los animales en el mismo peso. Este dato se usará para calcular el GMD, pero también puede marcar una diferencia en el comportamiento del contrato que los animales entren más pequeños y por tanto estén más tiempo en el contrato, o que entren más crecidos.
  - Peso de salida. Este dato no está disponible hasta que comienzan a salir los animales del contrato, ya en las últimas semanas. Es fundamental para el cálculo del GMD, pero no podremos usarlo en los contratos aún abiertos, que son los que querremos analizar. Si se podrá poner para estos un peso objetivo estimado, que va cambiando a lo largo del tiempo.
  - GMD. La Ganancia Media Diaria es el valor que queremos modelar y predecir. Es un dato que únicamente disponemos tras cerrar el contrato y sacar todos los animales de este. Es el valor para el que entrenaremos al algoritmo, para el que validaremos y nuestro objetivo para los contratos aún activos. Es importante revisar este valor porque en contratos con anulaciones o correcciones pueden aparecer valores anómalos que hay que corregir.
  - Bajas. Número de bajas producidas en el contrato. Puede ser más conveniente transformar estas para que sean en porcentaje sobre los animales del contrato, para poder compararlas de forma más sencilla.
  - Fecha de bajas. No es lo mismo las fechas en las que se producen las bajas, y si estas son a principio del engorde, al final, de forma homogénea o puntual. Este dato es más complejo de tratar y comparar, por lo que para una fase inicial igual no lo contemplaría, pero se dispondría del mismo por si se necesitara refinar más sobre el mismo. Se podría utilizar el indicador anterior proyectado a las semanas de engorde que aún le faltan, esto es si estamos en las primeras semanas se incrementaría más el número de bajas que si ya estamos casi al final del engorde.

- Tratamientos veterinarios. Todos los envíos de medicamento, las posibles enfermedades o problemas sanitarios detectados por los veterinarios en sus visitas periódicas o solicitadas por problemas puntuales, quedan registradas y se podría hacer uso de ellas para tener una información más rica y que definiera mejor los datos a tratar.
- Deficiencias detectadas. Los veterinarios y auditores realizan visitas periódicas a las instalaciones para valorar el estado de la granja y de los animales, puntuando y destacando las deficiencias a subsanar y cómo y cuándo se lleva a cabo la citada subsanación. De nuevo estos datos son muy extensos el cuestionario es de más de 100 puntos, y por tanto no lo hemos incluido en esta fase del problema, no obstante, también se resume con una puntuación y un indicador de apto o no apto, que sí se podría añadir si los datos anteriores no arrojan los resultados esperados. De todas formas, como la aplicación de estos cuestionarios es relativamente nueva, no nos valen para los contratos históricos, y por ello de momento optamos por dejarlos fuera de la fase inicial de datos del dataset.
- **Relativos a la alimentación de los animales.** La cantidad de pienso enviado y consumido por los animales, puede ser un buen indicador sobre si estos están engordando más rápidamente o menos, si hay algún problema de cualquier tipo es de esperar que los animales coman menos y la cantidad de pienso gastado por animal aflore que algo está sucediendo. Se dispone de los datos por kilos y fecha de envío hacia la granja. Adicionalmente se están empezando a instalar sensores para estimar la cantidad de pienso en cada silo, para saber en tiempo real el consumo. Estos datos se deberían transformar para poder compararlos de forma más eficiente entre contratos, para ello la propuesta es hacerlo ponderado por animal, y por semana de engorde del contrato (desde la entrada de los animales), de esta forma se podrían comparar los datos entre diferentes contratos. Estos datos al igual que los anteriores no están disponibles al inicio del contrato, pero son una excelente medida de corrección de los datos esperados del contrato conforme va avanzando la vida del mismo, siendo mucho más significativos cuando llevamos al menos 5 o 6 semanas del mismo, porque al inicio del contrato se envía un envío inicial de alimento, pero hasta que se realizan los siguientes envíos no sabemos realmente el ritmo al que se está consumiendo el mismo (al menos hasta tener desplegados los sensores de los silos en todas las granjas). Los datos que incorporar por contrato serían:
  - Kg Pienso/animal\_semana\_1
  - Kg Pienso/animal\_semana\_2
  - Kg Pienso/animal\_semana\_3
  - Kg Pienso/animal\_semana\_...
  - Kg Pienso/animal\_semana\_n
- Relativos a la granja del contrato. Disponemos de datos relativos a la granja, tales como el tamaño, la localización GPS de la misma (que podríamos usar para enlazar más datos como temperaturas, humedad, etc., o para agrupar por

localizaciones cercanas). El nombre de la granja, Valoración de las instalaciones, antigüedad de la granja, tiempo con la empresa, datos de contratos anteriores, etc.

- Nombre de la granja. Aunque partimos de una BBDD relacional y la mayoría de los datos están normalizados, con las granjas hay un problema, porque esta está identificada por el integrador y código de granja, de forma que la misma granja si a lo largo de su existencia ha facturado a distintos integradores (por ejemplo, al marido primero, en el siguiente engorde a su esposa o a sus hijos, o ha cambiado de dueño, etc.) puede aparecer con distintos nombres, por ejemplo unos con artículos y otros sin ellos, uno todo en mayúsculas y otro no, alguno renombrando añadiendo OLD o BAJA al nombre para no volver a utilizarlo por error, etc.
- REGA. Es el identificador de la granja, y puede servir para detectar granjas idénticas. Este dato no se usaba en los primeros años, por lo que no todas las granjas lo tienen relleno y habría que completarlo o corregirlo, sobre todo si se usan datos antiguos.
- Latitud y longitud. Estos datos permiten localizar las granjas y agruparlas por climas similares si son suficientemente cercanas. Pueden aparecer granjas sin este dato, que hay que ir corrigiendo, añadiéndoles su localización y teléfono de contacto.
- Población y código postal. Quizá sea más sencillo trabajar con estos datos para agrupar las granjas cercanas y de climatología similar, que con la localización GPS.
- Provincia: No todas las provincias tienen las mismas exigencias a nivel de documentación y requisitos de funcionamiento.
- Animales autorizados: Define el tamaño de la granja.

## Exploración de los Datos

Partimos de la carga de un Dataset inicial, como ya se comentó resumiendo los datos seleccionados a nivel de contrato. Las columnas que usaremos serán: ct\_codigo, ct\_integra, ct\_granja, ct\_nave, ct\_tipo, ct\_raza, ct\_fase, ct\_sexo, ct\_ali\_liquida, ct\_tipo\_ali, IncPeso, DiasMedios, GMD, EntradaInicial, EntradaFinal, NumAnimales, na\_nombre, na\_rega, se\_nombre, PesoEntMedio, PesoRecMedio, NumBajas, GPS\_Longitud, GPS\_Latitud, gr\_direccion, gr\_codpos, gr\_poblacion, kgTotalPienso.

Para iniciar el análisis exploratorio de los datos comienzo lanzando sobre el dataset inicial cargado en pandas la utilidad “ydata-profiling”, que automatiza una generación inicial de gráficos y estadísticos sobre el dataset para ayudar a iniciar el análisis exploratorio de los mismos. El reporte generado se puede exportar a HTML, adjunto el reporte completo obtenido como anexo (GMD profiling Dataset 01) al proyecto. Y paso a analizar lo más relevante de este análisis (Ilustración 5).

El primer vistazo general se muestra en la Ilustración 5:

Dataset statistics		Variable types	
Number of variables	27	Numeric	17
Number of observations	5332	Categorical	9
Missing cells	996	Boolean	1
Missing cells (%)	0.7%		
Duplicate rows	0		
Duplicate rows (%)	0.0%		

Ilustración 5. Estadísticas generales de dataset inicial. Generado con ydata-profiling.

Como se aprecia en la Ilustración 5 el dataset inicial contiene 5332 filas, correspondientes ese mismo número de contratos distintos, para un total de 27 variables, y 996 valores perdidos (que corresponden a 0.7%), no hay filas duplicadas.

La variable que se corresponde con el resultado que queremos prever es GMD, cuyo resumen mostramos en la Ilustración 6, extraída del documento generado en el anexo GMD profiling Dataset 01.

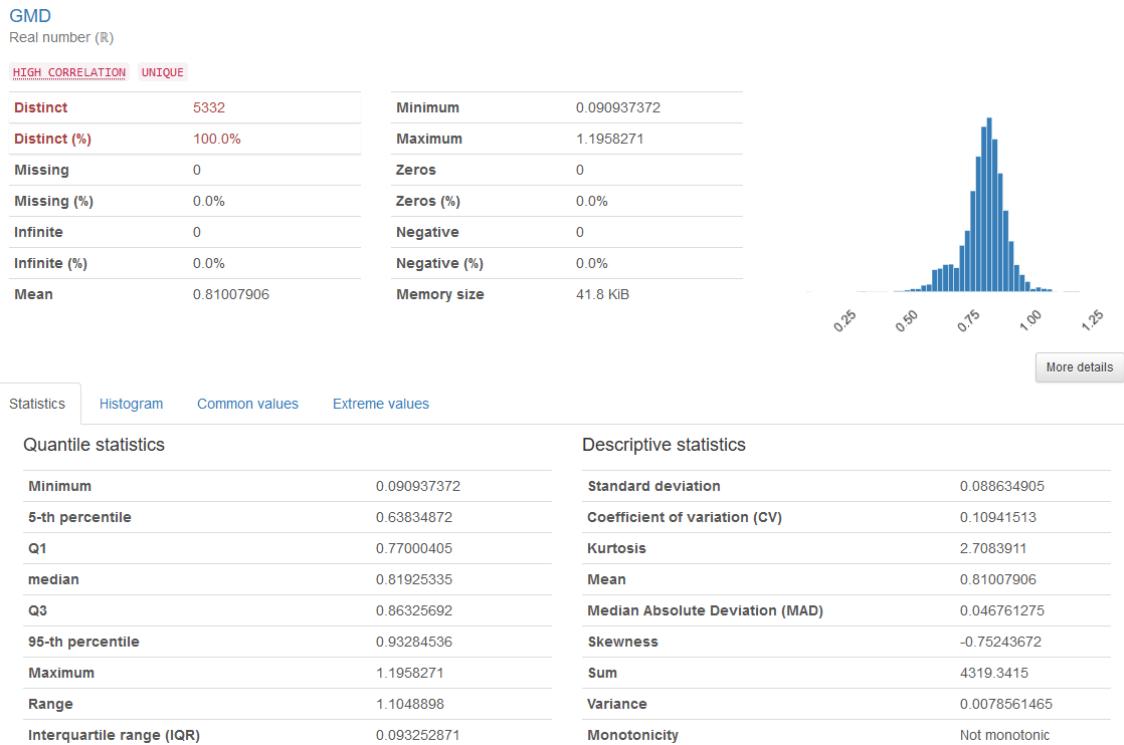


Ilustración 6. Análisis preliminar de variable GMD. Generado con ydata-profiling.

Del análisis preliminar de estos datos y tras observar el histograma, apreciamos que se comporta como la suma de dos distribuciones normales desigualmente balanceadas, pareciendo corresponderse a los dos tipos de ganado que hay (blanco=1 e ibérico=2). Para verificar esto, si se separa el GMD por tipo de cerdo como vemos en la Ilustración 7, que muestra un gráfico “Boxplot” (caja con bigotes), se puede apreciar como la distribución del GMD en ambos casos es muy diferente.

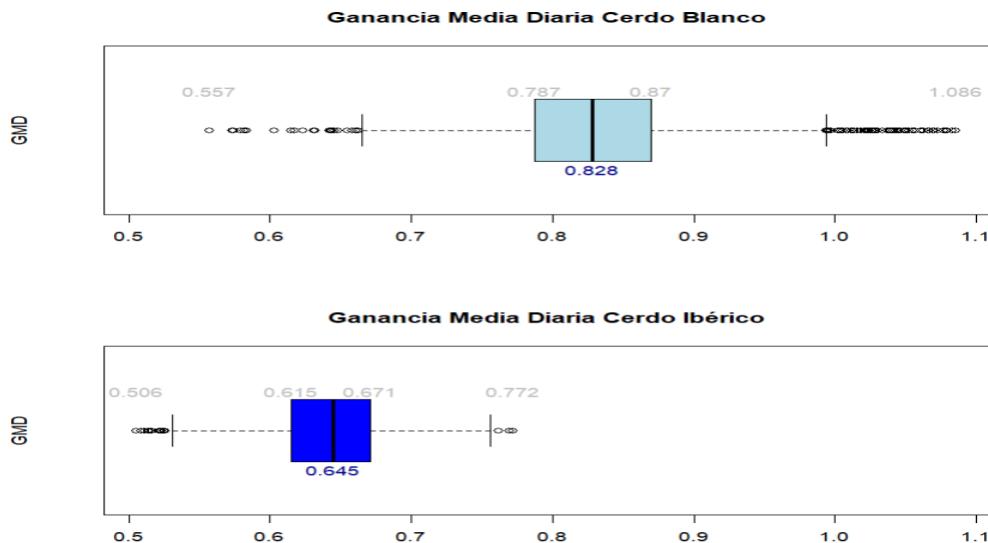


Ilustración 7. Diagrama “boxplot-wiskers” de GMD por tipo ganado. Elaboración propia.

A la vista de lo comentado esta variable, aunque únicamente tenga 2 valores es sumamente importante, y puede ser un problema su desbalanceo, pues el número de filas del ganado ibérico se corresponde únicamente con el 10%. Ante esta problemática se podría intentar balancear, analizar por separado ambos tipos, creando dos “datasets” uno para ganado blanco y otro para ganado ibérico, no hacer nada si los métodos son capaces de tratar correctamente estas diferencias. Se verá con la experimentación si son necesarias estas medidas y cuál de ellas aplicar.

### Atributos por eliminar del dataset

De cara al análisis que pretendemos realizar hay atributos del dataset inicial que podríamos eliminar porque no aportan mucho o incluso nada en absoluto a lo que pretendemos estimar. Queda para la siguiente fase analizar el detalle de las columnas que se podrían o incluso se deberían eliminar. Serían las que dispongan de un único valor, las que tengan muchos valores perdidos, las que no expliquen nada sobre la variable a estudiar, etc. En este último caso hay que ser especialmente cuidadoso, pues que no sea evidente su influencia no implica que en conjunción con otras variables pueda llegar a ser útil en la regresión.

### Añadir nuevos atributos

De un análisis de los atributos disponibles del dataset parece procedente añadir los siguientes campos calculados para facilitar su uso por parte de los algoritmos de regresión, aflorando patrones que no resultaban tan evidentes: semana y año de entrada en los contratos, porcentaje de hembras en contrato y kilos de pienso por animal y día.

De los anteriores datos hay algunos que sería interesante calcular a partir de los que tenemos para facilitar la búsqueda de patrones. Por ejemplo, se observa que hay diferencias en la estacionalidad, y no hay un mismo crecimiento en las distintas semanas del año, por lo que, de la fecha de entrada del contrato, podría ser interesante quedarnos con la semana del año de la misma o el mes, de esta forma podríamos descubrir patrones de estacionalidad en los datos, según la época del año, que quizás no sean los mismos en las distintas localizaciones, por ejemplo veranos muy calurosos en algunas zonas o inviernos más fríos en otras. También podría ser interesante quedarnos con el año por si hay que ponderar más los últimos años que los más antiguos, por las mejoras en todos los ámbitos introducidas en las granjas y los procesos de manejo y cuidado de los animales.

En el marco del proyecto se ha hecho un pequeño análisis del comportamiento temporal de los datos de los contratos de engorde para los últimos años, el código se puede observar en [TFM serie temporal.R](#). Aquí mostraré los principales gráficos de este comportamiento. Como se aprecia en la Ilustración 8 parece que el dataset esconde una clara componente estacional para los distintos meses del año, y en la Ilustración 9 parece apreciarse una tendencia interanual, que comprobaremos graficando los datos de la serie completa (Ilustración 10) y descomponiendo la serie en sus componentes principales (Ilustración 11) para comprobar su tendencia, estacionalidad y resto de componente aleatoria. Se aprecia una clara tendencia interanual, afectada por la crisis previa a 2018, que en esta empresa presentó un comportamiento anormalmente alto y se frenó y volvió a sus valores interanuales normales coincidiendo con el estallido de crisis.

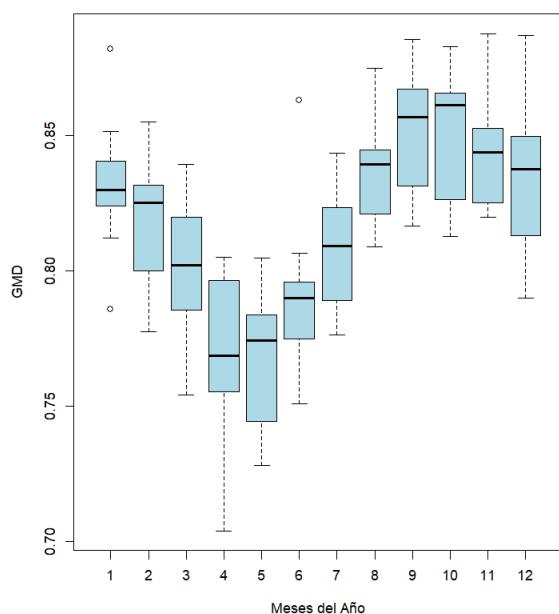


Ilustración 8. Boxplot para comportamiento de ctos por meses. Elaboración propia

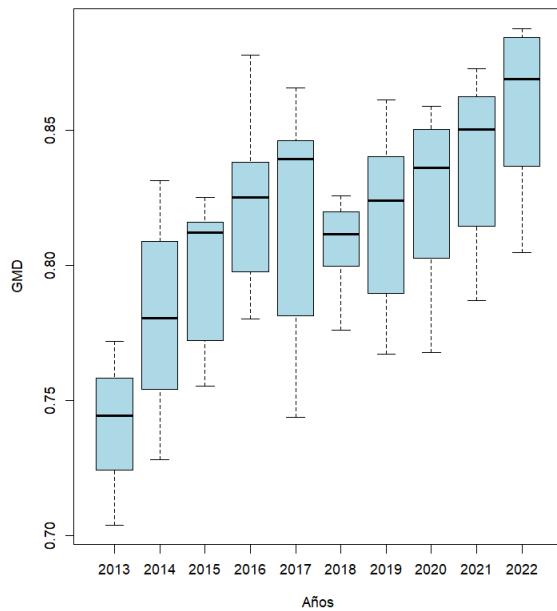


Ilustración 9. Boxplot para ctos agrupado por años. Elaboracion propia.

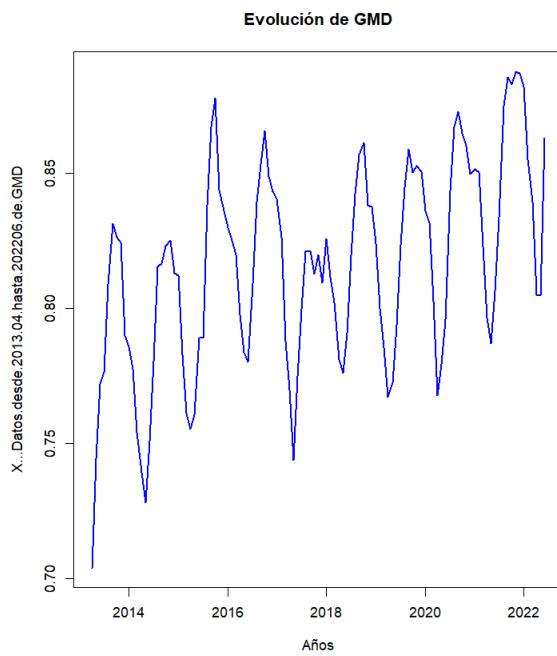


Ilustración 10. Evolución de GMD por meses para los últimos años. Elaboración propia.

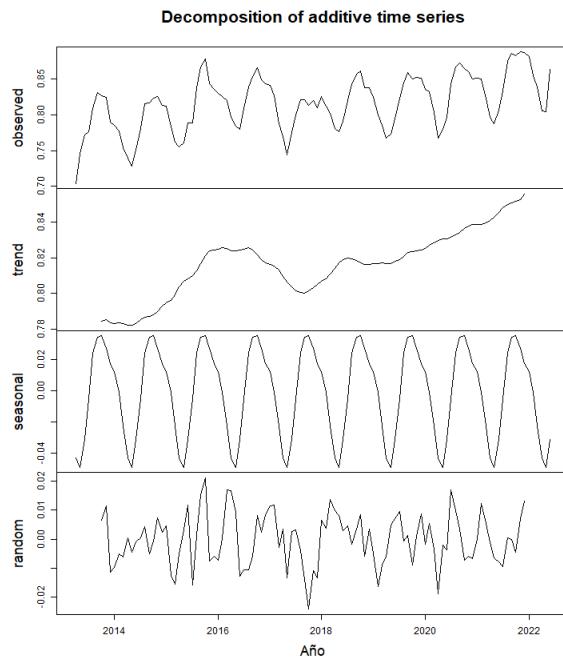


Ilustración 11. Descomposición de la serie temporal. Elaboración propia.

Otra columna calculada que podría ayudar es la que indica el porcentaje de hembras del contrato. En el dataset hay un campo de sexo, que indica un código que identifica el sexo del contrato, esto es si todos los animales son hembras o machos, si hay mezcla y en qué porcentaje, si hay machos castrados (que se tratan a todos los efectos como hembras), etc. El problema es que estos códigos no revelan directamente esta distribución de forma ordenada para poder comparar y usarla adecuadamente en los algoritmos de regresión, por ello parece conveniente añadir una columna que recoja esta distribución por sexos indicando el porcentaje de animales que son hembras. Esto haría innecesaria la columna de sexo, pues la nueva nos resultará mucho más descriptiva y útil.

En la versión inicial del dataset no se ha incorporado el pienso consumido, pero en la segunda versión del dataset para este proyecto se va a incorporar. Se dispondrá de un valor global para todo el contrato y la duración del mismo, esto es difícilmente interpretable entre la gran variedad de contratos con distinto tamaño (de animales) y duración. Por ello añadir al dataset la columna calculada que indique la cantidad de pienso, pero expresada por animal y día puede resultar mucho más ajustado a lo que pretendemos, de esta forma con independencia de las características del contrato será un buen indicador comparativo. La variable del pienso por animal y día puede resultar muy conveniente para el resultado buscado, pues si hay algún problema en la marcha del contrato suele ser un buen indicador que los animales bajen el consumo y que esto nos indique de forma temprana que tenderemos índices más bajos de transformación.

## Calidad de los Datos

### Datos Faltantes

De cara a analizar la calidad de los datos, lo primero que nos fijaremos será en los datos faltantes en el dataset. Se detectan datos faltantes en las columnas: NumBajas, GPS\_Longitud, GPS\_Latitud, gr\_direccion y gr\_codpos, como se puede apreciar en la Tabla 1.

Tabla 1. Columnas con datos faltantes. Elaboración propia

Columna	Faltan	Porcentaje
NumBajas	4	0,075%
GPS_Longitud	5	0,094%
GPS_Latitud	5	0,094%
gr_direccion	979	18,385%
gr_codpos	3	0,056%

Todas las correcciones se incluyen en un script que se adjunta como anexo al proyecto ()�.

Para el número de bajas, si está en blanco parece que se debe a que el contrato no ha tenido ninguna baja y por ello la consulta que recopilaba los datos del dataset lo ha dejado en blanco, procede por tanto sustituir en estos casos el valor por cero.

Para las columnas GPS\_Latitud y GPS\_Longitud (ambas forman parte de la misma característica localización GPS, separadas para poder apreciar más fácilmente diferencias entre localizaciones por diferente latitud o longitud), en este caso veo que se refieren a contratos antiguos de granjas que probablemente han cambiado de integrador y las tenemos dadas de alta con otro código del que podremos recoger las coordenadas. Si vemos los nombres de las naves (na\_nombre) de las granjas que no tenemos los datos de localización GPS, obtenemos: "PASO DEL PINO (OLD)" y "SALGADO (BAJA)". Buscamos la localización de esta granja en otras filas del dataset, que se refieran a la misma granja para otro integrador y podemos llenar estos valores por los valores reales. Las granjas que buscar son: "PASO DEL PINO" y "SALGADO". Se consiguen corregir los valores de estos (ver script anexo para más detalles).

Para el caso del código postal las filas con problemas tienen en el campo población incluido el código postal por lo que podemos extraerlo de allí, y de esa forma corregirlo.

Ya sólo falta el campo gr\_direccion. Este campo tiene muchas filas con valor nulo (en concreto 979 filas). Como es un campo que no tiene mucha influencia para el cálculo no merece la pena intentar corregirlo, ni eliminar las filas que tengan nulos en este campo. La decisión más acertada para este problema podría ser eliminar esta columna, o rellenarla con los datos en el campo provincia. En este proyecto se usará la opción de eliminar la columna.

### Atributos que precisan limpieza

El dataset de partida procede del ERP de la empresa Cefu S.A. y ya está bastante limpio en los atributos traídos. Aparte de los datos faltantes ya tratados en el apartado anterior, habría que comprobar si hay “outliers” ((Boukerche et al., 2021) y (Altman & Krzywinski, 2016)) que merece la pena eliminar.

La primera columna que comprobar será la de GMD, ya que es la que deseamos estimar, y añadir errores en la misma puede desvirtuar la regresión que pretendemos realizar. Como se apreciaba en el histograma (Ilustración 12) la distribución se parece mucho a la suma de dos distribuciones normales, la mayoritaria y centrada en 0.85 aproximadamente (del cerdo blanco) y la de ibérico centrada en 0.7 aproximadamente. En los extremos se observan unos pocos valores muy atípicos que deberíamos quitar. Si mostramos los valores menores a 0.5 o mayores a 1.1 obtenemos un total de 25 filas (19 menores que 0.5 y 6 mayores que 1.1, véase ) con valores atípicos que parece razonable eliminar, siendo además un número muy pequeño con respecto al total de 5332 filas iniciales. Antes de eliminar estos valores se mostraron sus valores y se valoró con los expertos de negocio en estos datos (departamento veterinario). Finalmente se entendió que eran errores de grabación y lo más procedente era eliminarlos del dataset, no resultando un problema por su pequeña proporción en el total de datos.

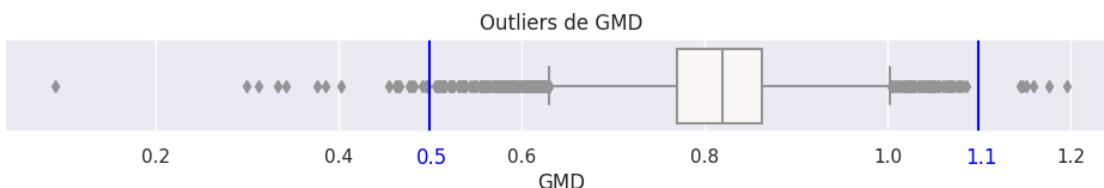


Ilustración 12. Boxplot con Outliers de GMD. Elaboración propia.

Para la columna de DiasMedios, se eliminarán a la vista de los datos los contratos con menos de 50 días o más 210 días. Esto nos arrojará 30 filas de “outliers” para esta variable, tratadas de forma similar a lo comentado en el caso de GMD.

Tras eliminar los outliers de ambas columnas aún disponemos de 5.277 filas en nuestro dataset. Se puede consultar con más detalle este estudio y su implementación en el anexo .

## Preparación de los Datos

### Descripción del Dataset

En esta fase partiremos del dataset ya mencionado en el apartado anterior, y aplicaremos las medidas que detectamos en esos mismos apartados, con el fin de obtener un dataset preparado para la fase de modelado y entrenamiento de los algoritmos. Por tanto, en esta fase terminaremos de seleccionar, limpiar, añadir las columnas calculadas que puedan aportar información al problema, etc.

Partimos por tanto de un dataset con las características generales que se muestran en la Ilustración 13:

Dataset statistics		Variable types	
Number of variables	27	Numeric	17
Number of observations	5332	Categorical	9
Missing cells	996	Boolean	1
Missing cells (%)	0.7%		
Duplicate rows	0		
Duplicate rows (%)	0.0%		

Ilustración 13. Estadísticas generales de dataset inicial. Generado con ydata-profiling.

Como se aprecia el dataset inicial contiene 5332 filas, correspondientes ese mismo número de contratos distintos, para un total de 27 variables, y 996 valores perdidos (que corresponden a 0.7%), no hay filas duplicadas.

Cada una de las variables de este dataset ya han sido comentadas en el apartado anterior, por lo que no se repetirán en este, y únicamente se definirán las nuevas columnas que se creen en este apartado.

### Selección de Datos

Partimos de todas las variables del dataset inicial ya mencionado y definido. Del mismo procede analizar aquellas variables que se evidencia, o al menos lo parece, que no aportan nada a la regresión que pretendemos estimar, que como ya mencionamos era sobre el valor esperado de la Ganancia Media Diaria de los contratos de porcino que puedan venir en un futuro. Son las siguientes:

- ct\_codigo: Es un código único identificativo del contrato. El código es de tipo entero y se autoincrementa para asignar un valor diferenciado a cada contrato, si bien a nivel e Base de Datos es muy utilizado como nexo de las diferentes tablas referentes a los movimientos y detalles de los contratos. En realidad su valor no expresa nada relacionado con ningún parámetro concreto del mismo y es numérico como podría haber sido alfanumérico pues su valor es único y no proporcional a las características de este. Por lo ya comentado no se usará en los datos con los que entrenar las regresiones, aunque no se eliminará del

dataset por si en algún punto se detectaran “outliers”, registros muy relevantes o necesitaríamos ampliar con nuevas variables no consideradas hasta ahora. Este campo nos permitiría traer nuevos datos de los contratos e incorporarlos correctamente al dataset, pues la mayoría de los datos se detallan a nivel de contrato en los datos de la empresa.

- ct\_granja: Número de granja de entre las del integrador. Se trata de un atributo que no aporta nada al modelo a construir. Este número ordena las granjas partiendo desde 1 para el integrador. Es un valor muy poco distribuido, con una gran cantidad de registros para los 3 primeros valores (81,49) y que si bien es útil en la BBDD para la integridad referencial no parece aportar nada a la regresión buscada. Ya hay otros campos que identifican de forma única a las granjas, este necesitaría hacerlo en conjunción con otros campos. Por todo lo explicado, será una columna que eliminar del dataset.
- ct\_nave: Número de nave principal del contrato. De nuevo este dato es muy dependiente del integrador y granja y no parece relevante, por lo que procede eliminarlo del dataset para el actual trabajo.
- ct\_ali\_liquida: Esta columna que para los datos tratados presenta un único valor, siendo para todos los contratos igual a 1, por lo que no aporta nada. Es un campo que tiene sentido en granjas de madres, pero no en granjas de engorde, que son las que estamos tratando, y por ello procede eliminarlo.
- ct\_tipo\_ali: La correlación de este campo con el tipo de ganado (ct\_tipo) es de 0.996, por lo que no parece acertado conservar ambas, siendo preferible desde mi punto de vista eliminar este campo y conservar el tipo de ganado que comunica mejor su significado.
- gr\_direccion: Esta columna tiene 979 datos faltantes y para recoger datos de su localización ya disponemos de otros atributos más representativos, fáciles de interpretar, representar y que si resultan completos y fiables. Eliminamos este valor.

Una vez analizadas las columnas que tras esta fase ya vemos procedente eliminar del dataset podemos pasar a hacerlo. De nuevo el código completo que contiene esta eliminación de las citadas columnas y los pasos de análisis previos que nos han llevado a la conclusión de que se debían eliminar incluso antes de comenzar con el modelado y evaluación de los métodos de regresión, se encuentra detallado en el anexo . Aquí únicamente reproduciré el comando usado para realizarlo, en Ilustración 14.

### Quitar Atributos innecesarios

```
# Quitamos las columnas que no necesitamos
df.drop(columns=["ct_granja","ct_nave","ct_ali_liquida","ct_tipo_ali","gr_direccion"], inplace=True)
```

Ilustración 14 - Quitar atributos innecesarios del Dataset Inicial. Elaboración propia.

Hasta este punto del proyecto las variables que mantenemos son las mostradas en la Tabla 2.

Tabla 2. Columnas del Dataset tras quitar atributos no necesarios. Elaboración propia.

Índice	Columna	Tipo
0	ct_codigo	int64
1	ct_integra	int64
2	ct_tipo	int64
3	ct_raza	int64
4	ct_fase	int64
5	ct_sexo	int64
6	IncPeso	float64
7	DiasMedios	float64
8	GMD	float64
9	EntradaInicial	object
10	EntradaFinal	object
11	NumAnimales	int64
12	na_nombre	object
13	na_rega	object
14	se_nombre	object
15	PesoEntMedio	float64
16	PesoRecMedio	float64
17	NumBajas	float64
18	GPS_Longitud	float64
19	GPS_Latitud	float64
20	gr_codpos	float64
21	gr_poblacion	object
22	KgPiensoTotal	int64

## Limpieza de Datos

De las columnas que aún nos quedan se deben realizar las medidas de limpieza que descubrimos del análisis del paso anterior, aquí las retomaremos e implementaremos para dejar los datos mejor preparados para poder usarse en los siguientes pasos del proceso, en los que crearemos el modelo y comenzaremos a probar algoritmos de regresión que nos puedan acercar a los resultados buscados.

El primer atributo que limpiar, y de vital importancia para el objetivo de este trabajo, es el de GMD, porque constituye el campo objetivo que deseamos poder estimar en función del resto de los atributos disponibles. En la Ilustración 15 se muestra un gráfico de caja y bigotes y un histograma de la distribución de los valores, marcando el punto que realmente se considera “outlier” para esta variable. Los rangos considerados “outliers” se han evaluado revisando el detalle de los contratos con los valores más extremos y consultándolos con los expertos veterinarios de la empresa. Puesto que los valores que consideramos “outliers” no son demasiados, ni parecen representar un tipo de comportamiento específico sino más bien errores en la captura de los datos, procede como ya se comentó en el apartado anterior eliminar completamente esos

registros y no influenciar de otra manera de forma desproporcionada al normal funcionamiento de este campo.

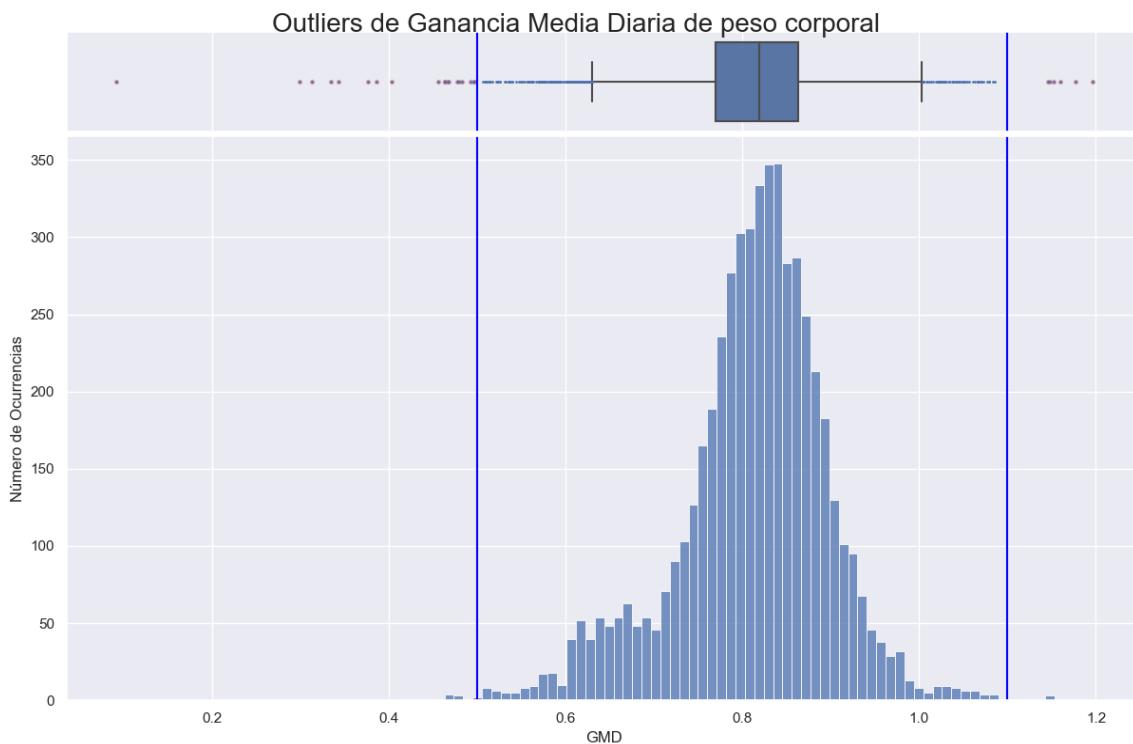


Ilustración 15. Distribución de valores de GMD. Elaboración propia.

Las instrucciones usadas para eliminar los valores que consideramos fuera de rango y que se deben a errores de introducción de datos son las mostradas en la Ilustración 16.

```
# Eliminamos los outliers de GMD (25 filas)
df.drop(df[df.GMD <= 0.5].index, inplace=True)
df.drop(df[df.GMD >= 1.1].index, inplace=True)
```

Ilustración 16. Eliminar outliers de GMD. Elaboración propia.

De la misma forma para la columna del dataset de Días Medios se comprobó que había unos pocos valores que quedaban fuera de rango y procedía analizar si eran “outliers”. Del análisis de estos datos extremos se vio procedente eliminar aquellos valores que estaban por debajo de 50 días o por encima de 210 días, eran en total unos 30 registros únicamente (Ilustración 17) y se decidió al igual que en la variable anterior eliminarlos pues no parecían responder a un comportamiento de algunos contratos, parecían un fallo en la captura de los datos.

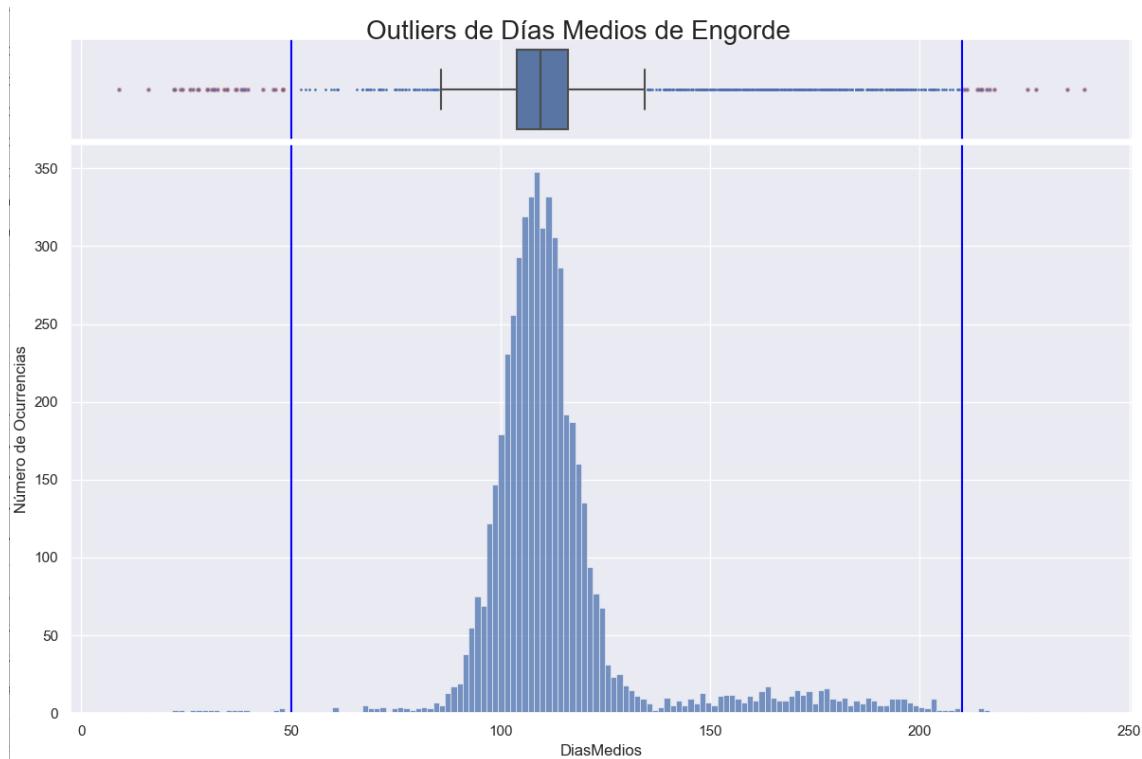


Ilustración 17. Outliers de DiasMedios en Dataframe inicial. Elaboración propia.

El código usado para eliminar estos datos es el que se muestra en la imagen de la Ilustración 18, que es un extracto del anexo .

```
# Eliminamos los outliers de DiasMedios (30 filas)
df.drop(df[df.DiasMedios <= 50].index, inplace=True)
df.drop(df[df.DiasMedios >= 210].index, inplace=True)
```

Ilustración 18. Eliminar outliers de DiasMedios. Elaboración propia.

Tras la eliminación de los valores fuera de rango de las dos columnas citadas, todavía disponemos en el dataset de 5277 filas, habiendo eliminado únicamente 55 registros, que es en torno al 1,03%.

Tras el análisis de los “outliers” detectados, como ya se indicó en el apartado anterior, y como se puede consultar con más detalle en el anexo , se corrigieron los errores en las columnas con valores faltantes.

La primera fue la de número de bajas, que únicamente afectaba a 4 filas y que tras analizarlas se corresponden a contratos que no han tenido bajas y por tanto procede rellenarlas con 0 bajas.

Para el caso de las coordenadas de las granjas que no disponemos de valor, podemos ver por el nombre de la granja o el REGA de la misma, a qué granjas se refieren realmente, en realidad son granjas que tenemos en el Dataset, pero que por cambio de propietario se dieron de baja y abrieron con un nuevo código, lo que provocó que

las cargáramos sin completar sus valores. Podemos recuperar sus valores correctos de las otras filas, que es precisamente lo que se ha hecho en el anexo de tratamiento de los datos, y cuyo extracto de esta parte del código se muestra en la Ilustración 19.

```
granja_01 = 'PASO DEL PINO (OLD)'  
latitud_01 = df[df.na_nombre=='PASO DEL PINO']['GPS_Latitud'].unique()  
longitud_01 = df[df.na_nombre=='PASO DEL PINO']['GPS_Longitud'].unique()  
print('Coordenadas Paso del Pino: Latitud=', latitud_01, ', Longitud=', longitud_01)  
df.loc[df.na_nombre==granja_01, 'GPS_Latitud'] = latitud_01  
df.loc[df.na_nombre==granja_01, 'GPS_Longitud'] = longitud_01  
  
granja_02 = 'SALGADO (BAJA)'  
latitud_02 = df[df.na_nombre=='SALGADO']['GPS_Latitud'].unique()  
longitud_02 = df[df.na_nombre=='SALGADO']['GPS_Longitud'].unique()  
print('Coordenadas Salgado: Latitud=', latitud_02, ', Longitud=', longitud_02)  
df.loc[df.na_nombre==granja_02, 'GPS_Latitud'] = latitud_02[0]  
df.loc[df.na_nombre==granja_02, 'GPS_Longitud'] = longitud_02[0]  
  
print('Número de filas con nulos en GPS_Latitud', df.GPS_Latitud[df.GPS_Longitud.isnull()].sum())
```

Ilustración 19. Rellenar valores perdidos de localización GPS de granjas. Elaboración propia.

Finalmente, para el caso de los códigos postales tenemos una situación análoga a la anterior y se puede resolver de la misma forma, con la certeza de completar con los valores correctos y sin riesgos de estar desvirtuando los registros y su veracidad. En ese caso en el propio registro en el campo de población aparece el valor que necesitamos y podemos recuperar, más concretamente aparece en el campo de la dirección de la granja (véase anexo ), el texto se muestra en la Ilustración 20.

Como se puede ver en el campo gr\_poblacion tienen metido el código postal, por lo que podemos sacarlo de aquí. En las 3 filas es 04600.

```
df['gr_codpos'] = df['gr_codpos'].fillna('04600')  
# Comprobamos que ya hemos quitado todos los nulos de la columna de CP  
df[df.gr_codpos.isnull()]
```

Ilustración 20. Rellenar valores perdidos de código postal. Elaboración propia.

El campo del nombre de la granja presenta problemas de limpieza, de forma que la misma granja en distintos registros recibe nombres ligeramente diferentes. Esto se debe a la forma en la que se almacena en la base de datos la granja, de forma que una misma granja puede tener varias instancias si cambia de integrador o condiciones de facturación, pudiendo diferir o no el nombre de estas instancias. Por ello un proceso de limpieza puede hacer que esta variable, que pueda ser categórica de cara a la regresión, sea más precisa.

Lo primero que se puede hacer con la misma es quitarle los stopwords que contengan y puedan aparecer en unas instancias y no en otras referidas a la misma granja. Estos stopwords serían los determinantes, y otros sufijos que se añaden al nombre para indicar que están de baja, que son la granja antigua o el detalle del integrador entre paréntesis. También ayuda el poner todas las palabras entre paréntesis y quitar las

tildes en los nombres. Veo de crear una función que haga precisamente todo lo indicado:

```
import re

def simplificar_nombre(cadena):
    # Pasamos a mayúsculas
    cadena = " " + cadena.upper() + " "
    sustituir = {"Á":"A", "É":"E", "Í":"I", "Ó":"O", "Ú":"U", "-":", ":","~":"", "\.":"", " OLD ":"", " BAJA ":" ", " LA ":" ", " EL ":" ",
    " LOS ":" ", " LAS ":" ", " IBERICO ":" ", " IB ":" ", " LECHONERA ":" ", " LECHONERAS ":" ", "\xa0":""}
    # Quitamos lo que esté entre paréntesis
    cadena = re.sub('\(.*)', ' ', cadena)
    # Sustituimos los caracteres acentuados y las stopwords
    for cad1, cad2 in sustituir.items():
        cadena = re.sub(cad1, cad2, cadena)
    while ' ' in cadena:
        cadena = re.sub(' ', ' ', cadena)
    return cadena.strip()
```

Ilustración 21. Función para simplificar nombres de granjas. Elaboración propia.

Y la llamo para crear una nueva columna con los nombres corregidos.

```
df['na_nombre2'] = df.na_nombre.map(simplificar_nombre)
```

Ilustración 22. Corregir nombre de granjas. Elaboración propia.

Como se puede ver a continuación se han conseguido eliminar 106 nombres de granjas que en realidad se referían a la misma granja.

```
print('na_nombre', str(len(df.na_nombre.unique())), ' filas')
print('na_nombre2', str(len(df.na_nombre2.unique())), ' filas')

na_nombre 597  filas
na_nombre2 491  filas
```

Ilustración 23. Mostrar reducción de ocurrencias duplicadas de granjas. Elaboración propia.

Ya podemos eliminar la columna original (que se mantuvo únicamente para poder comparar los valores que hemos conseguido consolidado como la misma granja).

```
# Borramos la columna del nombre original de la granja, porque ya disponemos de la simplificada
df.drop(['na_nombre'], inplace=True, axis=1)
```

Ilustración 24. Eliminar columna de nombre de granja no corregida. Elaboración propia.

## Formateo de Datos

Los campos de la fecha de entrada y salida del contrato están almacenados sin el formato correcto, por lo que lo primero puede ser corregir este para poder sacar más datos de los valores de estos atributos, datos que nos permitan analizar las

componentes temporales y estacionales que pueda tener influencia en la regresión a realizar. Se pasarán a formato “Datetime” (véase anexo ).

El campo Sexo en el dataset se representa por 2 campos un código ct\_sexo y su descripción se\_nombre. De este último se puede inferir que los animales pueden ser de los siguientes tipos: machos, hembras o machos castrados. Los castrados a nivel de crecimiento se comportan como las hembras, por lo que los podemos asemejar a estas. Los sexos de los contratos se componen de un porcentaje de animales de estos 3 tipos básicos de sexos. Por lo tanto, sería más adecuado para el cálculo usar un porcentaje total de animales hembras/castrados, frente a machos. Podemos hacer un mapeo (Ilustración 23) de estos valores en tanto por uno, con lo que añadiríamos la nueva columna del porcentaje de hembras (o castrados). Esta columna hará innecesarias el resto de las columnas de sexo que se podrán eliminar del dataset que usaremos para la regresión.

```
# Añadimos el mapeo de los valores
sexos = { 1:1, 2:0.5, 3:1, 4:0, 5:1, 8:0.66 }
# Añadimos la columna de Porcentaje de Hembras
df['PorcHembras'] = df.ct_sexo.map(sexos)
# Comprobamos que la hemos creado correctamente
df.value_counts(['ct_sexo','se_nombre','PorcHembras'])
```

ct_sexo	se_nombre	PorcHembras	
2	MACHO ENTERO + HEMBRA	0.50	2495
4	MACHO ENTERO	0.00	1156
5	HEMBRA	1.00	1104
8	MACHO ENTERO + CASTRADO+ HEMBRA	0.66	485
3	MACHO CASTRADO	1.00	20
1	MACHO CASTRADO + HEMBRA	1.00	17

dtype: int64

Ilustración 25. Calcular porcentaje de cada sexo en contratos. Elaboración propia.

## Atributos Calculados

Como parece razonable que en el cálculo pueda tener influencia la estacionalidad, dato apuntado por los expertos de la empresa (grupo de veterinarios), esta se puede mostrar más fácilmente si disponemos del número de semana de la entrada al contrato. Además, aprovecharemos para añadir el año por si este tiene influencia y las mejoras en las granjas y procesos se notan en la evolución del GMD. Estos campos se pueden obtener fácilmente del campo de fecha que formateamos como tal en el apartado anterior.

```
# Añadimos la Semana de Entrada y el Año de Entrada
df['semanaEntrada'] = df.EntradaInicial.dt.isocalendar().week
df['añoEntrada'] = df.EntradaInicial.dt.isocalendar().year
```

Ilustración 26. Añadir campos de semana y año de entrada en contrato. Elaboración propia.

Por otra parte, para el sexo como ya mostramos en el apartado anterior se añadió el campo de porcentaje de cada sexo que hay en el contrato, por lo que no se repetirá aquí.

En cuanto al pienso consumido, se dispone de este en cantidad total para el contrato, pero no es lo más conveniente para poder comparar entre contratos de muy distinto tamaño, duración, etc. Por ello sería más beneficioso para nuestra aplicación poder resumirlos de forma ponderada y comparable entre contratos, haciendo que el dato sea en función de animales y días del contrato, de esta forma lo transformaremos en un valor de kilos consumidos por cerda y día, que ya sí es comparable entre los distintos contratos. El pienso puede ser un gran indicador de la normal marcha del contrato, pues cuando hay problemas se empieza a notar muy rápidamente en el descenso del consumo. En la Ilustración 27 se aprecia la fórmula utilizada para ajustar la columna de Pienso por Cerdas y Día.

$$PiensoCerdaDia = \frac{KgPiensoTotal}{NumAnimales * DiasMedios}$$

Ilustración 27. Calcular consumo de pienso por cerda y día. Elaboración propia.

### Resumen de preparación de datos

Tras todo el proceso de limpieza anterior nos encontramos con un dataset que dispone de 5.277 filas y 28 columnas. Partíamos de un dataset con 5.332 filas (hemos quitado 55 filas por distintos motivos) y que tenía 27 columnas, se han añadido varias columnas calculadas nuevas y se han eliminado otras que ya no eran necesarias, corrigiendo los tipos de los datos por ejemplo de las columnas de tipo fecha.

El nuevo dataset está mucho más limpio, sin perder excesivas muestras en el proceso, ya no tiene “missing values”, y posee datos mejor preparados para poder usarlos en siguientes procesos de “Machine Learning”.

Se podría eliminar también la columna del código del contrato, que es un identificador que ya no aporta nada de cara a la estimación que queremos lograr en posteriores pasos, pero de momento se mantiene por si en futuras versiones algún dato se aprecia fuera de rango poder saber qué contrato lo provoca y poder consultar más detalle de este, o de cara poder facilitar si detectamos que necesitamos más datos o encontramos algún nuevo atributo relevante a incorporar, pues facilitaría poder hacerlo de forma única, pues en la base de datos los datos están almacenados haciendo referencia al campo de contrato.

### Modelado

El propósito principal del proyecto es el de obtener una buena aproximación del valor esperado de la Ganancia Media Diaria para los distintos contratos que se puedan abrir en el marco de la empresa Cefu S.A., aprovechando igualmente para poder comprender mejor qué parámetros son los que más afectan y en qué medida, al óptimo funcionamiento del engorde de los animales, para centrarse en intentar

controlar y mejorar en aquello que realmente se traducirá en resultados palpables. El objetivo estratégico principal desde el punto de vista de la empresa es poder adelantarse al comportamiento de los contratos en curso y poder estimar las disponibilidades que tendrá a corto y medio plazo.

La obtención de la estimación buscada nos dará un mejor control y planificación, permitirá adelantarse a problemas y necesidades futuras, optimizar procesos alineados con lo que los datos nos demuestran que realmente es importante, reduciendo la incertidumbre y mejorando el proceso día a día. Pues será un proceso vivo, que iterativamente podrá crecer con el negocio, adaptarse a los nuevos datos, descubrir nuevos parámetros relevantes, incorporarlos al procedimiento, evaluar su eficacia, y en definitiva permitir que la empresa mejore día a día, con un conocimiento basado en decisiones que se apoyen en los datos y nos alejen del peligro de apostar el futuro de la empresa a corazonadas, decisiones arbitrarias o poco fundamentadas.

En esta fase se partirá de los datos que se han venido recopilando, limpiando, generando y seleccionando en los apartados anteriores, se adaptarán a la forma que necesita cada algoritmo utilizado, y se irán definiendo la selección de los atributos que realmente se usarán en el algoritmo final. Se separarán los datos en datos de entrenamiento y prueba, se normalizarán las variables numéricas y tratarán las categóricas para adecuarse a la forma en la que la pueden tratar los distintos algoritmos probados, se separará la variable objetivo y se irán creando subconjuntos de variables a pasar a los distintos métodos para ver con cuáles se obtienen mejores resultados y por qué, de forma que quizás descubramos la necesidad de incorporar nuevos datos que podamos tener disponibles o sea interesante empezar a capturar.

### **Selección de Técnica de Modelado**

El proceso que se seguirá en esta fase del trabajo será el de probar distintos algoritmos de regresión, y evaluarlos de acuerdo con la precisión del modelo, y en qué porcentaje explica la variable objetivo. En el proceso se intentará decidir el algoritmo más apropiado, sus mejores hiperparámetros y sobre qué conjunto de variables se aplicará.

El primer modelo que elegiremos probar será el de RandomForest, ya que es un modelo sencillo, que permite identificar las variables más relevantes y en qué orden, siendo un buen punto de partida para comenzar a decidir las mejores variables se deben incluir inicialmente y hacernos una idea sobre qué resultados esperar con los distintos modelos para los datos que disponemos.

### **Generar modelos de prueba**

Para la generación de los modelos dividiremos los datos disponibles en datos de entrenamiento y de prueba, con lo que tendremos un set de datos para evaluar la precisión del modelo obtenido, que no han sido usados para el entrenamiento. Con esta estrategia evitamos que el algoritmo sobre aprenda y no sea capaz de generalizar, con lo que nos proporcionará mejores resultados sobre los datos que

realmente necesitamos estimar, que serán los de contratos futuros que aún desconocemos.

### Generar modelos

Una vez obtenido el modelo que mejores resultados proporcione, lo guardaremos y prepararemos para poder invocarlo con datos de nuevos contratos para que nos proporcione el valor estimado.

### Evaluación de modelo

La evaluación de modelo se medirá en el porcentaje de explicación de la variable objetivo con el modelo y variables seleccionadas, evaluadas como ya hemos comentado sobre los datos de pruebas, que estará constituido por un subconjunto disjunto, del utilizado para entrenamiento, de los datos del dataset obtenido y preparado en los anteriores pasos del proyecto.

### Revisión de parámetros

Para cada modelo se probará con varios hiperparámetros, con el objetivo de optimizar la precisión del modelo y ajustarnos a aquellos que permitan un mejor entrenamiento y modelo más ajustado a los resultados que esperamos de adecuación a la regresión de la variable GMD.

#### *Evaluación del Modelo*

Para cada modelo planteado se analizarán los resultados obtenidos con el mismo, haciendo esto primero sobre la puntuación del entrenamiento y luego sobre los datos de test, que al no haber sido usados para el entrenamiento permiten comprobar si el modelo generaliza correctamente o tiene un “overfitting” (sobreajuste) a los datos de entrenamiento.

La evaluación de cada modelo que probamos traerá consigo unos resultados que compongan un ranking de los mejores modelos. Finalmente habrá un modelo seleccionado que será el que mejor puntuación tenga de acuerdo a las métricas acordadas en la definición de los objetivos. El mejor modelo será el que se evalúe más a fondo y el que finalmente se desarrollará para poder usarlo con datos de contratos futuros.

Para evaluar los modelos se ha implementado una función que muestre las diferencias entre el valor real y el predicho por el modelo, junto con su coeficiente de Determinación  $R^2$  (Ilustración 28), aunque habrá que ser cauto con el coeficiente  $R^2$  porque si la regresión no es lineal podría no ser un estimador acertado (Spiess & Neumeyer, 2010).

```
# Función para Graficar diferencias entre valor predicho y real en datos de test del modelo pasado
def graficoDiferencias(modelo, X_test_s, y_test):
    y_pred = modelo.predict(X_test_s)
    diferencia = abs(y_pred - y_test)
    g = sns.jointplot(x=y_test, y=y_pred)
    # Draw a line of x=y
    x0, x1 = g.ax_joint.get_xlim()
    y0, y1 = g.ax_joint.get_ylim()
    lims = [max(x0, y0), min(x1, y1)]
    g.ax_joint.plot(lims, lims, '-r')
    g.ax_joint.scatter(x=y_test, y=y_pred, c=diferencia.values, cmap=sns.dark_palette("#69d", reverse=True, as_cmap=True))
    plt.show()
```

Ilustración 28. Función para Graficar diferencias del modelo y la realidad. Elaboración propia.

## 1.1.2. Preparación de Datos para regresión

### *Corregir tipo y dimensionalidad de Atributos*

Como paso inicial a implementar y probar ningún modelo se partirá de los datos que se habían preparado en los apartados anteriores del proyecto y se estudiará su adecuación a un modelo de regresión, así como si estos datos se pueden obtener para contratos en curso (que son los que realmente queremos estimar con el modelo final).

Tras la carga del Dataframe anterior disponemos de las siguientes columnas: 'ct\_codigo', 'ct\_integra', 'ct\_tipo', 'ct\_raza', 'ct\_fase', 'IncPeso', 'DiasMedios', 'GMD', 'EntradaInicial', 'EntradaFinal', 'NumAnimales', 'na\_rega', 'PesoEntMedio', 'PesoRecMedio', 'NumBajas', 'GPS\_Longitud', 'GPS\_Latitud', 'gr\_codpos', 'gr\_poblacion', 'KgPiensoTotal', 'semanaEntrada', 'añoEntrada', 'PorcHembras', 'PiensoCerdaDia', 'na\_nombre2'.

De las citadas columnas serían categóricas las columnas: ct\_integra, ct\_tipo, ct\_raza, ct\_fase, na\_rega, gr\_codpos, gr\_poblacion, na\_nombre2.

En un primer momento de las columnas categóricas hay que estudiar cuales se utilizarán y que número de ocurrencias distintas tienen, para evitar el problema de la “Maldición de la Cardinalidad” (Donoho, 2000). De este análisis las variables referentes al detalle de la granja (ct\_integra, na\_rega, gr\_codpos, gr\_poblacion, na\_nombre2) poseen demasiadas instancias y no serán usadas inicialmente.

La variable de raza si parece relevante de acuerdo a estudios anteriores y a la opinión de los expertos de la empresa, no obstante, posee 22 valores distintos y no bien balanceados. A la vista de esto se decide consultar la opinión de los veterinarios por si se pueden agrupar algunas de estas razas por ser muy similares en crecimiento y genética. Las agrupaciones propuestas se analizan en los datos a nivel del rango de valores de las filas para el GMD y finalmente se reducen a 8 razas principales. Se sustituyen los valores por el representativo de cada agrupación, se pasa a variable categórica y se codifica en las variables de entrenamiento y test según “One Hot Encodig” (véase la implementación del mismo en la Ilustración 29), lo que hará que el modelo codifique las razas sin añadir un sesgo de ordenación que únicamente se

debería a una mala codificación. Este paso se muestra en la Ilustración 30 y añadiría 8 nuevas columnas y se quitaría la original.

```
# Funcion para convertir en One Hot Encoding
def encode_and_bind(original_dataframe, feature_to_encode):
    dummies = pd.get_dummies(original_dataframe[[feature_to_encode]])
    res = pd.concat([original_dataframe, dummies], axis=1)
    res = res.drop([feature_to_encode], axis=1)
    return(res)
```

Ilustración 29. Comvertir en One Hot Encoding. Elaboración propia.

```
# Revisar la raza si se agrupan las razas con menos ocurrencias
agrupar_razas = {93 : 93, 85 : 93, 90 : 93, 95 : 93, 94 : 93, 82 : 93, 80 : 80, 96 : 80, 88 : 88, 0 : 0, 23 : 0,
                 84 : 0, 66 : 0, 18 : 0, 68 : 88, 7 : 7, 89 : 7, 65 : 7, 15 : 15, 97 : 7, 69 : 69, 81 : 81}
df.replace({'ct_raza' : agrupar_razas}, inplace=True)
```

Ilustración 30. Agrupar razas en 8 categorías. Elaboración propia.

### Selección de Variables

Se seleccionarán las variables a utilizar para el entrenamiento y test de los modelos (Ilustración 31). Quedarían tras esto quedaría para entrenamiento 20 columnas (detalladas en la Tabla 3 y correspondientes a 13 atributos, pero la de raza se ha codificado como 8 columnas).

```
# Cargamos las variables objetivo y las usadas (15 variables seleccionadas, una de ellas categórica con 8 valores).
y = df['GMD']
x0 = df[['ct_integra', 'ct_tipo', 'ct_raza', 'IncPeso', 'NumAnimales', 'na_rega',
          'PesoEntMedio', 'PesoRecMedio', 'bajas', 'GPS_Longitud', 'GPS_Latitud',
          'semanaEntrada', 'añoEntrada', 'PorcHembras', 'PiensoCerdaDia']]
features_to_encode = ['ct_raza'] # , 'na_rega']
x1 = x0.copy()
x1.drop(['ct_integra', 'na_rega'], inplace=True, axis=1)
for feature in features_to_encode:
    x1 = encode_and_bind(x1, feature)
```

Ilustración 31. Selección de variables para regresión. Elaboración propia.

*Tabla 3. Columnas Seleccionadas para regresión. Elaboración propia.*

Columna	Tipo
ct_tipo	int64
IncPeso	float64
NumAnimales	int64
PesoEntMedio	float64
PesoRecMedio	float64
bajas	float64
GPS_Longitud	float64
GPS_Latitud	float64
semanaEntrada	int64
añoEntrada	int64
PorcHembras	float64
PiensoCerdaDia	float64
ct_raza_0	uint8
ct_raza_7	uint8
ct_raza_15	uint8
ct_raza_69	uint8
ct_raza_80	uint8
ct_raza_81	uint8
ct_raza_88	uint8
ct_raza_93	uint8

#### *Correlación entre variables*

Con estas variables comprobaremos si hay variables que no sean independientes entre sí, para ello generaremos la matriz de dispersión (“scatter matrix”, Ilustración 32) y la de covarianzas (Ilustración 33) para cada par de variables y luego en detalle para cada variable con la variable objetivo GMD (Ilustración 34).

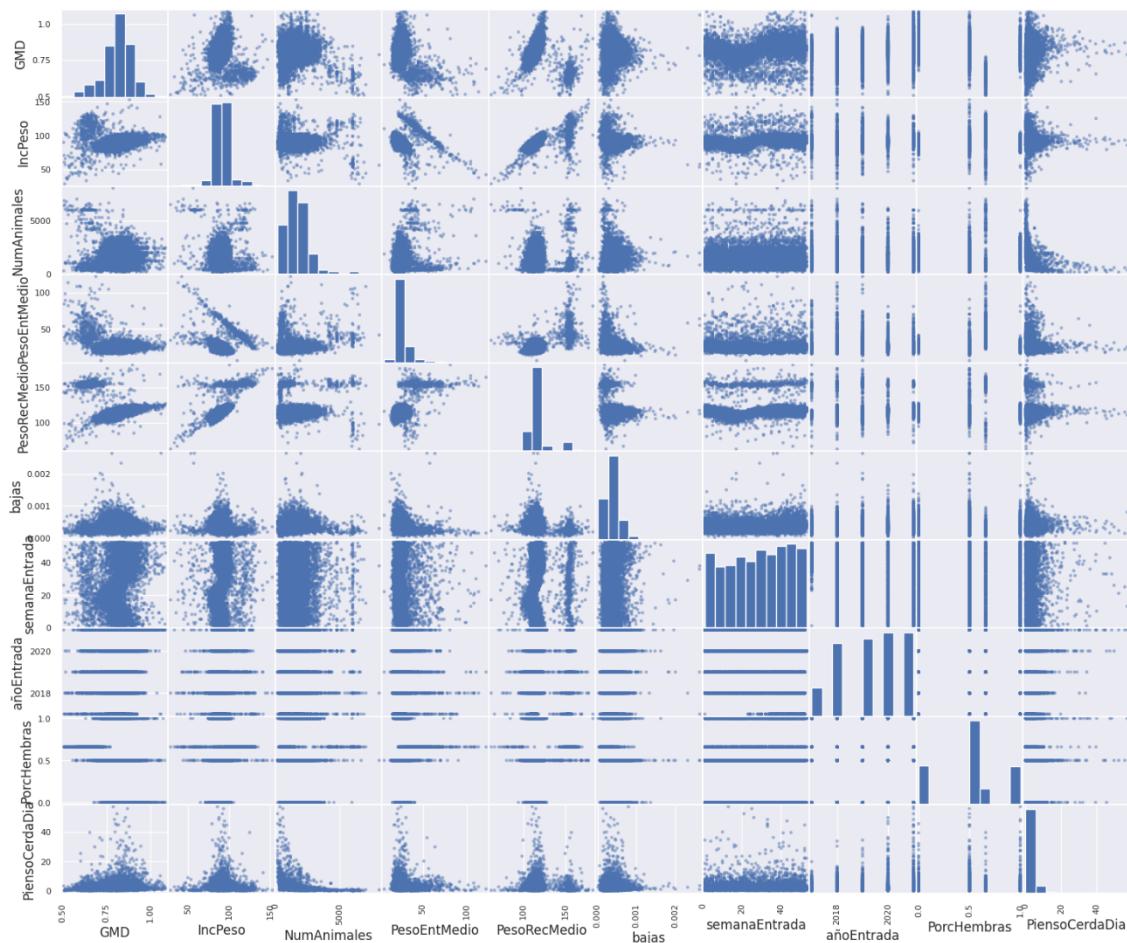


Ilustración 32. Matriz de dispersión de las variables seleccionadas para regresión. Elaboración propia.



Ilustración 33. Matriz de correlaciones. Elaboración propia.

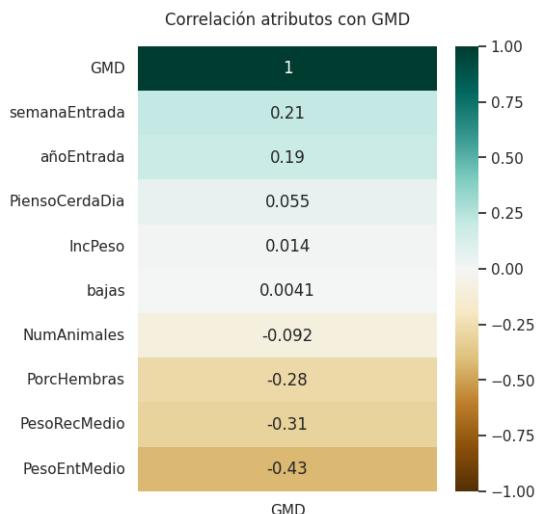


Ilustración 34. Correlación de atributos con GMD. Elaboración propia.

A la vista de las 3 ilustraciones anteriores (Ilustración 32, Ilustración 33 y Ilustración 34) no se aprecia ninguna correlación fuerte entre las variables que aconseje estudiar si quitarla antes de pasarla al modelo.

#### *Dividir datos en Entrenamiento y Prueba*

Finalmente, los datos seleccionados se dividirán en 2 categorías, la de entrenamiento (“train”) y la de prueba (“test”), de forma que podamos evaluar el modelo sin temor a un resultado afectado por el sobreajuste y la no generalización para casos no observados. Se ha elegido un 20% de los registros como registros de prueba y un 80% para entrenamiento, reordenando las filas (“shuffle”) para evitar que el grupo de prueba sea de un conjunto de registros similares por cercanía en el dataset original.

#### *Escalado de los datos*

Para evitar que de forma artificial unas variables tengan más influencia que otras debido a su diferente escala, se ha optado por escalar los datos haciendo uso de “RobustScaler” (Yin, 2015) y (Pires et al., 2020)), que es un método de “scikit-learn”, que intenta evitar la influencia de los “outliers” y para ello escala los valores en función del rango intercuartílico.

### 1.1.3. RandomForest

El primer algoritmo que se decidirá probar será el de RandomForest (Breiman, 2001), todo el proceso que se detalla a continuación se puede seguir en código para el apéndice Regresión RandomForest, disponible para su consulta y ejecución en la siguiente web de Github: [Regresión RandomForest.ipynb](#)

Si se genera el modelo con los parámetros por defecto y lanza el entrenamiento del mismo para los datos de entrenamiento preparados en el apartado anterior, obtenemos los datos que se muestran en la Ilustración 35. Como se puede apreciar tiene un coeficiente de determinación del 83.32%, lo que nos indica que el modelo explica la variación de la variable objetivo (GMD) en más de un 83%. Pero también nos interesa saber el resto de los errores y por ello mostramos estos indicadores en la Tabla 4.

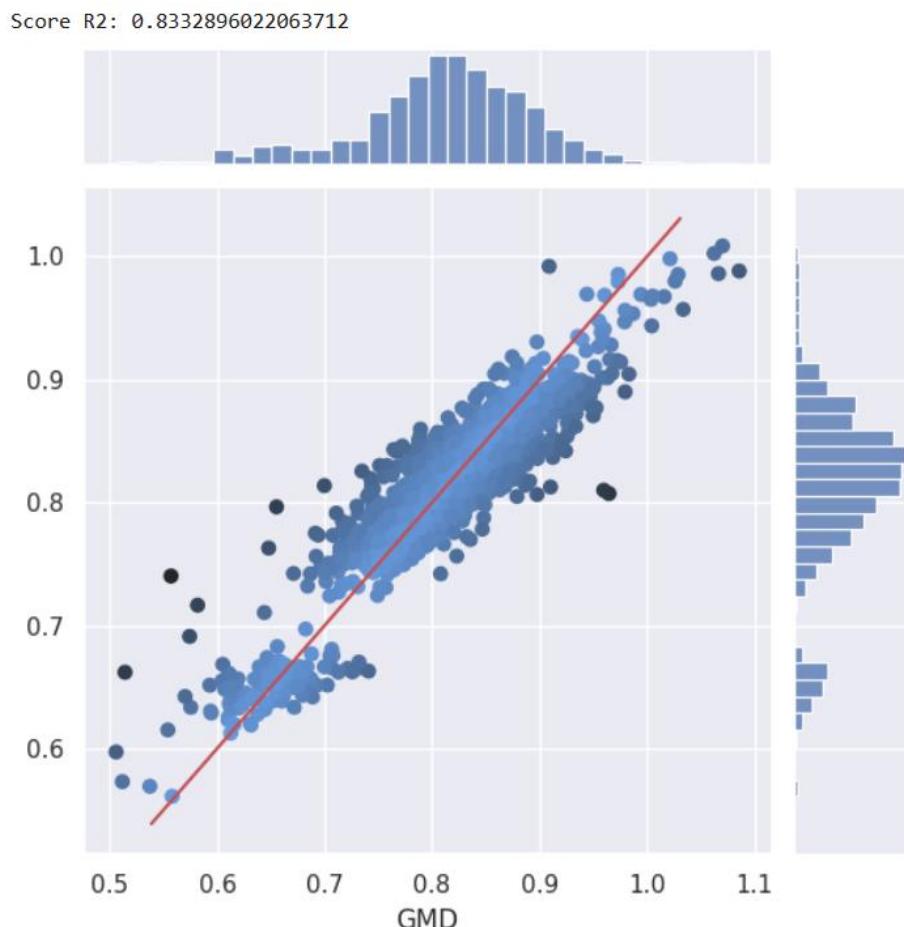


Ilustración 35. Diferencias RandomForest v1. Elaboración propia.

Tabla 4. Errores de estimación para RandomForest v1. Elaboración propia.

Métrica	Valor	Descripción
R <sup>2</sup>	0,8333	Coeficiente de Determinación
RMSE	0,0012	Raíz de error cuadrático medio
MAE	0,0265	Error absoluto medio
MAX	0,1832	Error Máximo

Las 10 características más relevantes para la regresión según el modelo del Random Forest son las que se muestran en la Tabla 5.

Tabla 5. Características más relevantes según RandomForest. Elaboración propia.

Orden	Característica	Importancia
1	PesoRecMedio	22,82%
2	ct_raza_69	11,54%
3	ct_tipo	11,44%
4	PesoEntMedio	8,69%
5	IncPeso	7,27%
6	PorcHembras	6,33%
7	semanaEntrada	5,68%
8	NumAnimales	4,12%
9	ct_raza_93	4,08%
10	bajas	4,06%

Con estos resultados el siguiente paso podría ser el de ver hasta qué punto se puede optimizar los resultados arrojados por el modelo si intentamos buscar una combinación de hiperparámetros que se ajuste mejor a nuestro problema. Para ello haré uso de una búsqueda aleatoria entre un rango de valores posibles proporcionados al optimizador.

Los valores de los hiperparámetros a intentar optimizar y sus posibles valores son los que muestro en la Ilustración 36, y su ejecución en el comando que lanza su búsqueda se muestra en la Ilustración 37, que tardó en ejecutarse para el entorno de “Google Colab” un tiempo de 31 minutos, en el que entrenó y evaluó aleatoriamente entre 300 combinaciones del rango de hiperparámetros propuesto la mejor solución para optimizar el error cuadrático medio, probando unas 300 combinaciones para 3 particiones de los datos cada una, de modo que estimara con estas el error mediante el uso de “Cross Validation” (Yang, 2007).

```
# Hago una optimización de los hiperparámetros para RandomForest
from sklearn.model_selection import RandomizedSearchCV
random_grid = {'bootstrap': [True, False],
               'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, None],
               'max_features': [1.0, 'sqrt', 2, 5, 10, 20],
               'min_samples_leaf': [1, 2, 4, 10],
               'min_samples_split': [2, 5, 10, 20],
               'n_estimators': [20, 50, 75, 100, 150, 250, 500, 750]}
```

Ilustración 36. Hiperparámetros a optimizar en RandomForest. Elaboración propia.

```
rf_random = RandomizedSearchCV(scoring="neg_mean_squared_error", estimator = rf,
                                 param_distributions = random_grid, n_iter = 300,
                                 cv = 3, verbose=1, random_state=123, n_jobs = -1)
rf_random.fit(X_train_s, y_train)

Fitting 3 folds for each of 300 candidates, totalling 900 fits
▶ RandomizedSearchCV
▶ estimator: RandomForestRegressor
    ▶ RandomForestRegressor
```

Ilustración 37. Búsqueda de hiperparámetros RandomForest. Elaboración propia.

Los parámetros finalmente seleccionados son los mostrados en la Tabla 6 y el error del nuevo modelo es el que se muestra en la Tabla 7, también se muestra como en el anterior las diferencias gráficamente (Ilustración 38).

Tabla 6. Mejores parámetros para RandomForest. Elaboración propia.

Parámetro	Valor
n_estimators	750
min_samples_split	2
min_samples_leaf	1
max_features	5
max_depth	60
bootstrap	False

Tabla 7. Errores de mejores parámetros para RandomForest. Elaboración propia.

Métrica	Valor	Descripción
R^2	0,8412	Coeficiente de Determinación
RMSE	0,0011	Raíz de error cuadrático medio
MAE	0,0259	Error absoluto medio
MAX	0,1791	Error Máximo

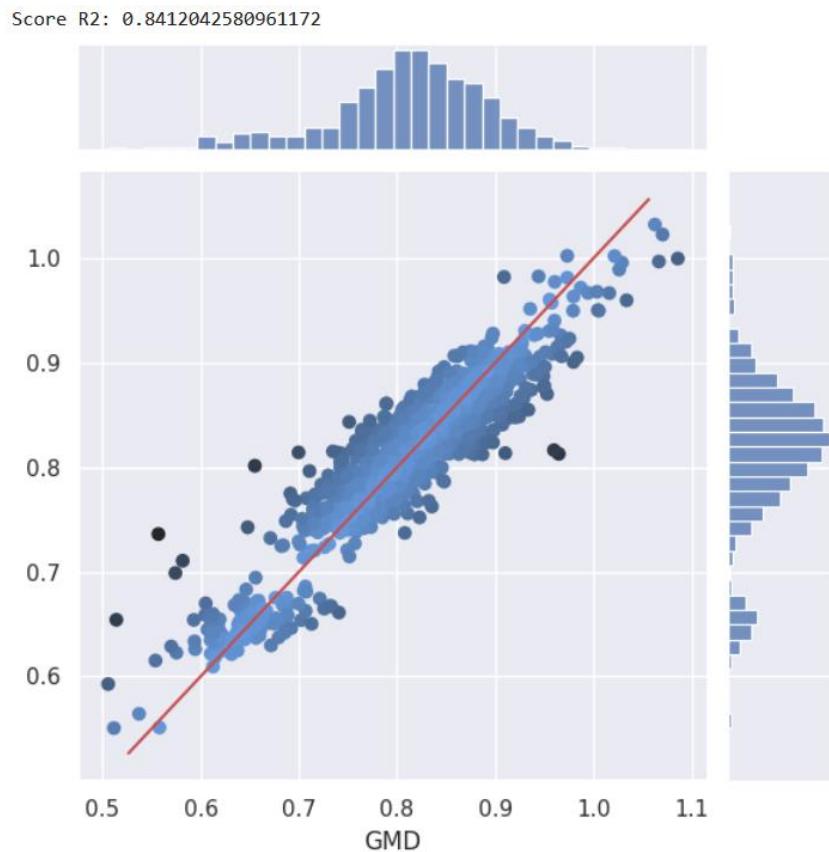


Ilustración 38. Errores RandomForest con mejores parámetros. Elaboración propia.

#### 1.1.4. LazyPredict

Tras la prueba de varias configuraciones del modelo de RandomForest, y obtener la clasificación de la importancia de las variables según el modelo de RandomForest entrenado, lo siguiente es decidir con qué otros modelos probar. Para facilitar este proceso y para poder hacernos una idea de como se comportarán una gran variedad de modelos en poco tiempo haremos uso de la herramienta de “Python” “LazyPredict”. Esta herramienta permite en su versión actual probar una gran cantidad de modelos de forma muy rápida y sencilla, y ofrece un ranking de los mismos en función de varios estimadores.

Para el caso que nos ocupa, que es de regresión, la versión actual (v0.2.12 al momento de escribir este proyecto) ofrece hasta 41 modelos a probar con tan sólo pasarle el conjunto de datos de entrenamiento y la variable objetivo. Luego el resultado que ofrece consta del ranking de modelos con los estimadores, en el caso de la regresión Adjusted R-Squared, R-Squared y RMSE.

En la prueba ejecutada desde el mismo cuaderno de “Jupyter Notebook” Regresión RandomForest, con el mismo conjunto de datos de entrenamiento y test que para el modelo de RandomForest se obtuvo la clasificación de los modelos que se muestran en la tabla Tabla 8.

Tabla 8. Ranking de modelos segun LazyPredict. Elaboración propia.

Modelo	R2 Ajustado	RMSE
HistGradientBoostingRegressor	0,85	0,03
LGBMRegressor	0,85	0,03
ExtraTreesRegressor	0,85	0,03
XGBRegressor	0,84	0,03
RandomForestRegressor	0,83	0,03
NuSVR	0,82	0,04
GradientBoostingRegressor	0,82	0,04
BaggingRegressor	0,81	0,04
KNeighborsRegressor	0,78	0,04
AdaBoostRegressor	0,73	0,04
SVR	0,71	0,05
LassoLarsIC	0,67	0,05
LassoLarsCV	0,67	0,05
LarsCV	0,67	0,05
LassoCV	0,67	0,05
ElasticNetCV	0,67	0,05
SGDRegressor	0,67	0,05
RidgeCV	0,67	0,05
Ridge	0,67	0,05
Lars	0,67	0,05
BayesianRidge	0,67	0,05
LinearRegression	0,67	0,05
TransformedTargetRegressor	0,67	0,05
MLPRegressor	0,66	0,05
OrthogonalMatchingPursuitCV	0,62	0,05
HuberRegressor	0,61	0,05
ExtraTreeRegressor	0,60	0,05
LinearSVR	0,57	0,06
DecisionTreeRegressor	0,57	0,06
GammaRegressor	0,56	0,06
TweedieRegressor	0,56	0,06
OrthogonalMatchingPursuit	0,53	0,06
DummyRegressor	-	0,08
LassoLars	-	0,08
ElasticNet	-	0,08
Lasso	-	0,08
GaussianProcessRegressor	- 5,75	0,22
KernelRidge	- 91,42	0,81

Como se puede apreciar en la Tabla 8, hay 8 modelos por encima del 80% para el coeficiente de autodeterminación ajustado, pero con pocas diferencias entre ellos. Con los parámetros por defecto hay 3 modelos que están en torno al 85%, y con los que convendría probar si hay margen de mejora haciendo ajuste de los hiperparámetros o modificando las variables usadas para entrenar el modelo.

Los primeros modelos serán los siguientes modelos a probar en el proyecto a ver si tienen aún más margen de optimización con el tuneo de sus hiperparámetros.

### 1.1.5. HistGradientBoostingRegressor

El primer modelo a probar será el que obtuvo la mejor puntuación según la herramienta LazyPredict. Es el modelo HistGradientBoostingRegressor. Este modelo se basa en árboles de decisión con aumento de gradiente según histogramas.

Lo primero que hago será generar el modelo con los parámetros por defecto y ver si efectivamente se comporta bien con nuestros datos (como nos decía la herramienta LazyPredict). Obtenemos como se muestra en la figura Ilustración 39 una puntuación según R2 de 0,8477.

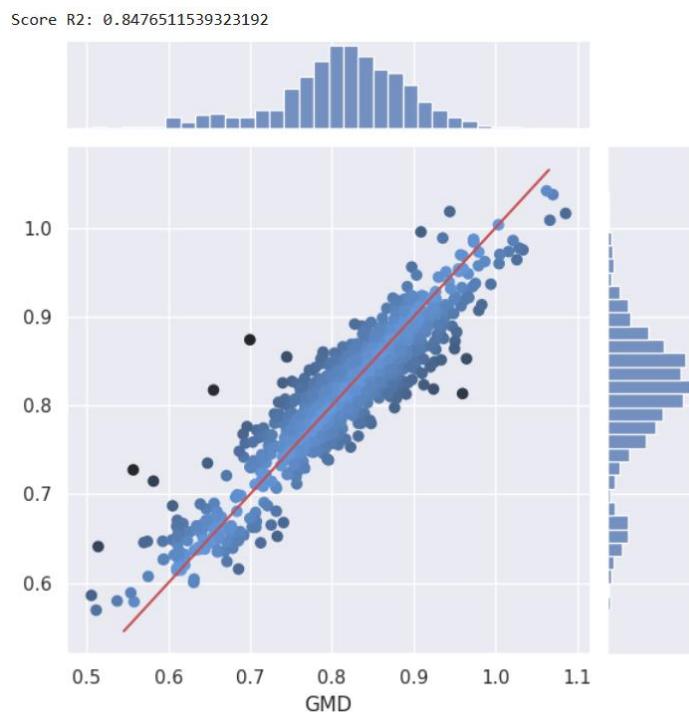


Ilustración 39. Resultado de HistGradientBoostingRegressor en dataset del proyecto. Elaboración propia.

Intentamos ver si podemos mejorar los resultados con un ajuste de los parámetros del método y para ello lanzamos una búsqueda aleatoria entre un gran número de

combinaciones de parámetros posibles. En la Ilustración 40 se muestra los parámetros que se posibilitan al modelo para probar los resultados con los mismos y en la Tabla 9 el mejor resultado obtenido. Como se puede observar el método tiende a sobreajustarse y mejorar los resultados en el conjunto de entrenamiento, pero no a hacerlo en el de test, por lo que esta optimización no ha representado ningún avance.

```
# Hacemos una optimización de los hiperparámetros básicos para RandomForest y vemos de probar de forma
# aleatoria con la combinación de 100 de estos modelos a ver cuál es el que mejor ajusta el MSRE.
from sklearn.model_selection import RandomizedSearchCV
random_grid = {'learning_rate': [0.0001, 0.001, 0.01, 0.1, 0.2, 0.25, 0.3, 0.5],
               'max_iter': [10, 30, 50, 100, 200, 500, 1000],
               'max_leaf_nodes': [5, 10, 20, 30, 50, 100, 200]}
modelo_random = RandomizedSearchCV(scoring="neg_mean_squared_error", estimator = modelo,
                                     param_distributions = random_grid, n_iter = 100,
                                     cv = 3, verbose=1, random_state=123, n_jobs = -1)
modelo_random.fit(X_train_s, y_train)

Fitting 3 folds for each of 100 candidates, totalling 300 fits
>   RandomizedSearchCV
>     estimator: HistGradientBoostingRegressor
    > HistGradientBoostingRegressor
```

Ilustración 40. Rangos de parámetros para la búsqueda aleatoria de la mejor combinación. Elaboración propia.

Tabla 9. Errores de HistGradientBoostingRegressor tras optimizar parámetros. Elaboración propia.

Estimadores para Train		
Metr.	Valor	Descripción
R^2	0,9181	Coeficiente de Determinación
RMSE	0,0006	Raíz de error cuadrático medio
MAE	0,0184	Error absoluto medio
MAX	0,1369	Error Máximo
Estimadores para Test		
Metr.	Valor	Descripción
R^2	0,8438	Coeficiente de Determinación
RMSE	0,0011	Raíz de error cuadrático medio
MAE	0,0253	Error absoluto medio
MAX	0,1744	Error Máximo

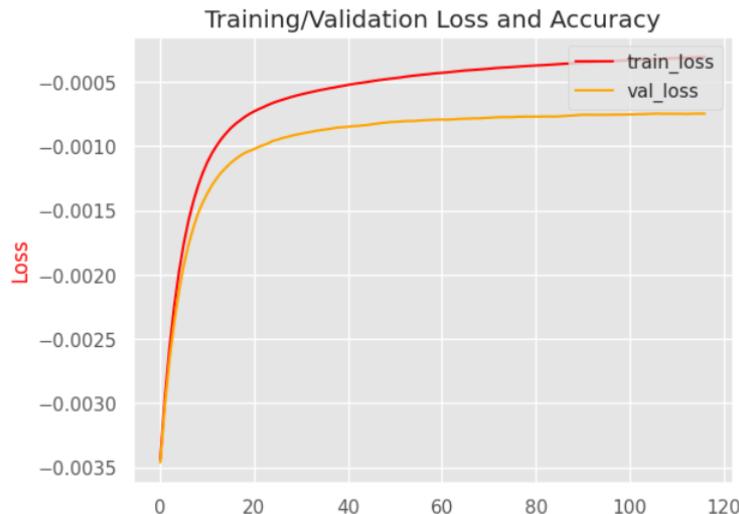
Intentamos ahora hacerlo con early\_stopping. Este modelo permite ser lanzado con la opción de early stopping, y esta busca optimizar no los resultados para train si no para test (reserva un porcentaje de los datos de entrenamiento para realizar la validación con datos no usados). Podemos ponerle un número de iteraciones máximas muy grande pues se espera que acabe antes el proceso por no mejorar según las condiciones de early stopping. Este modelo nos ofrece además para los modelos la puntuación obtenida del estimador para entrenamiento y test en cada una de las iteraciones del método. Lanzamos de nuevo el modelo con estos nuevos parámetros y vemos que deja de mejorar en validación para los parámetros seleccionados tras 95 iteraciones. Veo de volverlo a lanzar con mayor libertad de parámetros a ver si encuentra alguno que mejore los resultados en validación. Como se puede apreciar en la Ilustración 40 de este modo tampoco mejoramos en validación y el método

rápidamente tiende a sobreajustarse y comienzan a alejarse las gráficas de pérdida para entrenamiento y validación.

*Tabla 10. HistGradientBoostingRegressor mediante earlystopping tras búsqueda de parámetros.  
Elaboración propia.*

Estimadores para Train		
Metr.	Valor	Descripción
R^2	0.8920809102796025	(Coeficiente de Determinación)
RMSE	0.0007425233930035075	(Raíz de error cuadrático medio)
MAE	0.020471776688240143	(Error absoluto medio)
MAX	0.22465169947440788	(Error Máximo)

Estimadores para Test		
Metr.	Valor	Descripción
R^2	0.8378583629080998	(Coeficiente de Determinación)
RMSE	0.00116185006680291	(Raíz de error cuadrático medio)
MAE	0.025875958889140085	(Error absoluto medio)
MAX	0.18115629775007847	(Error Máximo)



Dejamos aquí este modelo y vemos de probar con el siguiente de la lista según LazyPredict.

### 1.1.6. KNeighborsRegressor

Para variar un poco y dado que todos los modelos que aparecen en la cima del Ranking de la herramienta LazyPredict son de tipo bosque, se probará ahora con el modelo de los K-vecinos más cercanos. Es de esperar que este modelo obtenga peores valores, pero merece la pena probarlo y al ser sencillo nos puede valer para probar a incluir menos variables a ver si beneficia o perjudica a los resultados obtenidos. Como en los modelos anteriores las pruebas y ajustes del modelo y los resultados obtenidos están disponibles en un cuaderno de Jupyter Notebook, alojado en Github, en el enlace [KNeighborsRegressor.ipynb](#).

Como en el resto de los modelos lo primero fue cargar los datos que ya teníamos preparados de los procesos anteriores del proyecto y probar con los valores por defecto del modelo para ver el error que obteníamos, que se muestra en la Ilustración

41, que como se puede apreciar tiene un coeficiente de autodeterminación de apenas 74,59%.

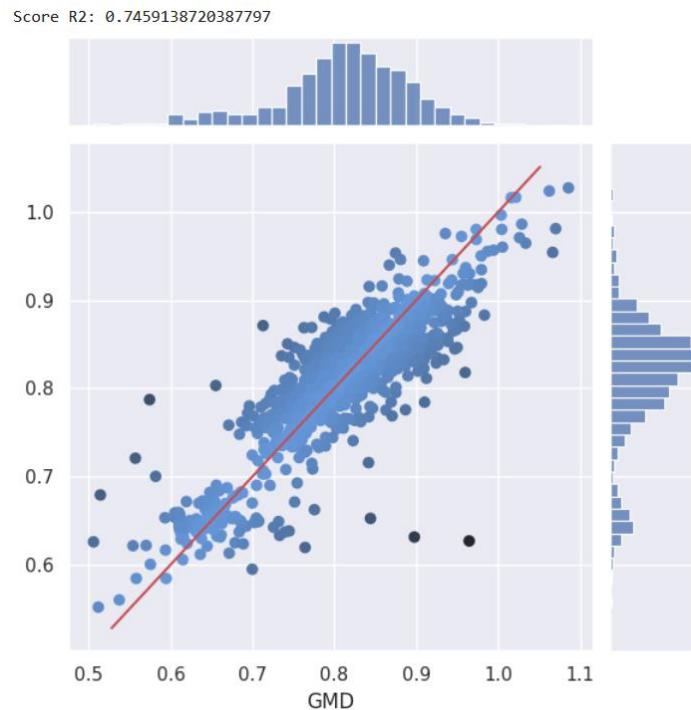


Ilustración 41. Errores de KNeighborsRegressor. Elaboración propia.

Lo siguiente fue comprobar el valor de número de vecinos más cercanos óptimo para el dataset del problema. Para ello se itera sobre los 100 primeros números como valor de número de vecinos más cercanos a contemplar. Y en la se muestra el resultado y cómo el valor más alto se encuentra entre 9 y 16, con valor absoluto más alto para 16.

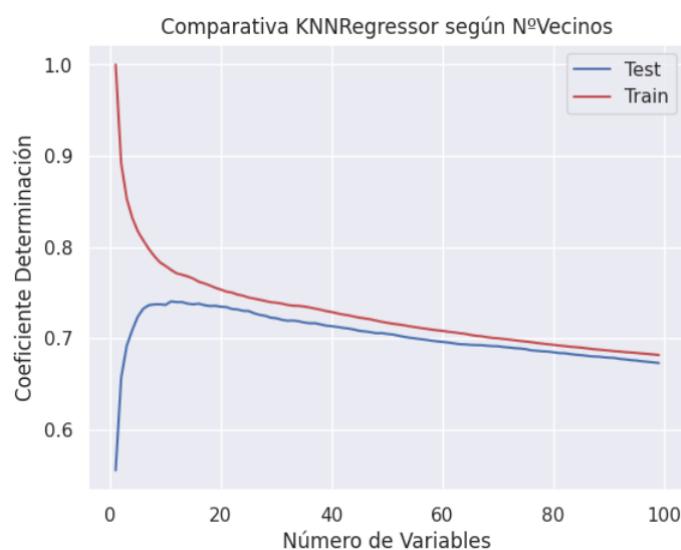


Ilustración 42. Comparativa de valor de R2 para KNN según nº vecinos. Elaboración propia.

Al principio para los datos de entrenamiento parece arrojar valores muy buenos, pero en realidad comprobamos en test que no generalizan nada el comportamiento. Los óptimos se mueven en torno a 10-16 vecinos más cercanos, en concreto para k=16 se obtiene el mejor valor de la serie con un R<sup>2</sup>=0.7050.

La siguiente prueba que realizamos es si obtiene mejores resultados con menos variables, y para ello obtenemos el top 10 de variables según la calificación proporcionada por el método de RandomForest y vamos viendo como se comporta para distinto número de variables, desde 1 a 10 en el orden de importancia. Como se aprecia en la figura para este modelo se comporta mejor con más variable en general, salvo por el paso de 3 variables a 4 variables (Ilustración 43).

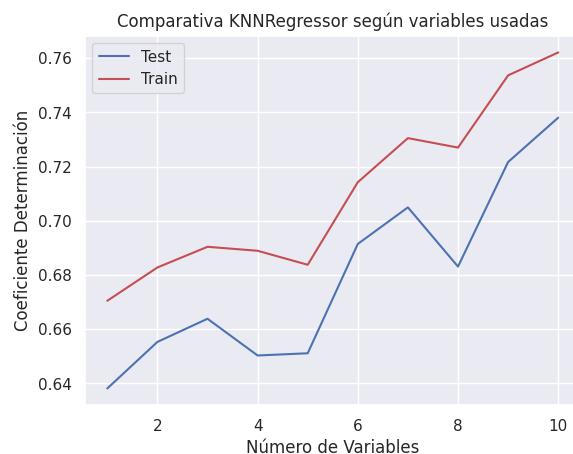


Ilustración 43. Comparativa de R<sup>2</sup> para KNN según número de variables usadas. Elaboración propia.

### 1.1.7. Evaluación modelo final

#### Evaluación de Resultados

Por implementar...

#### Resultados de Minería

Por implementar...

#### Éxito desde punto de vista del negocio

Por implementar...



**Criterio**

Por implementar...

**1.1.8. Implementación del modelo para producción**

**Plan de desarrollo**

Por implementar...

**Plan de monitoreo y mantenimiento**

Por implementar...

**Informe final de producción**

Por implementar...

**Proyecto de revisión**

Por implementar...



## 4.4. Resultados

## 5. Conclusión y trabajos futuros

### 6. *¡¡¡Por implementar!!! (quitar)*

Especifica y resalta los resultados de la investigación o del desarrollo en relación con los objetivos planteados.

Habla sobre las dificultades que te has encontrado y las perspectivas para el futuro de este trabajo.

## 7. Referencias

- Agostini, P., De Blas, C., & Gasa, J. (s. f.). MADRID, 6 y 7 de Noviembre de 2013 XXIX CURSO DE ESPECIALIZACION FEDNA 61 CARACTERIZACIÓN E INFLUENCIA DE LOS PRINCIPALESFACTORES DE PRODUCCIÓN SOBRE LOS RENDIMIENTOS DE CERDOS DE CEBO EN CONDICIONES COMERCIALES ESPAÑOLAS.
- Altman, N., & Krzywinski, M. (2016). THIS MONTH Analyzing outliers: Influential or nuisance? <https://doi.org/10.1038/nmeth.3812>
- Ayele, W. Y. (2020). Adapting CRISP-DM for Idea Mining A Data Mining Process for Generating Ideas Using a Textual Dataset. IJACSA) International Journal of Advanced Computer Science and Applications, 11(6). [www.ijacsa.thesai.org](http://www.ijacsa.thesai.org)
- Boukerche, A., Lining Zheng, & Alfandi, O. (2021). Outlier Detection: Methods, Models, and Classification. ACM Computing Surveys, 53(3), 1-37. <https://doi.org/10.1145/3381028>
- Breiman, L. (2001). Random Forests. 45, 5-32.
- Campos Benvenga, M. A., Nääs, I. de A., Lima, N. D. da S., & Pereira, D. F. (2022). Hybrid Metaheuristic Algorithm for Optimizing Monogastric Growth Curve (Pigs and Broilers). AgriEngineering, 4(4). <https://doi.org/10.3390/agriengineering4040073>
- Donoho, D. L. (2000). High-Dimensional Data Analysis: The Curses and Blessings of Dimensionality.
- Lázaro, S. F., Ibáñez-Escríche, N., Varona, L., Silva, F. F. e., Brito, L. C., Guimarães, S. E. F., & Lopes, P. S. (2017). Bayesian analysis of pig growth curves combining pedigree and genomic information. Livestock Science, 201, 34-40. <https://doi.org/10.1016/J.LIVSCI.2017.03.024>

- Moughan, P. J., & Verstegen, M. W. A. (1988). The modelling of growth in the pig. Netherlands Journal of Agricultural Science, 36(2), Article 2. <https://doi.org/10.18174/njas.v36i2.16687>
- Ncr, & Clinton, J. (1999). CRISP-DM 1.0 Step-by-step data mining guide.
- Pires, I. M., Hussain, F., Garcia, N. M., Lameski, P., & Zdravevski, E. (2020). Homogeneous Data Normalization and Deep Learning: A Case Study in Human Activity Classification. <https://doi.org/10.3390/fi12110194>
- Spiess, A. N., & Neumeyer, N. (2010). An evaluation of R<sup>2</sup>as an inadequate measure for nonlinear models in pharmacological and biochemical research: A Monte Carlo approach. BMC Pharmacology, 10(1), 1-11. <https://doi.org/10.1186/1471-2210-10-6/FIGURES/4>
- Tolosa, A. F., Derouchey, J. M., Tokach, M. D., Goodband, R. D., Woodworth, J. C., Gebhardt, J. T., Ritter, M. J., & Pilcher, C. M. (2021). A Meta-Analysis to Understand the Relationship between Pig Body Weight and Variation from Birth to Market. <https://doi.org/10.3390/ani11072088>
- Wang, L., Hu, Q., Wang, L., Shi, H., Lai, C., & Zhang, S. (s. f.). Predicting the growth performance of growing-finishing pigs based on net energy and digestible lysine intake using multiple regression and artificial neural networks models. <https://doi.org/10.1186/s40104-022-00707-1>
- Wirth, R., & Hipp, J. (2000). CRISP-DM: Towards a Standard Process Model for Data Mining.
- www.3tres3.com. (2010, mayo 18). Manejo en cebo—El crecimiento del animal. www.3tres3.com. [https://www.3tres3.com/latam/articulos/manejo-en-cebo-el-crecimiento-del-animal\\_10904/](https://www.3tres3.com/latam/articulos/manejo-en-cebo-el-crecimiento-del-animal_10904/)



Yang, Y. (2007). CONSISTENCY OF CROSS VALIDATION FOR COMPARING

REGRESSION PROCEDURES 1. *The Annals of Statistics*, 35(6), 2450-2473.

<https://doi.org/10.1214/009053607000000514>

Yin, L. (2015). Published by Science and Education Centre of North America

Investigating Robust Estimation and Forecasting of Volatilities of Futures with

Interquartile Range Models. *Journal of Finance and Economics*, 3(2), 1-10.

<https://doi.org/10.12735/jfe.v3i2p01>

## Glosario

- **Contrato:** Conjunto de animales de similares características, criados en una misma granja y período de tiempo.
- **GMD:** Acrónimo de Ganancia Media Diaria.
- **Ganancia Media Diaria:** Cantidad media de peso en Kilos que se espera que un animal engorde cada día, durante el período normal de engorde.
- **REGA:** Registro general de explotaciones ganaderas. Es un código identificativo de la granja único a nivel estatal.
- **Tipo de Ganado:** Para el ganado porcino hay principalmente dos tipos de ganado el blanco y el ibérico, con crecimientos y pesos objetivos claramente diferenciados.
- **ydata-profiling:** Herramienta para Análisis Exploratorio de Datos (<https://ydata-profiling.ydata.ai/docs/master/>)



# Apéndice I

Contiene documentos de trabajo de elaboración propia para el desarrollo del Trabajo Fin de Máster, y referenciados en el texto principal del proyecto.

## 1.1. GMD profiling Dataset 01

Se puede consultar el fichero del informe generado en la siguiente URL:  
[GMD\\_profiling\\_dataset01.html](#)

Se trata de un fichero HTML generado por la herramienta ydata\_profiling ([YData](#)).

## 1.2. TFM\_Preparar\_Dataset

Enlace al cuaderno de trabajo en “Jupyter Notebook” para poder probar y generar los datos de preparación del dataset.

Se puede consultar el fichero del informe generado en la siguiente URL:  
[TFM\\_Preparar\\_Dataset.ipynb](#)

Adjunto el contenido del cuaderno de Jupyter Notebook citado.

# TFM\_Preparar\_Dataset

April 1, 2023

## 0.1 #Introducción

TFM: Aplicación de ciencia de datos en el sector de producción animal para la predicción y explicación de óptimos en ganado porcino.

*Titulo:* Preparar Dataset Inicial

*Autor:* Jose Eduardo Cámará Gómez

---

## 1 Carga y limpieza de dataset

```
[1]: # Importación de paquetes
import numpy as np
import pandas as pd
import seaborn as sns

import matplotlib.pyplot as plt
import matplotlib.mlab as mlab
import matplotlib
plt.style.use('ggplot')
from matplotlib.pyplot import figure

%matplotlib inline
matplotlib.rcParams['figure.figsize'] = (12,8)

pd.options.mode.chained_assignment = None
sns.set(style="darkgrid")
```

## 2 Cargar Dataset Inicial

```
[2]: from google.colab import files
# Cargamos el fichero del dataset gmd.csv
uploaded = files.upload()

<IPython.core.display.HTML object>
Saving gmd.csv to gmd.csv
```

```
[3]: # Leemos el fichero csv con los datos
df = pd.read_csv('gmd.csv', sep=';')
```

Comprobamos la cantidad de filas y columnas del dataset y los tipos iniciales que infiere en la carga.

```
[4]: print(df.shape)
print(df.dtypes)
```

```
(5332, 27)
ct_codigo      int64
ct_integra     int64
ct_granja      int64
ct_nave        int64
ct_tipo         int64
ct_raza         int64
ct_fase         int64
ct_sexo         int64
ct_ali_liquida object
ct_tipo_ali    int64
IncPeso        float64
DiasMedios     float64
GMD            float64
EntradaInicial object
EntradaFinal   object
NumAnimales    int64
na_nombre       object
na_rega         object
se_nombre       object
PesoEntMedio   float64
PesoRecMedio   float64
NumBajas        float64
GPS_Longitud   float64
GPS_Latitud    float64
gr_direccion   object
gr_codpos      float64
gr_poblacion   object
dtype: object
```

## 2.1 Añadir Columna de Pienso Consumido

```
[5]: # Cargamos el fichero del dataset pienso_por_cto.csv
uploaded = files.upload()
```

```
<IPython.core.display.HTML object>
Saving pienso_por_cto.csv to pienso_por_cto.csv
```

```
[6]: df_pienso = pd.read_csv('pienso_por_cto.csv', sep=';')
```

```
[7]: df_pienso.head()

[7]:   Contra  KgPienso
  0  200191    136400
  1   23803    1180850
  2    3047    264020
  3  200073    303050
  4    2873    225440

[8]: df_pienso.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5357 entries, 0 to 5356
Data columns (total 2 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   Contra      5357 non-null   int64  
 1   KgPienso   5357 non-null   int64  
dtypes: int64(2)
memory usage: 83.8 KB

[9]: df_pienso.columns = ["ct_codigo", "KgPiensoTotal"]

[10]: df_pienso["ct_codigo"] = pd.to_numeric(df["ct_codigo"], downcast='integer')

[11]: # Añadimos al Dataframe el Pienso total
      df = df.merge(df_pienso, on="ct_codigo", how="left")
```

## 2.2 Quitar Atributos innecesarios

```
[12]: # Quitamos las columnas que no necesitamos
      df.
      .drop(columns=["ct_granja", "ct_nave", "ct_ali_liquida", "ct_tipo_ali", "gr_direccion"], u
            .inplace=True)

[13]: df.info()

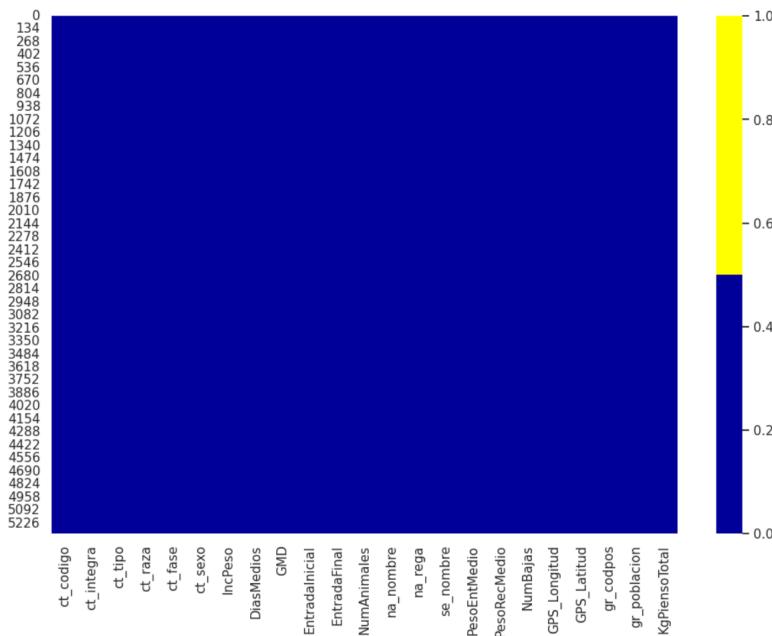
<class 'pandas.core.frame.DataFrame'>
Int64Index: 5332 entries, 0 to 5331
Data columns (total 23 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   ct_codigo   5332 non-null   int64  
 1   ct_integra  5332 non-null   int64  
 2   ct_tipo     5332 non-null   int64  
 3   ct_raza     5332 non-null   int64  
 4   ct_fase     5332 non-null   int64  
 5   ct_sexo     5332 non-null   int64
```

```
6   IncPeso      5332 non-null  float64
7   DiasMedios    5332 non-null  float64
8   GMD          5332 non-null  float64
9   EntradaInicial 5332 non-null  object
10  EntradaFinal   5332 non-null  object
11  NumAnimales    5332 non-null  int64
12  na_nombre      5332 non-null  object
13  na_rega        5332 non-null  object
14  se_nombre       5332 non-null  object
15  PesoEntMedio   5332 non-null  float64
16  PesoRecMedio   5332 non-null  float64
17  NumBajas        5328 non-null  float64
18  GPS_Longitud    5327 non-null  float64
19  GPS_Latitud     5327 non-null  float64
20  gr_codpos      5329 non-null  float64
21  gr_poblacion    5332 non-null  object
22  KgPiensoTotal   5332 non-null  int64
dtypes: float64(9), int64(8), object(6)
memory usage: 999.8+ KB
```

### 2.3 Tratar Missing Values

```
[14]: cols = df.columns[:] # Todas las columnas
colours = ['#000099', '#ffff00'] # specify the colours - yellow is missing. ↴
                                   ↵blue is not missing.
sns.heatmap(df[cols].isnull(), cmap=sns.color_palette(colours))
```

```
[14]: <Axes: >
```



```
[15]: # Mostramos las columnas con algún valor perdido
print('Columnas con valores perdidos:')
print('    Columna          \t Faltan\t Porcentaje')
print('    ----- \t ----- \t -----')
for col in df.columns:
    num_missing = np.sum(df[col].isnull())
    pct_missing = np.mean(df[col].isnull())
    if num_missing>0:
        print('    {} \t {} \t {}%'.format(col.ljust(15), num_missing,round(pct_missing*100)))
```

Columnas con valores perdidos:		
Columna	Faltan	Porcentaje
-----	-----	-----
NumBajas	4	0%
GPS_Longitud	5	0%
GPS_Latitud	5	0%
gr_codpos	3	0%

- 1) Para la columna **NumBajas**, que sólo tiene 4 filas en este estado y con pocos animales, es de suponer que no se ha producido ninguna baja en estos contratos. Por lo que podría ser razonable rellenarlos con 0.

```
[16]: print('Rellenamos con 0 bajas los faltantes')
df['NumBajas'] = df['NumBajas'].fillna(0)
df[df.NumBajas==0]
```

Rellenamos con 0 bajas los faltantes

```
[16]:      ct_codigo  ct_integra  ct_tipo  ct_raza  ct_fase  ct_sexo   IncPeso \
5          20316           1         2       69        20         8  29.724638
2771     205463          398         2       69        20         8 18.250000
4146     200979          504         1         0        20         2 11.000000
4147     201026          504         1         0        20         2  6.000000

      DiasMedios      GMD EntradaInicial ... na_rega \
5    46.289855  0.642142  2021-04-06 ... ES300080840002
2771  61.000000  0.299180  2021-09-15 ... ES300011440001
4146  16.000000  0.687500  2018-09-10 ... ES300261140006
4147  9.000000  0.666667  2018-09-17 ... ES300261140006

      se_nombre PesoEntMedio PesoRecMedio NumBajas \
5    MACHO ENTERO + CATRADO+ HEMBRA    128.26087  157.985507  0.0
2771  MACHO ENTERO + CATRADO+ HEMBRA    140.00000  158.250000  0.0
4146      MACHO ENTERO + HEMBRA     30.00000  41.000000  0.0
4147      MACHO ENTERO + HEMBRA     20.00000  26.000000  0.0

      GPS_Longitud  GPS_Latitud gr_codpos      gr_poblacion KgPiensoTotal
5      -1.39403    37.84721  30840.0  ALHAMA DE MURCIA        286920
2771    -1.01195    38.16640  30640.0    ABANILLA        458300
4146    -1.29353    37.60660  30870.0  MAZARRON (MURCIA)        398680
4147    -1.29353    37.60660  30870.0  MAZARRON (MURCIA)        727460
```

[4 rows x 23 columns]

- 2) Para las columnas **GPS\_Latitud** y **GPS\_Longitud** (ambas forman parte de la misma característica localización GPS, separadas para poder apreciar más facilmente diferencias entre localizaciones por diferente latitud o longitud), en este caso veo que se refieren a contratos antiguos de granjas que probablemente han cambiado de integrador y las tenemos dadas de alta con otro código del que podremos recoger las coordenadas. Vamos a ver los datos a los que se refiere

```
[17]: df[df.GPS_Longitud.isnull()]
```

```
[17]:      ct_codigo  ct_integra  ct_tipo  ct_raza  ct_fase  ct_sexo   IncPeso \
3957      3202        488         1         0        20         2  84.837291
5277     204789        625         1         0        21         4  83.708418
5278     204122        625         1         0        22         5  94.881932
```

```

5279      204788        625      1      0      21      4  84.951442
5280      204121        625      1      0      21      4  97.528509

      DiasMedios      GMD EntradaInicial ... na_rega \
3957  103.733356  0.817840  2017-12-04 ... ES300261240028
5277  108.700956  0.770080  2021-05-03 ... ES300241940002
5278  114.547425  0.828320  2020-12-04 ... ES300241940002
5279  109.050046  0.779013  2021-05-06 ... ES300241940002
5280  109.661638  0.889358  2020-12-04 ... ES300241940002

      se_nombre PesoEntMedio PesoRecMedio NumBajas GPS_Longitud \
3957  MACHO ENTERO + HEMBRA    28.863920   113.701211    116.0      NaN
5277      MACHO ENTERO       24.688869   108.397287     64.0      NaN
5278          HEMBRA       22.535682   117.417614     51.0      NaN
5279      MACHO ENTERO       23.702348   108.653790     76.0      NaN
5280      MACHO ENTERO       22.660912   120.189421     49.0      NaN

      GPS_Latitud gr_codpos      gr_poblacion KgPiensoTotal
3957         NaN    30850.0      TOTANA (MURCIA)      532060
5277         NaN    30811.0      DIP. ORTILLO - LORCA      301240
5278         NaN    30811.0      DIP. ORTILLO - LORCA      1841585
5279         NaN    30811.0      DIP. ORTILLO - LORCA      581780
5280         NaN    30811.0      DIP. ORTILLO - LORCA      386520

```

[5 rows x 23 columns]

```
[18]: df[df.GPS_Longitud.isnull()].na_nombre.unique()
```

```
[18]: array(['PASO DEL PINO (OLD)', 'SALGADO (BAJA)'), dtype=object)
```

Si vemos los nombre de las naves (**na\_nombre**) de las granjas que no tenemos los datos de localización GPS, obtenemos: “PASO DEL PINO (OLD)” y “SALGADO (BAJA)”. Buscamos la localización de esta granja en otras filas del dataset, que se refieran a la misma granja para otro integrador y podemos llenar estos valores por los valores reales. Las granjas a buscar son: “PASO DEL PINO” y “SALGADO”.

```
[19]: granja_01 = 'PASO DEL PINO (OLD)'
latitud_01 = df[df.na_nombre=='PASO DEL PINO']['GPS_Latitud'].unique()
longitud_01 = df[df.na_nombre=='PASO DEL PINO']['GPS_Longitud'].unique()
print('Coordenadas Paso del Pino: Latitud=', latitud_01, ', Longitud=', longitud_01)
df.loc[df.na_nombre==granja_01, 'GPS_Latitud'] = latitud_01
df.loc[df.na_nombre==granja_01, 'GPS_Longitud'] = longitud_01

granja_02 = 'SALGADO (BAJA)'
latitud_02 = df[df.na_nombre=='SALGADO']['GPS_Latitud'].unique()
longitud_02 = df[df.na_nombre=='SALGADO']['GPS_Longitud'].unique()
print('Coordenadas Salgado: Latitud=', latitud_02, ', Longitud=', longitud_02)
```

```
df.loc[df.na_nombre==granja_02, 'GPS_Latitud'] = latitud_02[0]
df.loc[df.na_nombre==granja_02, 'GPS_Longitud'] = longitud_02[0]

print('Número de filas con nulos en GPS_Latitud', df.GPS_Latitud[df.
    ~GPS_Longitud.isnull()].sum())
```

Coordenadas Paso del Pino: Latitud= [37.69666] , Longitud= [-1.41345]

Coordenadas Salgado: Latitud= [37.7087] , Longitud= [-1.86494]

Número de filas con nulos en GPS\_Latitud 0.0

Ya no tenemos nulos en los campos de localización GPS y hemos podido recuperar los valores correctos de los mismos.

3) Para la columna gr\_codpos compruebo que sólo hay 3 filas, vemos los valores de los mismos.

[20]: df[df.gr\_codpos.isnull()]

```
ct_codigo ct_integra ct_tipo ct_raza ct_fase ct_sexo IncPeso \
5028 203585 585 1 84 2 2 96.288022
5095 203121 594 1 0 21 4 88.620976
5096 203698 594 1 80 21 4 96.320930

DiasMedios GMD EntradaInicial ... na_rega \
5028 120.135838 0.801493 2020-07-31 ... ES040530000627
5095 107.285173 0.826032 2020-04-07 ... ES040530000075
5096 107.721184 0.894169 2020-08-27 ... ES040530000075

se_nombre PesoEntMedio PesoRecMedio NumBajas GPS_Longitud \
5028 MACHO ENTERO + HEMBRA 17.038567 113.326590 34.0 -1.90652
5095 MACHO ENTERO 25.600294 114.221270 55.0 -1.93933
5096 MACHO ENTERO 20.633824 116.954753 73.0 -1.93933

GPS_Latitud gr_codpos gr_poblacion KgPiensoTotal
5028 37.4532 NaN 04600-HUERCAL OVERA (ALMERIA) 188100
5095 37.4519 NaN 04600-HUERCAL OVERA (ALMERIA) 221600
5096 37.4519 NaN 04600-HUERCAL OVERA (ALMERIA) 813600
```

[3 rows x 23 columns]

Como se puede ver en el campo gr\_poblacion tienen metido el código postal, por lo que podemos sacarlo de aquí. En las 3 filas es 04600.

[21]: df['gr\_codpos'] = df['gr\_codpos'].fillna('04600')
# Comprobamos que ya hemos quitado todos los nulos de la columna de CP
df[df.gr\_codpos.isnull()]

[21]: Empty DataFrame
Columns: [ct\_codigo, ct\_integra, ct\_tipo, ct\_raza, ct\_fase, ct\_sexo, IncPeso,
DiasMedios, GMD, EntradaInicial, EntradaFinal, NumAnimales, na\_nombre, na\_rega,

```
se_nombre, PesoEntMedio, PesoRecMedio, NumBajas, GPS_Longitud, GPS_Latitud,  
gr_codpos, gr_poblacion, KgPiensoTotal]  
Index: []  
[0 rows x 23 columns]
```

## 2.4 Outliers (Datos fuera de rango)

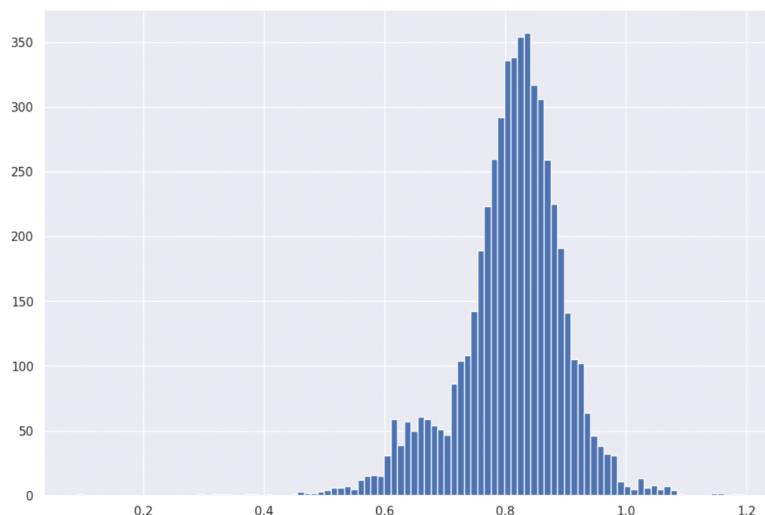
En este paso intentaremos analizar las columnas que tienen valores fuera de rango, y que su no tratamiento pueda distorsionar los cálculos que queremos realizar en una fase posterior.

### 2.4.1 Columna Objetivo (GMD)

El primer campo y más importante a analizar es el de la columna GMD, ya que es el objetivo a estimar, en esta columna los valores atípicos pueden influir de una manera muy decisiva y es muy importante detectarlos. Partimos de un histograma y un gráfico de caja para hacernos una idea de la distribución de los valores.

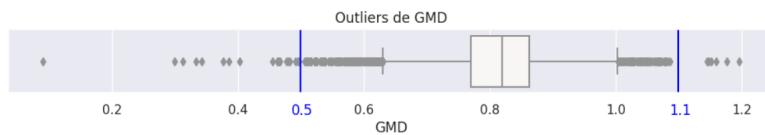
```
[22]: # histograma de GMD.  
df['GMD'].hist(bins=100)
```

```
[22]: <Axes: >
```



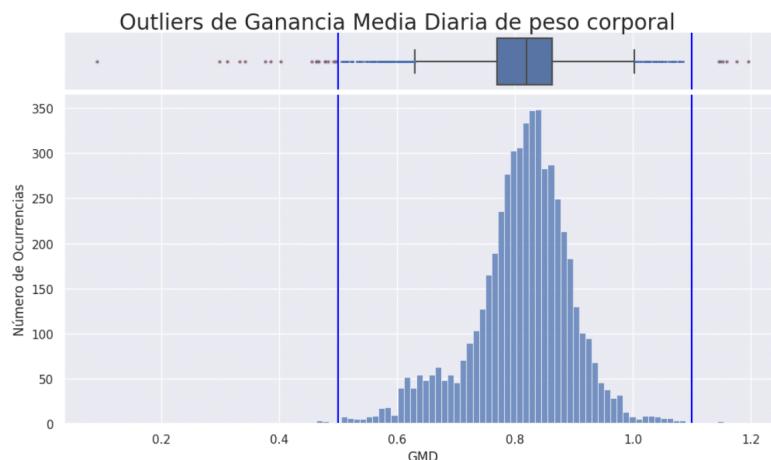
```
[29]: matplotlib.rcParams['figure.figsize'] = (12,1)
graph = sns.boxplot(data=df, x="GMD", palette="vlag")
graph.set(title='Outliers de GMD')
graph.axvline(0.5, color="blue")
graph.axvline(1.1, color="blue")
graph.text(x=0.5-0.015, y=0.85, s="0.5", color="blue")
graph.text(x=1.1-0.015, y=0.85, s="1.1", color="blue")
```

[29]: Text(1.0850000000000002, 0.85, '1.1')



Como se aprecia en el histograma la distribución se parece mucho a 2 distribuciones normales, la mayoritaria y centrada en 0.85 aproximadamente (del cerdo blanco) y la de ibérico centrada en 0.7 aproximadamente. En los extremos se observan unos pocos valores muy atípicos que deberíamos quitar. Paso a mostrar el detalle de estas 2 colas de valores.

```
[32]: # Mostramos gráfico conjunto de cajas y bigotes e histograma para GMD, ↴
      ↴remarcando los valores a eliminar
matplotlib.rcParams['figure.figsize'] = (10,6)
f, (ax_box, ax_hist) = plt.subplots(2, sharex=True, ↴
      ↴gridspec_kw={"height_ratios": (.15, .85)})
flierprops = dict(marker='o', markerfacecolor='r', markersize=1, ↴
      ↴linestyle='none', markeredgecolor='b')
sns.boxplot(x=df["GMD"], ax=ax_box, flierprops=flierprops)
sns.scatterplot(x=df.GMD[(df.GMD<0.5) | (df.GMD>1.1)], y=0, ax=ax_box, ↴
      ↴color='r', alpha=0.5, s=15)
sns.histplot(data=df, x="GMD", ax=ax_hist)
ax_box.axvline(0.5, color="blue")
ax_box.axvline(1.1, color="blue")
ax_hist.set(title="")
ax_hist.set(ylabel='Número de Ocurrencias')
ax_hist.axvline(0.5, color="blue")
ax_hist.axvline(1.1, color="blue")
ax_box.set(xlabel='')
f.suptitle('Outliers de Ganancia Media Diaria de peso corporal', fontsize=20)
f.tight_layout(pad=0.3)
plt.show()
```



```
[33] : df[df.GMD<=0.5]
```

```
[33] :   ct_codigo  ct_integra  ct_tipo  ct_raza  ct_fase  ct_sexo  IncPeso \
39      21158          1         1        7        2        2    79.924812
100     75104          1         2       69        2        8    65.086577
129     59014          1         1        0        2        2    19.455093
160     14941          1         2       69        2        8   12.831223
181     15179          1         2       81        2        8  103.925923
191     16535          1         2       69        2        8   42.597701
211     14942          1         2       69        2        8   12.610404
212     13545          1         2       69        2        8   38.733761
255     15378          1         2       81        2        8  109.862490
261     15779          1         2       81        2        8  100.088941
267     15272          1         2       81        2        8  104.465732
277     23802          1         2       69        2        8   38.650325
903     2609          160        2       69        2        8   72.207439
2771    205463         398        2       69        2        8  18.250000
4142    200881         504        1        0        2        2  15.000000
4167    200954         504        1        0        2        2   8.902439
4168     3164          504        1        0        2        2   47.162162
4203    201029         504        2       69        2        8   2.007362
4242    201031         504        1        0        2        2   12.290951

DiasMedios      GMD EntradaInicial ... na_rega \
39    162.320802  0.492388  2019-08-09 ... ES020370000205
```

100	131.341684	0.495552	2017-12-05	...	ES180890000206
129	58.274132	0.333855	2018-03-31	...	ES300302340012
160	26.587501	0.482604	2018-10-09	...	ES020370000167
181	216.645332	0.479705	2020-12-02	...	ES020370000167
191	85.784223	0.496568	2018-07-26	...	ES020370000233
211	27.701812	0.455219	2019-03-29	...	ES020370000167
212	81.158613	0.477260	2017-09-01	...	ES020370000167
255	235.240870	0.467021	2020-09-16	...	ES020370000167
261	214.815845	0.465929	2020-06-18	...	ES020370000167
267	225.736246	0.462778	2020-08-24	...	ES020370000167
277	95.991323	0.402644	2019-01-17	...	ES451570000015
903	155.919403	0.463107	2017-06-29	...	ES300243440013
2771	61.000000	0.299180	2021-09-15	...	ES300011440001
4142	48.000000	0.312500	2018-08-09	...	ES300261140006
4167	26.000000	0.342402	2018-08-31	...	ES300261140006
4168	125.216216	0.376646	2017-11-24	...	ES300261140006
4203	22.074118	0.090937	2018-09-26	...	ES300210540165
4242	31.819588	0.386270	2018-09-26	...	ES300390540029

	se_nombre	PesoEntMedio	PesoRecMedio	NumBajas	\
39	MACHO ENTERO + HEMBRA	30.000000	109.924812	31.0	
100	MACHO ENTERO + CATRADO+ HEMBRA	39.859739	104.946315	276.0	
129	MACHO ENTERO + HEMBRA	40.956072	60.411166	58.0	
160	MACHO ENTERO + CATRADO+ HEMBRA	140.249220	153.080443	9.0	
181	MACHO ENTERO + CATRADO+ HEMBRA	41.455078	145.381001	45.0	
191	MACHO ENTERO + CATRADO+ HEMBRA	33.658855	76.256556	219.0	
211	MACHO ENTERO + CATRADO+ HEMBRA	142.085176	154.695580	44.0	
212	MACHO ENTERO + CATRADO+ HEMBRA	108.032249	146.766011	52.0	
255	MACHO ENTERO + CATRADO+ HEMBRA	36.068333	145.930823	58.0	
261	MACHO ENTERO + CATRADO+ HEMBRA	41.032900	141.121841	47.0	
267	MACHO ENTERO + CATRADO+ HEMBRA	40.269841	144.735573	36.0	
277	MACHO ENTERO + CATRADO+ HEMBRA	60.000000	98.650325	23.0	
903	MACHO ENTERO + CATRADO+ HEMBRA	73.567164	145.774603	20.0	
2771	MACHO ENTERO + CATRADO+ HEMBRA	140.000000	158.250000	0.0	
4142	MACHO ENTERO + HEMBRA	25.000000	40.000000	2.0	
4167	MACHO ENTERO + HEMBRA	31.097561	40.000000	2.0	
4168	MACHO ENTERO + HEMBRA	90.000000	137.162162	13.0	
4203	MACHO ENTERO + CATRADO+ HEMBRA	145.122050	147.129412	37.0	
4242	MACHO ENTERO + HEMBRA	100.000000	112.290951	31.0	

	GPS_Longitud	GPS_Latitud	gr_codpos	gr_poblacion	\
39	-1.48801	38.42037	2499.0	CANCARIX, HELLIN (ALBACETE)	
100	-2.94016	37.49124	18500.0	GUADIX (GRANADA)	
129	-1.17603	38.00758	30107.0	MURCIA	
160	-1.52367	38.45461	2499.0	HELLIN	
181	-1.52367	38.45461	2499.0	HELLIN	
191	-1.52367	38.45461	2499.0	HELLIN	

211	-1.52367	38.45461	2499.0		HELLIN
212	-1.52367	38.45461	2499.0		HELLIN
255	-1.52367	38.45461	2499.0		HELLIN
261	-1.52367	38.45461	2499.0		HELLIN
267	-1.52367	38.45461	2499.0		HELLIN
277	-4.26492	40.15770	45370.0		TOLEDO
903	-1.90272	37.82722	30800.0	ZARCILLA DE RAMOS (LORCA)	
2771	-1.01195	38.16640	30640.0		ABANILLA
4142	-1.29353	37.60660	30870.0		MAZARRON (MURCIA)
4167	-1.29353	37.60660	30870.0		MAZARRON (MURCIA)
4168	-1.29353	37.60660	30870.0		MAZARRON (MURCIA)
4203	-1.19509	37.80025	30319.0	FUENTE ALAMO (MURCIA)	
4242	-1.47832	37.73890	30850.0		TOTANA (MURCIA)

#### KgPiensoTotal

39	410300
100	242520
129	406180
160	152060
181	321040
191	172580
211	245320
212	394440
255	339700
261	291240
267	302140
277	383760
903	388760
2771	458300
4142	113880
4167	452500
4168	438080
4203	142660
4242	494600

[19 rows x 23 columns]

[34] : df[df.GMD>=1.1]

	ct_codigo	ct_integra	ct_tipo	ct_raza	ct_fase	ct_sexo	IncPeso	\
123	84821	1	1	0	2	2	153.663761	
130	59012	1	1	0	2	2	44.806573	
131	59015	1	1	0	2	2	83.461764	
278	23805	1	2	81	2	8	57.355823	
295	204130	1	2	69	20	8	35.751105	
4159	200879	504	1	0	2	2	55.000000	

```

DiasMedios      GMD EntradaInicial .. na_rega \
123  132.547357  1.159312  2019-10-14 .. ES180450000019
130   38.901605  1.151792  2017-10-24 .. ES300302340012
131   70.938251  1.176541  2018-10-01 .. ES300302340012
278   47.963306  1.195827  2020-05-25 .. ES451570000015
295   31.138230  1.148142  2021-01-22 .. ES300191340001
4159  48.000000  1.145833  2018-08-09 .. ES300261140006

se_nombre PesoEntMedio PesoRecMedio NumBajas \
123      MACHO ENTERO + HEMBRA    47.205674  200.869435  152.0
130      MACHO ENTERO + HEMBRA    46.672828   91.479401   21.0
131      MACHO ENTERO + HEMBRA    18.900672  102.362436  103.0
278      MACHO ENTERO + CATRADO+ HEMBRA  32.156863  89.512686  264.0
295      MACHO ENTERO + CATRADO+ HEMBRA  25.000000  60.751105   25.0
4159     MACHO ENTERO + HEMBRA    25.000000  80.000000    3.0

GPS_Longitud GPS_Latitud gr_codpos gr_poblacion \
123      -2.71646  37.75253  18818.0 CASTILLEJAR (GRANADA)
130      -1.17603  38.00758  30107.0          MURCIA
131      -1.17603  38.00758  30107.0          MURCIA
278      -4.26492  40.15770  45370.0        TOLEDO
295      -1.46452  38.32480  30530.0       CIEZA (MURCIA)
4159     -1.29353  37.60660  30870.0      MAZARRON (MURCIA)

KgPiensoTotal
123      213060
130      410800
131      71800
278      175560
295      64620
4159     535640

[6 rows x 23 columns]

```

Veo correcto el número de líneas a eliminar quitando las que sean para valores de GMD menores o iguales a 0.5 o mayores o iguales a 1.1. El total de líneas que se eliminarán son  $19+6 = 25$  filas. Lo que es un número muy poco significativo en el tamaño de la muestra seleccionada inicialmente (que eran 5332 filas).

Procedo por tanto a eliminarlas.

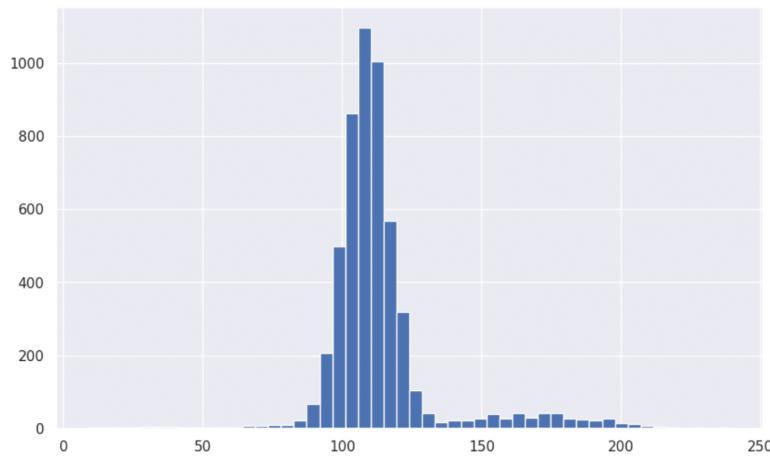
```
[35]: # Eliminamos los outliers de GMD (25 filas)
df.drop(df[df.GMD <= 0.5].index, inplace=True)
df.drop(df[df.GMD >= 1.1].index, inplace=True)
# Mostramos las filas y columnas que aún nos quedan en el dataframe
df.shape
```

[35]: (5307, 23)

## 2.4.2 Días de Engorde Medios

```
[36]: df.DiasMedios.hist(bins=50)
```

```
[36]: <Axes: >
```



Creo que sería razonable mostrar como candidatos a eliminar los menores de 50 días o mayores de 210 días.

```
[37]: df[df.DiasMedios<=50]
```

	ct_codigo	ct_integra	ct_tipo	ct_raza	ct_fase	ct_sexo	IncPeso	\
5	20316	1	2	69	20	8	29.724638	
125	59013	1	1	0	2	2	38.550571	
282	205123	1	2	69	20	8	21.889020	
283	203617	1	2	69	20	8	18.138455	
284	204804	1	2	69	20	8	20.406071	
285	204512	1	2	69	20	8	23.835784	
286	205019	1	2	69	20	8	17.666089	
287	203966	1	2	69	20	8	18.420934	
288	205752	1	2	69	20	8	26.797845	
289	204066	1	2	69	20	8	15.864016	
290	204368	1	2	69	20	8	23.029515	
291	203618	1	2	69	20	8	16.775227	
292	205387	1	2	69	20	8	32.255497	
293	203798	1	2	69	20	8	23.158567	

296	205239	1	2	69	20	8	27.183813
3423	204648	447	1	0	2	2	13.459485
4146	200979	504	1	0	2	2	11.000000
4147	201026	504	1	0	2	2	6.000000
4178	200880	504	1	0	2	2	20.844444
4207	201179	504	2	69	2	8	22.001969
4255	200921	504	1	15	2	2	18.295788

	DiasMedios	GMD	EntradaInicial	...	na_rega	\
5	46.289855	0.642142	2021-04-06	...	ES300080840002	
125	39.731789	0.970270	2017-12-31	...	ES300302340012	
282	34.725560	0.630343	2021-07-14	...	ES300191340001	
283	31.527604	0.575320	2020-08-13	...	ES300191340001	
284	38.132400	0.535137	2021-05-18	...	ES300191340001	
285	22.328431	1.067508	2021-03-04	...	ES300191340001	
286	34.646370	0.509897	2021-06-21	...	ES300191340001	
287	23.668147	0.778301	2020-10-27	...	ES300191340001	
288	43.246810	0.619649	2021-11-17	...	ES300191340001	
289	27.834669	0.569937	2020-11-19	...	ES300191340001	
290	30.143490	0.763996	2021-01-29	...	ES300191340001	
291	29.988491	0.559389	2020-09-04	...	ES300191340001	
292	45.887354	0.702928	2021-09-15	...	ES300191340001	
293	36.678018	0.631402	2020-09-22	...	ES300191340001	
296	38.479404	0.706451	2021-08-13	...	ES300191340001	
3423	23.999061	0.560834	2021-04-05	...	ES300261340002	
4146	16.000000	0.687500	2018-09-10	...	ES300261140006	
4147	9.000000	0.666667	2018-09-17	...	ES300261140006	
4178	37.000000	0.563363	2018-08-09	...	ES300261140006	
4207	34.000000	0.647117	2018-11-14	...	ES300210540165	
4255	32.545054	0.562168	2018-08-16	...	ES300161740043	

	se_nombre	PesoEntMedio	PesoRecMedio	NumBajas	\
5	MACHO ENTERO + CATRADO+ HEMBRA	128.260870	157.985507	0.0	
125	MACHO ENTERO + HEMBRA	51.476669	90.027240	41.0	
282	MACHO ENTERO + CATRADO+ HEMBRA	35.937500	57.826520	21.0	
283	MACHO ENTERO + CATRADO+ HEMBRA	22.982917	41.121372	6.0	
284	MACHO ENTERO + CATRADO+ HEMBRA	35.164030	55.570101	17.0	
285	MACHO ENTERO + CATRADO+ HEMBRA	27.500000	51.335784	4.0	
286	MACHO ENTERO + CATRADO+ HEMBRA	41.240000	58.906089	31.0	
287	MACHO ENTERO + CATRADO+ HEMBRA	24.148370	42.569304	24.0	
288	MACHO ENTERO + CATRADO+ HEMBRA	45.441264	72.239108	26.0	
289	MACHO ENTERO + CATRADO+ HEMBRA	27.806196	43.670213	13.0	
290	MACHO ENTERO + CATRADO+ HEMBRA	25.200573	48.230088	20.0	
291	MACHO ENTERO + CATRADO+ HEMBRA	23.611111	40.386338	14.0	
292	MACHO ENTERO + CATRADO+ HEMBRA	37.604067	69.859564	10.0	
293	MACHO ENTERO + CATRADO+ HEMBRA	22.279221	45.437788	21.0	
296	MACHO ENTERO + CATRADO+ HEMBRA	31.660000	58.843813	14.0	

3423	MACHO ENTERO + HEMBRA	22.084507	35.543992	8.0
4146	MACHO ENTERO + HEMBRA	30.000000	41.000000	0.0
4147	MACHO ENTERO + HEMBRA	20.000000	26.000000	0.0
4178	MACHO ENTERO + HEMBRA	19.155556	40.000000	3.0
4207	MACHO ENTERO + CATRADO+ HEMBRA	26.748031	48.750000	14.0
4255	MACHO ENTERO + HEMBRA	21.704212	40.000000	38.0

	GPS_Longitud	GPS_Latitud	gr_codpos	gr_poblacion \
5	-1.39403	37.84721	30840.0	ALHAMA DE MURCIA
125	-1.17603	38.00758	30107.0	MURCIA
282	-1.46452	38.32480	30530.0	CIEZA (MURCIA)
283	-1.46452	38.32480	30530.0	CIEZA (MURCIA)
284	-1.46452	38.32480	30530.0	CIEZA (MURCIA)
285	-1.46452	38.32480	30530.0	CIEZA (MURCIA)
286	-1.46452	38.32480	30530.0	CIEZA (MURCIA)
287	-1.46452	38.32480	30530.0	CIEZA (MURCIA)
288	-1.46452	38.32480	30530.0	CIEZA (MURCIA)
289	-1.46452	38.32480	30530.0	CIEZA (MURCIA)
290	-1.46452	38.32480	30530.0	CIEZA (MURCIA)
291	-1.46452	38.32480	30530.0	CIEZA (MURCIA)
292	-1.46452	38.32480	30530.0	CIEZA (MURCIA)
293	-1.46452	38.32480	30530.0	CIEZA (MURCIA)
296	-1.46452	38.32480	30530.0	CIEZA (MURCIA)
3423	-1.33239	37.65790	30870.0	MAZARRON (MURCIA)
4146	-1.29353	37.60660	30870.0	MAZARRON (MURCIA)
4147	-1.29353	37.60660	30870.0	MAZARRON (MURCIA)
4178	-1.29353	37.60660	30870.0	MAZARRON (MURCIA)
4207	-1.19509	37.80025	30319.0	FUENTE ALAMO (MURCIA)
4255	-0.99792	37.70060	30594.0	POZO ESTRECHO - CARTAGENA

	KgPiensoTotal
5	286920
125	404440
282	581800
283	206000
284	220540
285	160540
286	1006147
287	233700
288	302780
289	448020
290	267860
291	273480
292	518940
293	217580
296	194100
3423	72200

```

4146      398680
4147      727460
4178      340660
4207      240400
4255      314660

```

[21 rows x 23 columns]

[38] : df[df.DiasMedios>=210]

```

[38] :   ct_codigo  ct_integra  ct_tipo  ct_raza  ct_fase  ct_sexo    IncPeso \
762      200976      135        2       69        2        8  135.050994
766      201073      135        2       69        2        8  140.514860
1502     2873       256        2       69        2        8  122.743165
4204     201131      504        2       69        2        8  124.248636
4211     203649      504        2       69        2        8  123.370774
4219     201452      504        2       69        2        8  129.523709
4222     202417      504        2       69        2        8  122.681826
4224     201358      504        2       69        2        8  140.357554
4757     203399      548        2       69        2        8  123.987441

          DiasMedios      GMD EntradaInicial ... na_rega \
762    227.661543  0.593210  2018-09-04 ... ES300210340046
766    217.820675  0.645094  2018-10-11 ... ES300210340046
1502   214.207547  0.573010  2017-09-07 ... ES300390740055
4204   215.894214  0.575507  2018-10-24 ... ES300210540165
4211   211.114173  0.584379  2020-08-18 ... ES300210640034
4219   213.700215  0.606100  2019-01-09 ... ES300210640034
4222   214.688492  0.571441  2019-10-04 ... ES300210640034
4224   239.282859  0.586576  2018-12-26 ... ES300210640034
4757   210.578947  0.588793  2020-06-25 ... ES300330140087

          se_nombre PesoEntMedio PesoRecMedio NumBajas \
762    MACHO ENTERO + CATRADO+ HEMBRA      33.266092   168.317086   106.0
766    MACHO ENTERO + CATRADO+ HEMBRA      29.865320   170.380180    39.0
1502   MACHO ENTERO + CATRADO+ HEMBRA      34.510204   157.253369    21.0
4204   MACHO ENTERO + CATRADO+ HEMBRA      36.121200   160.369836   425.0
4211   MACHO ENTERO + CATRADO+ HEMBRA      28.676471   152.047244    54.0
4219   MACHO ENTERO + CATRADO+ HEMBRA      35.267857   164.791566    66.0
4222   MACHO ENTERO + CATRADO+ HEMBRA      32.877698   155.559524    52.0
4224   MACHO ENTERO + CATRADO+ HEMBRA      27.994580   168.352134    82.0
4757   MACHO ENTERO + CATRADO+ HEMBRA      29.071038   153.058480    24.0

          GPS_Longitud GPS_Latitud gr_codpos           gr_poblacion \
762      -1.24652    37.75651  30320.0  30320-FUENTE ALAMO (MURCIA)
766      -1.24652    37.75651  30320.0  30320-FUENTE ALAMO (MURCIA)
1502     -1.46582    37.69884  30850.0           PARETON-TOTANA

```

```

4204      -1.19509    37.80025    30319.0      FUENTE ALAMO (MURCIA)
4211      -1.18819    37.64211    30319.0      LAS PALAS (FUENTE ALAMO)
4219      -1.18819    37.64211    30319.0      LAS PALAS (FUENTE ALAMO)
4222      -1.18819    37.64211    30319.0      LAS PALAS (FUENTE ALAMO)
4224      -1.18819    37.64211    30319.0      LAS PALAS (FUENTE ALAMO)
4757      -1.72306    37.57477    30891.0      PUERTO LUMBRERAS

```

	KgPiensoTotal
762	529060
766	359820
1502	324120
4204	427120
4211	216180
4219	414780
4222	74780
4224	213560
4757	441900

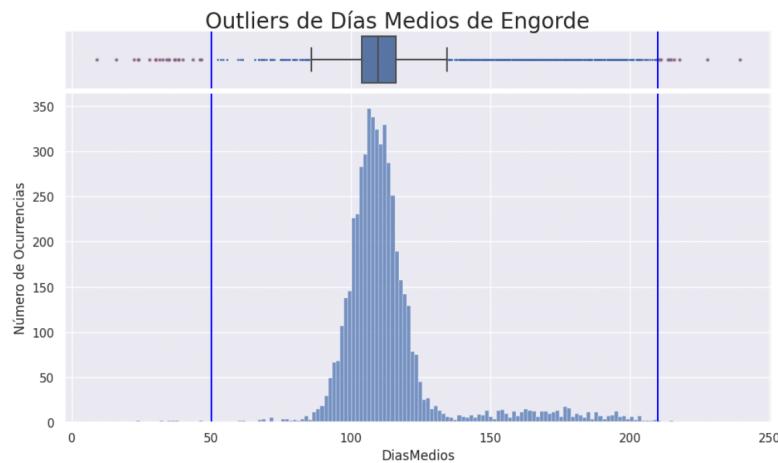
[9 rows x 23 columns]

Tras decidir procedente eliminar los valores señalados veo de mostrar un gráfico conjunto que resuma gráficamente los valores que vamos a eliminar.

```

[39]: # DiasMedios
f, (ax_box, ax_hist) = plt.subplots(2, sharex=True,
                                   gridspec_kw={"height_ratios": (.15, .85)})
flierprops = dict(marker='o', markerfacecolor='r', markersize=1,
                  linestyle='none', markeredgecolor='b')
sns.boxplot(x=df["DiasMedios"], ax=ax_box, flierprops=flierprops)
sns.scatterplot(x=df.DiasMedios[(df.DiasMedios<50) | (df.DiasMedios>210)], y=0,
                 ax=ax_box, color='r', alpha=0.5, s=15)
sns.histplot(data=df, x="DiasMedios", ax=ax_hist)
ax_box.axvline(50, color="blue")
ax_box.axvline(210, color="blue")
ax_hist.set(title="")
ax_hist.set(ylabel='Número de Ocurrencias')
ax_hist.axvline(50, color="blue")
ax_hist.axvline(210, color="blue")
ax_box.set(xlabel='')
f.suptitle('Outliers de Dias Medios de Engorde', fontsize=20)
f.tight_layout(pad=0.3)
plt.show()

```



```
[40]: # Eliminamos los outliers de DiasMedios (30 filas)
df.drop(df[df.DiasMedios <= 50].index, inplace=True)
df.drop(df[df.DiasMedios >= 210].index, inplace=True)
# Mostramos las filas y columnas que aún nos quedan en el dataframe
df.shape
```

```
[40]: (5277, 23)
```

Aún disponemos de 5.277 filas en nuestro dataset.

## 2.5 Añadir Columnas Calculadas

Con el objetivo de estimar el cálculo del GMD puede ser conveniente crear nuevas columnas trasformadas de las que disponemos para facilitar que afloren patrones que creemos que pueden estar en la muestra y permitan una mejor estimación del resultado. Por ejemplo los relativos a estacionalidad.

## 2.6 Detallar Fecha de Entrada

Como creemos que en el cálculo pueda tener influencia la estacionalidad, esta se puede mostrar más fácilmente si disponemos del número de semana de la entrada al contrato. Además aprovecharemos para añadir el año por si este tiene influencia y las mejoras en las granjas y procesos se notan en la evolución del GMD.

```
[41]: # Corrijo el tipo de los campos de tipo Fecha.
df['EntradaInicial'] = pd.to_datetime(df.entradaInicial, format='%Y-%m-%d')
df['EntradaFinal'] = pd.to_datetime(df.entradaFinal, format='%Y-%m-%d')
# Añadimos la Semana de Entrada y el Año de Entrada
```

```
df['semanaEntrada'] = df.EntradaInicial.dt.isocalendar().week
df['añoEntrada'] = df.EntradaInicial.dt.isocalendar().year
df.columns
```

```
[41]: Index(['ct_codigo', 'ct_integra', 'ct_tipo', 'ct_raza', 'ct_fase', 'ct_sexo',
       'IncPeso', 'DiasMedios', 'GMD', 'EntradaInicial', 'EntradaFinal',
       'NumAnimales', 'na_nombre', 'na_rega', 'se_nombre', 'PesoEntMedio',
       'PesoRecMedio', 'NumBajas', 'GPS_Longitud', 'GPS_Latitud', 'gr_codpos',
       'gr_poblacion', 'KgPiensoTotal', 'semanaEntrada', 'añoEntrada'],
      dtype='object')
```

## 2.7 Cuantificar Sexo

El campo Sexo en el dataset se representa por 2 campos un código ct\_sexo y su descripción se\_nombre. De este último se puede inferir que los animales pueden ser de los siguientes tipos: machos, hembras o machos castrados. Los castrados a nivel de crecimiento se comportan como las hembras, por lo que los podemos asemejar a estas. Los sexos de los contratos se componen de un porcentaje de animales de estos 3 tipos básicos de sexos. Por lo tanto sería más adecuado para el cálculo usar un porcentaje total de animales hembras/castrados, frente a machos. Podemos hacer un mapeo de estos valores en tanto por uno, con lo que añadiríamos la nueva columna del porcentaje de hembras (o castrados).

```
[42]: # Mostramos los valores que hay de los distintos sexos
df.value_counts(['ct_sexo','se_nombre'])
```

```
[42]: ct_sexo  se_nombre
2        MACHO ENTERO + HEMBRA      2495
4        MACHO ENTERO              1156
5        HEMBRA                   1104
8        MACHO ENTERO + CATRADO+ HEMBRA    485
3        MACHO CASTRADO              20
1        MACHO CASTRADO + HEMBRA        17
dtype: int64
```

```
[43]: # Añadimos el mapeo de los valores
sexos = { 1:1, 2:0.5, 3:1, 4:0, 5:1, 8:0.66 }
# Añadimos la columna de Porcentaje de Hembras
df['PorcHembras'] = df.ct_sexo.map(sexos)
# Comprobamos que la hemos creado correctamente
df.value_counts(['ct_sexo','se_nombre','PorcHembras'])
```

```
[43]:   ct_sexo  se_nombre      PorcHembras
2        MACHO ENTERO + HEMBRA      0.50      2495
4        MACHO ENTERO              0.00      1156
5        HEMBRA                   1.00      1104
8        MACHO ENTERO + CATRADO+ HEMBRA    0.66      485
3        MACHO CASTRADO              1.00      20
1        MACHO CASTRADO + HEMBRA        1.00      17
```

```
dtype: int64
```

Tras esto las 2 columnas anteriores podrían ser irrelevantes, y ser mucho más adecuada la nueva columna calculada. Por tanto podremos eliminar las anteriores del dataset.

```
[44]: # Eliminar las 2 columnas de sexo iniciales, las resume más pertinentemente la nueva columna de PorcHembras
      df.drop(['ct_sexo', 'se_nombre'], inplace=True, axis=1)
```

## 2.8 Pienso por cerda y día

Anteriormente se añadió la cantidad total en Kilos de pienso consumido por el contrato, pero para poder utilizarlo en la regresión es preferible pasarlo a una cantidad comparable directamente.

```
[45]: df["PiensoCerdaDia"] = df["KgPiensoTotal"] / (df["NumAnimales"] * df["DiasMedios"])
```

```
[46]: df.columns
```

```
[46]: Index(['ct_codigo', 'ct_integra', 'ct_tipo', 'ct_raza', 'ct_fase', 'IncPeso',
           'DiasMedios', 'GMD', 'EntradaInicial', 'EntradaFinal', 'NumAnimales',
           'na_nombre', 'na_rega', 'PesoEntMedio', 'PesoRecMedio', 'NumBajas',
           'GPS_Longitud', 'GPS_Latitud', 'gr_codpos', 'gr_poblacion',
           'KgPiensoTotal', 'semanaEntrada', 'añoEntrada', 'Porchembras',
           'PiensoCerdaDia'],
          dtype='object')
```

## 3 Datos Inconsistentes

### 3.1 Nombre de la Granja

La columna **na\_nombre** se corresponde con el nombre de la granja. Esta columna dice el nombre de la granja y viene definida por el integrador y nave concreta que se usa, por lo que si una misma granja a lo largo de los años tiene distintos integrados o está formada por distintas naves, puede darse el caso que tenga distintos nombres, porque o bien los antiguos se les añade un sufijo (OLD, baja, etc) o porque al escribirlos se cometa algún error tipográfico, por ejemplo es común que en ocasiones se le añada algún artículo y en otras no.

Para nuestro problema puede ser interesante corregir todo esto para que las granjas se puedan definir de forma más única y se pueda saber los distintos contratos que se refieren a la misma granja. Para ello lo primero que se me ocurre es poner todos los nombres en mayúsculas, y después quitar las stopwords como pueden ser "EL", "LA", "LOS", "LAS", "OLD", "BAJA". También todo lo que venga entre paréntesis, que son añadidos, por ejemplo indicando si esa granja es de tal integrador.

Para implementar todas estas modificaciones voy a definir un pequeño método en python que simplifique los nombres de las naves, con lo comentado anteriormente.

```
[47]: import re

def simplificar_nombre(cadena):
    # Pasamos a mayúsculas
    cadena = " " + cadena.upper() + " "
    sustituir = {"Á":"A", "É":"E", "Í":"I", "Ó":"O", "Ú":"U", "-": " ", ":" "", "\r\n": "", " OLD ":" ", " BAJA ":" ", " LA ":" ", " EL ":" ",
                 " LOS ":" ", " LAS ":" ", " IBERICO ":" ", " IB ":" ", " LECHONERA ":" ", " LECHONERAS ":" ", "\xa0":""}
    # Quitamos lo que esté entre paréntesis
    cadena = re.sub('(\.\*)', ' ', cadena)
    # Sustituimos los caracteres acentuados y las stopwords
    for cad1, cad2 in sustituir.items():
        cadena = re.sub(cad1, cad2, cadena)
    while ' ' in cadena:
        cadena = re.sub(' ', ' ', cadena)
    return cadena.strip()
```

Veo de crear una nueva columna con el nombre de la granja simplificado

```
[48]: df['na_nombre2'] = df.na_nombre.map(simplificar_nombre)
```

Muestro el número total de nombres de granjas distintos que tengo antes y después de aplicar la corrección del nombre para ver si ha merecido la pena.

```
[49]: print('na_nombre', str(len(df.na_nombre.unique())), ' filas')
print('na_nombre2', str(len(df.na_nombre2.unique())), ' filas')
```

```
na_nombre 597 filas
na_nombre2 491 filas
```

Como se ve se ha conseguido eliminar 106 ocurrencias de nombres de granjas que en realidad eran la misma y tenían nombres con alguna diferencia.

Podemos borrar la columna antigua del nombre que ya no necesitaremos.

```
[50]: # Borramos la columna del nombre original de la granja, porque ya disponemos de la simplificada
df.drop(['na_nombre'], inplace=True, axis=1)
```

## 4 Conclusiones

Tras todo el proceso de limpieza anterior nos encontramos con un dataset que dispone de 5.277 filas y 28 columnas. Partíamos de un dataset con 5.332 filas (hemos quitado 55 filas por distintos motivos) y que tenía 27 columnas, se han añadido varias columnas calculadas nuevas y se han eliminado otras que ya no eran necesarias, corrigiendo los datos por ejemplo de las columnas de tipo fecha.

El nuevo dataset está mucho más limpio, sin perder excesivas muestras en el proceso, ya no tiene missing values, y posee datos mejor preparados para poder usarlos en siguientes procesos de Machine

Learning.

Se podría eliminar también la columna del código del contrato, que es un identificador que ya no aporta nada de cara a la estimación que queremos lograr en posteriores pasos, pero de momento lo dejo por si en futuras versiones algún dato se ve como fuera de rango poder saber que contrato lo provoca y poder consultar más detalle del mismo.

```
[51]: df.info
```

```
[51]: <bound method DataFrame.info of
      ct_codigo  ct_integra  ct_tipo  ct_raza
      ct_fase    IncPeso  \
0        20312       1       2      69      2  128.863349
1        20315       1       2      69      2  112.100242
2        20313       1       2      69      2  119.896276
3        20311       1       2      69      2  115.469246
4        20314       1       2      69      2  115.154338
...
5327     205865      645      1      93     22  89.653104
5328     205864      645      1      93     21  93.623925
5329     205825      646      1      84     21  91.368970
5330     205625      647      1      84      2  90.636188
5331     205947      649      1      84      2  88.114371

      DiasMedios   GMD EntradaInicial EntradaFinal ... GPS_Longitud  \
0    197.617456  0.652085  2018-11-02  2018-12-27 ... -1.39403
1    180.365355  0.621518  2021-03-13  2021-04-23 ... -1.39403
2    191.990242  0.624492  2019-09-06  2019-10-03 ... -1.39403
3    196.899940  0.586436  2017-11-03  2017-12-07 ... -1.39403
4    184.792945  0.623153  2020-05-04  2020-06-12 ... -1.39403
...
5327    86.725438  1.033758  2021-12-17  2021-12-20 ... -1.70317
5328    86.724202  1.079559  2021-12-13  2021-12-17 ... -1.70317
5329   101.552088  0.899725  2021-12-07  2021-12-10 ... -4.28840
5330   101.254597  0.895132  2021-10-29  2021-11-03 ... -1.28144
5331   105.371496  0.836226  2021-12-31  2022-01-05 ... -1.28359

      GPS_Latitud gr_codpos      gr_poblacion KgPiensoTotal  \
0      37.84721  30840.0  ALHAMA DE MURCIA      136400
1      37.84721  30840.0  ALHAMA DE MURCIA      1180850
2      37.84721  30840.0  ALHAMA DE MURCIA      264020
3      37.84721  30840.0  ALHAMA DE MURCIA      303050
4      37.84721  30840.0  ALHAMA DE MURCIA      225440
...
5327    37.87150  30800.0  LORCA (MURCIA)      164040
5328    37.87150  30800.0  LORCA (MURCIA)      506100
5329    39.72323  45164.0          GALVEZ      309600
5330    37.73186  30338.0 FUENTE ALAMO (MURCIA)      285360
5331    37.69240  30335.0  LA PINILLA - FUENTE ALAMO      1460280
```

```
semanaEntrada añoEntrada PorcHembras PiensoCerdaDia na_nombre2
0              44      2018      0.66     0.111795  GRANJA 2
1              10      2021      0.66     1.178152  GRANJA 2
2              36      2019      0.66     0.222736  GRANJA 2
3              44      2017      0.66     0.373933  GRANJA 2
4              19      2020      0.66     0.197596  GRANJA 2
...
5327            50      2021      1.00     1.074708  PRADOS
5328            50      2021      0.00     3.039449  PRADOS
5329            49      2021      0.00     3.387424  HIGUERA
5330            43      2021      0.50     1.416202  CHORLITOS
5331            52      2021      0.50     7.860690  VIÑA VIEJA
```

[5277 rows x 25 columns]>

```
[52]: # Grabamos los datos del Dataset modificado
df.to_csv("gmd_02.csv", sep=";")
```



### 1.3. Regresión RandomForest

Enlace al cuaderno de trabajo en “Jupyter Notebook” para poder probar y generar los datos de preparación del dataset.

Se puede consultar el fichero en la siguiente URL: [Regresión RandomForest.ipynb](#)

Se adjunta el contenido del cuaderno como anexo.

# Regresión\_RandomForest

March 30, 2023

## 0.1 #Introducción

TFM: Aplicación de ciencia de datos en el sector de producción animal para la predicción y expli-  
cación de óptimos en ganado porcino.

*Titulo:* Regresión RandomForest

*Autor:* Jose Eduardo Cámará Gómez

---

## 1 Preparación y selección de variables para modelos

### 1.1 Importar paquetes

```
[1]: # Importación de paquetes
import numpy as np
import pandas as pd
import seaborn as sns

import matplotlib.pyplot as plt
import matplotlib.mlab as mlab
import matplotlib
from matplotlib.pyplot import figure

from sklearn.preprocessing import OneHotEncoder
from sklearn.ensemble import RandomForestRegressor
from sklearn.datasets import make_regression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import RobustScaler

sns.set(style="darkgrid")

[2]: from google.colab import files
# Cargamos el fichero del dataset gmd_02.csv
uploaded = files.upload()

<IPython.core.display.HTML object>
Saving gmd_02.csv to gmd_02.csv
```

```
[25]: # Leemos el fichero csv con los datos
df = pd.read_csv('gmd_02.csv', sep=';')

[26]: df.columns

[26]: Index(['Unnamed: 0', 'ct_codigo', 'ct_integra', 'ct_tipo', 'ct_raza',
       'ct_fase', 'IncPeso', 'DiasMedios', 'GMD', 'EntradaInicial',
       'EntradaFinal', 'NumAnimales', 'na_rega', 'PesoEntMedio',
       'PesoRecMedio', 'NumBajas', 'GPS_Longitud', 'GPS_Latitud', 'gr_codpos',
       'gr_poblacion', 'KgPiensoTotal', 'semanaEntrada', 'añoEntrada',
       'PorcHembras', 'PiensoCerdaDia', 'na_nombre2'],
      dtype='object')
```

## 1.2 Agrupar razas similares, para reducir categorías

```
[27]: # Revisar la raza si se agrupan las razas con menos ocurrencias
agrupar_razas = {93 : 93, 85 : 93, 90 : 93, 95 : 93, 94 : 93, 82 : 93, 80 : 80,
                 96 : 80, 88 : 88, 0 : 0, 23 : 0,
                 84 : 0, 66 : 0, 18 : 0, 68 : 88, 7 : 7, 89 : 7, 65 : 7, 15 : 15,
                 97 : 7, 69 : 69, 81 : 81}
df.replace({'ct_raza' : agrupar_razas}, inplace=True)
```

## 1.3 Expresar bajas en porcentaje de animales y ponderado por los días.

```
[28]: df["bajas"] = df["NumBajas"] / (df["NumAnimales"] * df["DiasMedios"])
```

## 1.4 Convertir tipos de variables Categóricas y Fecha

```
[29]: # Convertimos los tipos
df["ct_integra"] = df["ct_integra"].astype("category")
df["ct_tipo"] = df["ct_tipo"].astype("category")
df["ct_raza"] = df["ct_raza"].astype("category")
df["ct_fase"] = df["ct_fase"].astype("category")
df['EntradaInicial']= pd.to_datetime(df['EntradaInicial'])
df['EntradaFinal']= pd.to_datetime(df['EntradaFinal'])
df["na_rega"] = df["na_rega"].astype("category")
df["NumBajas"] = df["NumBajas"].astype("int64")
df["gr_codpos"] = df["gr_codpos"].astype("category")
df["gr_poblacion"] = df["gr_poblacion"].astype("category")
df["na_nombre2"] = df["na_nombre2"].astype("category")
```

## 1.5 Convertir variables categóricas a usar en OneHotEncoding

```
[30]: # Funcion para convertir en One Hot Encoding
def encode_and_bind(original_dataframe, feature_to_encode):
    dummies = pd.get_dummies(original_dataframe[[feature_to_encode]])
    res = pd.concat([original_dataframe, dummies], axis=1)
```

```
res = res.drop([feature_to_encode], axis=1)
return(res)
```

## 1.6 Seleccionar Variables a Utilizar

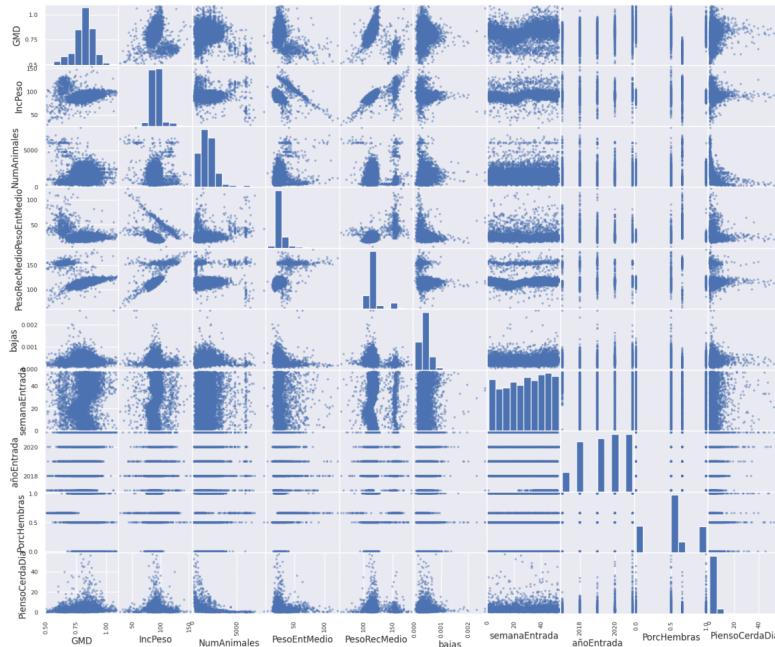
```
[31]: # Cargamos las variables objetivo y las usadas (15 variables seleccionadas, una de ellas categórica con 8 valores).
y = df['GMD']
x0 = df[['ct_integra', 'ct_tipo', 'ct_raza', 'IncPeso', 'NumAnimales', 'na_rega', 'PesoEntMedio', 'PesoRecMedio', 'bajas', 'GPS_Longitud', 'GPS_Latitud', 'semanaEntrada', 'añoEntrada', 'PorcHembras', 'PiensoCerdaDia']]
features_to_encode = ['ct_raza'] # , 'na_rega']
x1 = x0.copy()
x1.drop(['ct_integra', 'na_rega'], inplace=True, axis=1)
for feature in features_to_encode:
    x1 = encode_and_bind(x1, feature)
```

```
[65]: x1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5277 entries, 0 to 5276
Data columns (total 20 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   ct_tipo          5277 non-null   int64  
 1   IncPeso          5277 non-null   float64 
 2   NumAnimales      5277 non-null   int64  
 3   PesoEntMedio     5277 non-null   float64 
 4   PesoRecMedio     5277 non-null   float64 
 5   bajas            5277 non-null   float64 
 6   GPS_Longitud     5277 non-null   float64 
 7   GPS_Latitud      5277 non-null   float64 
 8   semanaEntrada    5277 non-null   int64  
 9   añoEntrada       5277 non-null   int64  
 10  PorcHembras      5277 non-null   float64 
 11  PiensoCerdaDia  5277 non-null   float64 
 12  ct_raza_0        5277 non-null   uint8  
 13  ct_raza_7        5277 non-null   uint8  
 14  ct_raza_15       5277 non-null   uint8  
 15  ct_raza_69       5277 non-null   uint8  
 16  ct_raza_80       5277 non-null   uint8  
 17  ct_raza_81       5277 non-null   uint8  
 18  ct_raza_88       5277 non-null   uint8  
 19  ct_raza_93       5277 non-null   uint8  
dtypes: float64(8), int64(4), uint8(8)
memory usage: 536.1 KB
```

## 1.7 Comprobar si hay variables dependientes

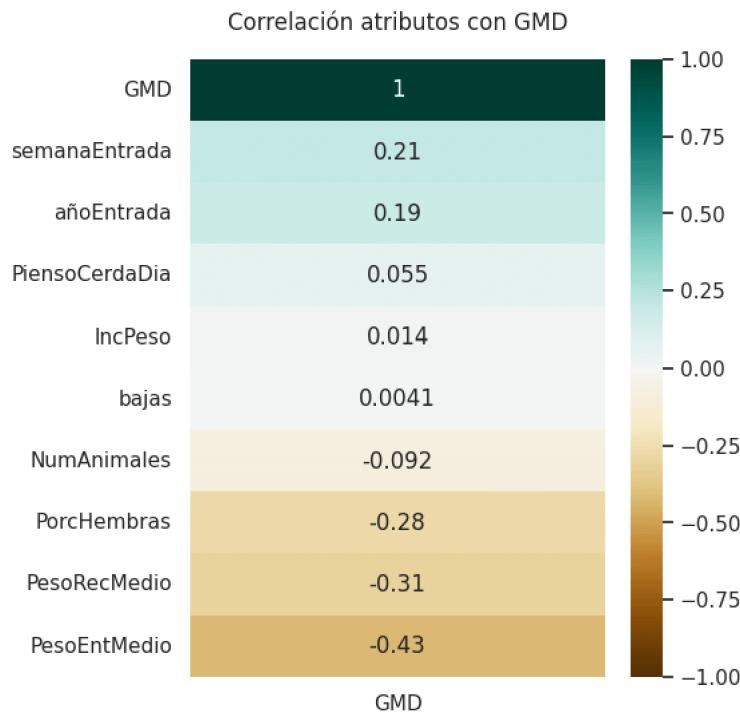
```
[10]: attributes = ['GMD', 'IncPeso', 'NumAnimales', 'PesoEntMedio', 'PesoRecMedio',  
    ↪'bajas', 'semanaEntrada', 'añoEntrada', 'PorcHembras', 'PiensoCerdaDia']  
pd.plotting.scatter_matrix(df[attributes], figsize=(18, 15))  
plt.show()
```



```
[11]: # Mostramos la matriz de correlaciones entre las principales variables.  
plt.figure(figsize=(16, 6))  
mask = np.triu(np.ones_like(df[attributes].corr(), dtype=bool))  
heatmap = sns.heatmap(df[attributes].corr(), mask=mask, vmin=-1, vmax=1,  
    ↪annot=True)  
heatmap.set_title('Matriz de correlaciones', fontdict={'fontsize':12}, pad=12);  
plt.show()
```



```
[12]: # Mostramos ordenadas las correlaciones de las variables con la variable Objetivo GMD.  
plt.figure(figsize=(5, 6))  
heatmap = sns.heatmap(df[attributes].corr()[['GMD']].sort_values(by='GMD', ascending=False), vmin=-1, vmax=1, annot=True, cmap='BrBG')  
heatmap.set_title('Correlación atributos con GMD', fontdict={'fontsize':12}, pad=16);
```



## 1.8 Dividir datos en entrenamiento y test

```
[50]: X_train, X_test, y_train, y_test = train_test_split(x1, y, test_size = 0.2, random_state = 123)
```

## 1.9 Escalar datos

```
[51]: ## Vemos de escalar las variables para que no se vean influenciadas por los outliers.  
scaler = RobustScaler()  
scaler.fit(X_train)  
X_train_s = scaler.transform(X_train)  
X_test_s = scaler.transform(X_test)
```

```
[52]: # Mostramos las columnas usadas para entrenamiento y sus tipos.  
X_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>  
Int64Index: 4221 entries, 3073 to 3582  
Data columns (total 20 columns):  
 #   Column            Non-Null Count  Dtype     
---  --     
 0   ct_tipo           4221 non-null    int64    
 1   IncPeso           4221 non-null    float64  
 2   NumAnimales       4221 non-null    int64    
 3   PesoEntMedio     4221 non-null    float64  
 4   PesoRecMedio     4221 non-null    float64  
 5   bajas             4221 non-null    float64  
 6   GPS_Longitud     4221 non-null    float64  
 7   GPS_Latitud      4221 non-null    float64  
 8   semanaEntrada    4221 non-null    int64    
 9   añoEntrada       4221 non-null    int64    
 10  PorcHembras      4221 non-null    float64  
 11  PiensoCerdaDia  4221 non-null    float64  
 12  ct_raza_0         4221 non-null    uint8    
 13  ct_raza_7         4221 non-null    uint8    
 14  ct_raza_15        4221 non-null    uint8    
 15  ct_raza_69        4221 non-null    uint8    
 16  ct_raza_80        4221 non-null    uint8    
 17  ct_raza_81        4221 non-null    uint8    
 18  ct_raza_88        4221 non-null    uint8    
 19  ct_raza_93        4221 non-null    uint8    
dtypes: float64(8), int64(4), uint8(8)  
memory usage: 461.7 KB
```

## 2 Modelo RandomForest

### 2.1 Creación del modelo

```
[53]: rf = RandomForestRegressor(  
          n_estimators = 100,  
          criterion   = 'squared_error',  
          max_depth   = None,  
          max_features = 'sqrt',  
          oob_score    = False,  
          n_jobs       = -1,  
          random_state = 123  
         )  
rf.fit(X_train_s, y_train)
```

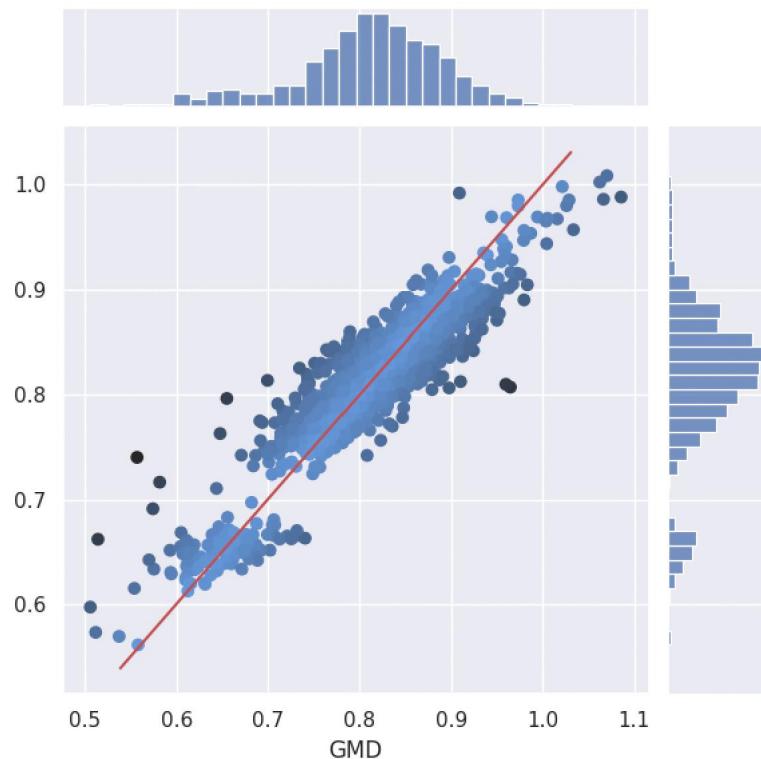
```
[53]: RandomForestRegressor(max_features='sqrt', n_jobs=-1, random_state=123)
```

## 2.2 Ver diferencias entre valor real y predicho en Test

```
[54]: # Función para Graficar diferencias entre valor predicho y real en datos de test del modelo pasado
def graficoDiferencias(modelo, X_test_s, y_test):
    y_pred = modelo.predict(X_test_s)
    diferencia = abs(y_pred - y_test)
    g = sns.jointplot(x=y_test, y=y_pred)
    # Draw a line of x=y
    x0, x1 = g.ax_joint.get_xlim()
    y0, y1 = g.ax_joint.get_ylim()
    lims = [max(x0, y0), min(x1, y1)]
    g.ax_joint.plot(lims, lims, '-r')
    g.ax_joint.scatter(x=y_test, y=y_pred, c=diferencia.values, cmap=sns.dark_palette("#69d", reverse=True, as_cmap=True))
    plt.show()

[55]: # Graficar las diferencias
print('Score R2:',rf.score(X_test_s, y_test))
graficoDiferencias(rf, X_test_s, y_test)
```

Score R2: 0.8332896022063712



```
[56]: # Analizamos otros errores del método
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import max_error
y_pred = rf.predict(X_test_s)
# Definimos la función con las métricas a mostrar
def mostrar_metricas(y_test, y_pred):
    print("Metr.\tValor\t\tDescripción")
    print("R^2 \t", r2_score(y_test, y_pred), "\t (Coeficiente de Determinación)")
    print("RMSE\t", mean_squared_error(y_test, y_pred, squared=True), "\t (Raíz de error cuadrático medio)")
```

```

    print("MAE \t", mean_absolute_error(y_test, y_pred), "\t (Error absoluto\u
    -medio)")
    print("MAX \t", max_error(y_test, y_pred), "\t (Error Máximo)")
# Pedimos que muestre las métricas para el modelo de RandomForest
print("Métricas para RandomForest v1")
mostrar_metricas(y_test, y_pred)

```

Métrica	Valor	Descripción
R^2	0.8332896022063712	(Coeficiente de Determinación)
RMSE	0.0011945882025570303	(Raíz de error cuadrático medio)
MAE	0.026452136729027687	(Error absoluto medio)
MAX	0.1832065150160347	(Error Máximo)

### 2.3 Variables más importantes según modelo

```
[57]: # Mostrar las variables más importantes
important_features_dict = {}
for idx, val in enumerate(rf.feature_importances_):
    important_features_dict[idx] = val

important_features_list = sorted(important_features_dict, u
    ↪key=important_features_dict.get, reverse=True)
print(f'Las 10 características más relevantes para la regresión son:')
print(' Orden\tCaracterística\tImportancia')
for i in range(10):
    print(f'\t{i+1}\t{x1.columns[important_features_list[i]]}\t{important_features_dict.get(important_features_list[i])}')

```

Las 10 características más relevantes para la regresión son:

Orden	Característica	Importancia
1	PesoRecMedio	0.22822026064823203
2	ct_raza_69	0.1153962404420927
3	ct_tipo	0.11438670034793824
4	PesoEntMedio	0.08694802628728203
5	IncPeso	0.07274022404653528
6	PorcHembras	0.06325723671377499
7	semanaEntrada	0.05678781772447855
8	NumAnimales	0.04124358743077506
9	ct_raza_93	0.040755746297112934
10	bajas	0.04064345282866445

### 2.4 Optimización de Hiperparámetros

Intentamos ver hasta dónde se pueden optimizar los hiperparámetros haciendo uso de una búsqueda aleatoria entre una gran variedad de valores de esos hiperparámetros.

#### 2.4.1 Definir hiperparámetros a optimizar y con qué posibles valores

```
[58]: # Hago una optimización de los hiperparámetros para RandomForest
from sklearn.model_selection import RandomizedSearchCV
random_grid = {'bootstrap': [True, False],
               'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, None],
               'max_features': [1.0, 'sqrt', 2, 5, 10, 20],
               'min_samples_leaf': [1, 2, 4, 10],
               'min_samples_split': [2, 5, 10, 20],
               'n_estimators': [20, 50, 75, 100, 150, 250, 500, 750]}
```

#### 2.4.2 Lanzar búsqueda de mejores parámetros

```
[59]: rf_random = RandomizedSearchCV(scoring="neg_mean_squared_error", estimator = rf,
                                     param_distributions = random_grid, n_iter = 300,
                                     cv = 3, verbose=1, random_state=123, n_jobs = -1)
rf_random.fit(X_train_s, y_train)
```

Fitting 3 folds for each of 300 candidates, totalling 900 fits

```
[59]: RandomizedSearchCV(cv=3,
                        estimator=RandomForestRegressor(max_features='sqrt',
                                                       n_jobs=-1,
                                                       random_state=123),
                        n_iter=300, n_jobs=-1,
                        param_distributions={'bootstrap': [True, False],
                                             'max_depth': [10, 20, 30, 40, 50, 60,
                                                           70, 80, 90, 100, 110,
                                                           None],
                                             'max_features': [1.0, 'sqrt', 2, 5, 10,
                                                               20],
                                             'min_samples_leaf': [1, 2, 4, 10],
                                             'min_samples_split': [2, 5, 10, 20],
                                             'n_estimators': [20, 50, 75, 100, 150,
                                                               250, 500, 750]},
                        random_state=123, scoring='neg_mean_squared_error',
                        verbose=1)
```

#### 2.4.3 Analizar mejor modelo y su error

```
[60]: rf_random.best_params_
```

```
[60]: {'n_estimators': 750,
       'min_samples_split': 2,
       'min_samples_leaf': 1,
       'max_features': 5,
       'max_depth': 60,
```



```
'bootstrap': False}
```

Tras probar aleatoriamente entre 300 combinaciones del rango de hiperparámetros propuesto la mejor solución para optimizar el error cuadrático medio ha sido la que se muestra. La búsqueda de los mejores hiperparámetros tardó en Google Colab 31 minutos, probando las 300 combinaciones para 3 particiones de los datos cada una, usando Cross Validation.

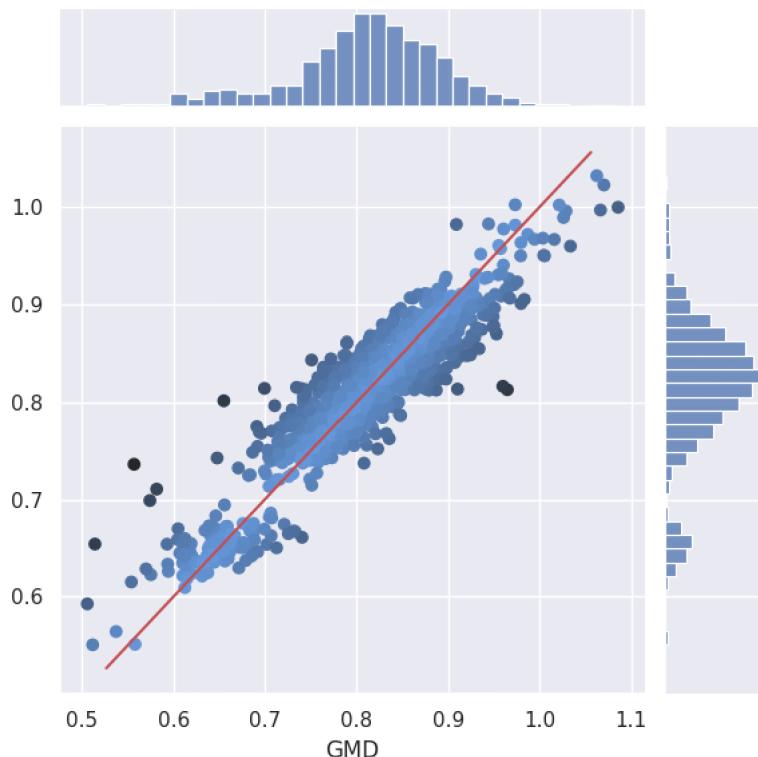
```
[61]: # Medimos las diferencias de la predicción para los valores de test (no usados en entrenamiento)
y_pred = rf_random.best_estimator_.predict(X_test_s)
mostrar_metricas(y_test, y_pred)
```

Metr.	Valor	Descripción
R^2	0.8412042580961172	(Coeficiente de Determinación)
RMSE	0.0011378745561479255	(Raíz de error cuadrático medio)
MAE	0.025939472844693622	(Error absoluto medio)
MAX	0.17919904826057953	(Error Máximo)

El error obtenido es de tan sólo -0.001137...

```
[62]: print('Score R2:',rf_random.best_estimator_.score(X_test_s, y_test))
graficoDiferencias(rf_random.best_estimator_, X_test_s, y_test)
```

Score R2: 0.8412042580961172



Hemos obtenido un coeficiente de determinación de 84,12%, mejorando el de los parámetros iniciales en un 0,8%.

Si medimos las diferencias sobre el conjunto de datos de test (no usado en el entrenamiento), el error se mantiene similar al obtenido en la validación cruzada del modelo, por lo que parece que no está sobreajustado y generaliza correctamente.

### 3 Estimar mejores modelos con LazzyPredict

La librería lazzypredict permite estimar los modelos que mejor representan nuestro modelo según una métrica dada, probando en más de 40 modelos y ofreciendo un ranking de los resultados. No ofrecen los mejores hiperparámetros para cada uno de esos modelos, pero es un buen punto de partida, para seleccionar los modelos más prometedores y realizar sobre ellos la optimización de

hiperparámetros, con la que encontrar más rápidamente un buen modelo que se aproxime a la mejor solución disponible con los métodos y variables actuales.

[63]: `!pip install lazypredict`

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting lazypredict
  Downloading lazypredict-0.2.12-py2.py3-none-any.whl (12 kB)
Requirement already satisfied: pandas in /usr/local/lib/python3.9/dist-packages (from lazypredict) (1.4.4)
Requirement already satisfied: xgboost in /usr/local/lib/python3.9/dist-packages (from lazypredict) (1.7.4)
Requirement already satisfied: tqdm in /usr/local/lib/python3.9/dist-packages (from lazypredict) (4.65.0)
Requirement already satisfied: lightgbm in /usr/local/lib/python3.9/dist-packages (from lazypredict) (3.3.5)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.9/dist-packages (from lazypredict) (1.2.2)
Requirement already satisfied: joblib in /usr/local/lib/python3.9/dist-packages (from lazypredict) (1.1.1)
Requirement already satisfied: click in /usr/local/lib/python3.9/dist-packages (from lazypredict) (8.1.3)
Requirement already satisfied: wheel in /usr/local/lib/python3.9/dist-packages (from lightgbm->lazypredict) (0.40.0)
Requirement already satisfied: scipy in /usr/local/lib/python3.9/dist-packages (from lightgbm->lazypredict) (1.10.1)
Requirement already satisfied: numpy in /usr/local/lib/python3.9/dist-packages (from lightgbm->lazypredict) (1.22.4)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.9/dist-packages (from scikit-learn->lazypredict) (3.1.0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.9/dist-packages (from pandas->lazypredict) (2022.7.1)
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.9/dist-packages (from pandas->lazypredict) (2.8.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.9/dist-packages (from python-dateutil>=2.8.1->pandas->lazypredict) (1.16.0)
Installing collected packages: lazypredict
Successfully installed lazypredict-0.2.12
```

[64]: `import lazypredict
from lazypredict.Supervised import LazyRegressor
# Borramos el modelo que tarda mucho
del lazypredict.Supervised.REGRESSORS[29:32] # PassiveAggressiveRegressor, u
 PoissonRegressor, QuantileRegressor
reg = LazyRegressor( verbose=1, ignore_warnings=False, custom_metric=None )
models, predictions = reg.fit(X_train_s, X_test_s, y_train, y_test)`

```
print(models)

3%|          | 1/39 [00:00<00:17,  2.16it/s]
{'Model': 'AdaBoostRegressor', 'R-Squared': 0.7294843224535585, 'Adjusted R-Squared': 0.7242569663657046, 'RMSE': 0.04402749602429004, 'Time taken': 0.4621419906616211}

10%|         | 4/39 [00:00<00:07,  4.97it/s]
{'Model': 'BaggingRegressor', 'R-Squared': 0.8087985942459139, 'Adjusted R-Squared': 0.8051038810912456, 'RMSE': 0.037014621034570525, 'Time taken': 0.3705098628997803}
{'Model': 'BayesianRidge', 'R-Squared': 0.6678693280291408, 'Adjusted R-Squared': 0.6614513440297038, 'RMSE': 0.04878454501228551, 'Time taken': 0.0506894588470459}
{'Model': 'DecisionTreeRegressor', 'R-Squared': 0.5652880485358663, 'Adjusted R-Squared': 0.5568878175896994, 'RMSE': 0.05581212484199173, 'Time taken': 0.10013699531555176}
{'Model': 'DummyRegressor', 'R-Squared': -0.00023142543513321456, 'Adjusted R-Squared': -0.01955956892180244, 'RMSE': 0.08465995131903602, 'Time taken': 0.015442609786987305}
{'Model': 'ElasticNet', 'R-Squared': -0.00023142543513321456, 'Adjusted R-Squared': -0.01955956892180244, 'RMSE': 0.08465995131903602, 'Time taken': 0.018407106399536133}

18%|         | 7/39 [00:01<00:03,  8.00it/s]
{'Model': 'ElasticNetCV', 'R-Squared': 0.6681781248918917, 'Adjusted R-Squared': 0.66176610798159, 'RMSE': 0.048761861142817164, 'Time taken': 0.16571950912475586}
{'Model': 'ExtraTreeRegressor', 'R-Squared': 0.5951761055935756, 'Adjusted R-Squared': 0.587353421643693, 'RMSE': 0.053859316296242195, 'Time taken': 0.03992414474487305}

26%|         | 10/39 [00:03<00:11,  2.61it/s]
{'Model': 'ExtraTreesRegressor', 'R-Squared': 0.8457726097022569, 'Adjusted R-Squared': 0.8427923702762135, 'RMSE': 0.0332436362606322, 'Time taken': 1.9740560054779053}
{'Model': 'GammaRegressor', 'R-Squared': 0.5607752921566995, 'Adjusted R-Squared': 0.552287858188713, 'RMSE': 0.05610107046939275, 'Time taken': 0.1643831729888916}

28%|         | 11/39 [00:12<01:04,  2.31s/it]
{'Model': 'GaussianProcessRegressor', 'R-Squared': -5.754981835711436, 'Adjusted R-Squared': -5.885512885676875, 'RMSE': 0.2200087024019368, 'Time taken': 8.994166851043701}

31%|         | 12/39 [00:14<01:01,  2.29s/it]
```

```
{'Model': 'GradientBoostingRegressor', 'R-Squared': 0.8165102364870722,
'Adjusted R-Squared': 0.8129645405737789, 'RMSE': 0.036260491564244754, 'Time
taken': 2.229362964630127}

33% | 13/39 [00:20<01:25, 3.29s/it]

{'Model': 'HistGradientBoostingRegressor', 'R-Squared': 0.8476511539323192,
'Adjusted R-Squared': 0.8447072148778711, 'RMSE': 0.033040556338584305, 'Time
taken': 6.170478820800781}

36% | 14/39 [00:21<01:02, 2.49s/it]

{'Model': 'HuberRegressor', 'R-Squared': 0.6128868887288887, 'Adjusted
R-Squared': 0.6054064421342779, 'RMSE': 0.05266798549079409, 'Time taken':
0.3008115291595459}

38% | 15/39 [00:21<00:46, 1.93s/it]

{'Model': 'KNeighborsRegressor', 'R-Squared': 0.7774162315974172, 'Adjusted
R-Squared': 0.7731150959761113, 'RMSE': 0.03993691484379072, 'Time taken':
0.42801809310913086}

46% | 18/39 [00:24<00:25, 1.24s/it]

{'Model': 'KernelRidge', 'R-Squared': -91.41536249120558, 'Adjusted R-Squared':
-93.20116659731585, 'RMSE': 0.8137665843939424, 'Time taken':
2.88581631183624268}

{'Model': 'Lars', 'R-Squared': 0.6679207205257995, 'Adjusted R-Squared':
0.6615037296180855, 'RMSE': 0.048780770509096766, 'Time taken':
0.0256050201721191406}

{'Model': 'LarsCV', 'R-Squared': 0.6681936306928815, 'Adjusted R-Squared':
0.6617819134115845, 'RMSE': 0.04876072182590559, 'Time taken':
0.07918930053710938}

{'Model': 'Lasso', 'R-Squared': -0.00023142543513321456, 'Adjusted R-Squared':
-0.01955956892180244, 'RMSE': 0.08465995131903602, 'Time taken':
0.029938936233520508}

51% | 20/39 [00:24<00:15, 1.22it/s]

{'Model': 'LassoCV', 'R-Squared': 0.6681910306152943, 'Adjusted R-Squared':
0.661779263090952, 'RMSE': 0.048760912873189954, 'Time taken':
0.28230786323547363}

{'Model': 'LassoLars', 'R-Squared': -0.00023142543513321456, 'Adjusted
R-Squared': -0.01955956892180244, 'RMSE': 0.08465995131903602, 'Time taken':
0.07087373733520508}

62% | 24/39 [00:25<00:06, 2.49it/s]

{'Model': 'LassoLarsCV', 'R-Squared': 0.6681936306928815, 'Adjusted R-Squared':
0.6617819134115845, 'RMSE': 0.04876072182590559, 'Time taken':
0.16783523559570312}

{'Model': 'LassoLarsIC', 'R-Squared': 0.6683368877931932, 'Adjusted R-Squared':
0.6619279387650423, 'RMSE': 0.04875019449068347, 'Time taken':
0.07441520690917969}
```

```
{'Model': 'LinearRegression', 'R-Squared': 0.6663880702676847, 'Adjusted R-Squared': 0.6599414629298621, 'RMSE': 0.04889321021693962, 'Time taken': 0.06666707992553711}

64% | 25/39 [00:25<00:06, 2.29it/s]

{'Model': 'LinearSVR', 'R-Squared': 0.5688952733890773, 'Adjusted R-Squared': 0.5605647472709918, 'RMSE': 0.05558007891373986, 'Time taken': 0.5810821056365967}

67% | 26/39 [00:28<00:11, 1.18it/s]

{'Model': 'MLPRegressor', 'R-Squared': 0.6629604534566966, 'Adjusted R-Squared': 0.6564476119775989, 'RMSE': 0.0491437392191756, 'Time taken': 2.312641143798828}

69% | 27/39 [00:31<00:17, 1.43s/it]

{'Model': 'NuSVR', 'R-Squared': 0.8224590191486887, 'Adjusted R-Squared': 0.819028275557359, 'RMSE': 0.035667861720169364, 'Time taken': 3.309314489364624}

{'Model': 'OrthogonalMatchingPursuit', 'R-Squared': 0.5294344783992626, 'Adjusted R-Squared': 0.5203414248417604, 'RMSE': 0.05806812705734184, 'Time taken': 0.013873577117919922}

{'Model': 'OrthogonalMatchingPursuitCV', 'R-Squared': 0.6209229536716963, 'Adjusted R-Squared': 0.6135977933561734, 'RMSE': 0.05211845234256676, 'Time taken': 0.022565841674804688}

77% | 30/39 [00:31<00:06, 1.34it/s]

{'Model': 'RANSACRegressor', 'R-Squared': -5.611708736322594e+21, 'Adjusted R-Squared': -5.720147552483417e+21, 'RMSE': 6341256605.834454, 'Time taken': 0.17681217193603516}

90% | 35/39 [00:35<00:02, 1.51it/s]

{'Model': 'RandomForestRegressor', 'R-Squared': 0.8322351812395536, 'Adjusted R-Squared': 0.8289933489929749, 'RMSE': 0.03467194562106037, 'Time taken': 3.8039724826812744}

{'Model': 'Ridge', 'R-Squared': 0.6679289247918978, 'Adjusted R-Squared': 0.6615120924207267, 'RMSE': 0.04878016792272903, 'Time taken': 0.014435291290283203}

{'Model': 'RidgeCV', 'R-Squared': 0.667928924792042, 'Adjusted R-Squared': 0.6615120924208737, 'RMSE': 0.048780167922718436, 'Time taken': 0.0195159912109375}

{'Model': 'SGDRegressor', 'R-Squared': 0.6680895261414477, 'Adjusted R-Squared': 0.6616757971779974, 'RMSE': 0.04876837058650609, 'Time taken': 0.020760536193847656}

{'Model': 'SVR', 'R-Squared': 0.7056216216914125, 'Adjusted R-Squared': 0.6999331506129857, 'RMSE': 0.04592833705970492, 'Time taken': 0.13870549201965332}

{'Model': 'TransformedTargetRegressor', 'R-Squared': 0.6663880702676847, 'Adjusted R-Squared': 0.6599414629298621, 'RMSE': 0.04889321021693962, 'Time taken': 0.02387833595275879}
```

```
{'Model': 'TweedieRegressor', 'R-Squared': 0.5589925924906035, 'Adjusted R-Squared': 0.5504707102198905, 'RMSE': 0.05621480506031914, 'Time taken': 0.026605606079101562}  
100%| 39/39 [00:36<00:00, 1.06it/s]  
{'Model': 'XGBRegressor', 'R-Squared': 0.8400534872446372, 'Adjusted R-Squared': 0.8369627333749684, 'RMSE': 0.03385440261176636, 'Time taken': 0.8272502422332764}  
{'Model': 'LGBMRegressor', 'R-Squared': 0.8476133664338951, 'Adjusted R-Squared': 0.8446686971862408, 'RMSE': 0.033044653654034827, 'Time taken': 0.1853652000427246}
```

Model	Adjusted R-Squared \
HistGradientBoostingRegressor	0.84
LGBMRegressor	0.84
ExtraTreesRegressor	0.84
XGBRegressor	0.84
RandomForestRegressor	0.83
NuSVR	0.82
GradientBoostingRegressor	0.81
BaggingRegressor	0.81
KNeighborsRegressor	0.77
AdaBoostRegressor	0.72
SVR	0.70
LassoLarsIC	0.66
LassoLarsCV	0.66
LarsCV	0.66
LassoCV	0.66
ElasticNetCV	0.66
SGDRegressor	0.66
RidgeCV	0.66
Ridge	0.66
Lars	0.66
BayesianRidge	0.66
LinearRegression	0.66
TransformedTargetRegressor	0.66
MLPRegressor	0.66
OrthogonalMatchingPursuitCV	0.61
HuberRegressor	0.61
ExtraTreeRegressor	0.59
LinearSVR	0.56
DecisionTreeRegressor	0.56
GammaRegressor	0.55
TweedieRegressor	0.55
OrthogonalMatchingPursuit	0.52
DummyRegressor	-0.02

LassoLars	-0.02	
ElasticNet	-0.02	
Lasso	-0.02	
GaussianProcessRegressor	-5.89	
KernelRidge	-93.20	
RANSACRegressor	-5720147552483417260032.00	
Model	R-Squared	RMSE \
HistGradientBoostingRegressor	0.85	0.03
LGBMRegressor	0.85	0.03
ExtraTreesRegressor	0.85	0.03
XGBRegressor	0.84	0.03
RandomForestRegressor	0.83	0.03
NuSVR	0.82	0.04
GradientBoostingRegressor	0.82	0.04
BaggingRegressor	0.81	0.04
KNeighborsRegressor	0.78	0.04
AdaBoostRegressor	0.73	0.04
SVR	0.71	0.05
LassoLarsIC	0.67	0.05
LassoLarsCV	0.67	0.05
LarsCV	0.67	0.05
LassoCV	0.67	0.05
ElasticNetCV	0.67	0.05
SGDRegressor	0.67	0.05
RidgeCV	0.67	0.05
Ridge	0.67	0.05
Lars	0.67	0.05
BayesianRidge	0.67	0.05
LinearRegression	0.67	0.05
TransformedTargetRegressor	0.67	0.05
MLPRegressor	0.66	0.05
OrthogonalMatchingPursuitCV	0.62	0.05
HuberRegressor	0.61	0.05
ExtraTreeRegressor	0.60	0.05
LinearSVR	0.57	0.06
DecisionTreeRegressor	0.57	0.06
GammaRegressor	0.56	0.06
TweedieRegressor	0.56	0.06
OrthogonalMatchingPursuit	0.53	0.06
DummyRegressor	-0.00	0.08
LassoLars	-0.00	0.08
ElasticNet	-0.00	0.08
Lasso	-0.00	0.08
GaussianProcessRegressor	-5.75	0.22
KernelRidge	-91.42	0.81
RANSACRegressor	-5611708736322594144256.00	6341256605.83

Model	Time Taken
HistGradientBoostingRegressor	6.17
LGBMRegressor	0.19
ExtraTreesRegressor	1.97
XGBRegressor	0.83
RandomForestRegressor	3.80
NuSVR	3.31
GradientBoostingRegressor	2.23
BaggingRegressor	0.37
KNeighborsRegressor	0.43
AdaBoostRegressor	0.46
SVR	0.14
LassoLarsIC	0.07
LassoLarsCV	0.17
LarsCV	0.08
LassoCV	0.28
ElasticNetCV	0.17
SGDRegressor	0.02
RidgeCV	0.02
Ridge	0.01
Lars	0.03
BayesianRidge	0.05
LinearRegression	0.07
TransformedTargetRegressor	0.02
MLPRegressor	2.31
OrthogonalMatchingPursuitCV	0.02
HuberRegressor	0.30
ExtraTreeRegressor	0.04
LinearSVR	0.58
DecisionTreeRegressor	0.10
GammaRegressor	0.16
TweedieRegressor	0.03
OrthogonalMatchingPursuit	0.01
DummyRegressor	0.02
LassoLars	0.07
ElasticNet	0.02
Lasso	0.03
GaussianProcessRegressor	8.99
KernelRidge	2.86
RANSACRegressor	0.18



## 1.4. HistGradientBoostingRegressor

Enlace al cuaderno de trabajo en “Jupyter Notebook” para poder probar y generar los datos de preparación del dataset.

Se puede consultar el fichero en la siguiente URL:  
[HistGradientBoostingRegressor.ipynb](#)

Se adjunta el contenido del cuaderno como anexo.

# HistGradientBoostingRegressor

April 2, 2023

## 0.1 #Introducción

TFM: Aplicación de ciencia de datos en el sector de producción animal para la predicción y explicación de óptimos en ganado porcino.

*Titulo:* HistGradientBoostingRegressor

*Autor:* Jose Eduardo Cámará Gómez

---

## 0.2 Importar paquetes

```
[1]: # Importación de paquetes
import numpy as np
import pandas as pd
import seaborn as sns

import matplotlib.pyplot as plt
import matplotlib.mlab as mlab
import matplotlib
from matplotlib.pyplot import figure

from sklearn.preprocessing import OneHotEncoder
from sklearn.ensemble import RandomForestRegressor
from sklearn.datasets import make_regression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import RobustScaler

from sklearn.ensemble import HistGradientBoostingRegressor

sns.set(style="darkgrid")

[2]: from google.colab import files
# Cargamos el fichero del dataset gmd_02.csv
uploaded = files.upload()
# Leemos el fichero csv con los datos
df = pd.read_csv('gmd_02.csv', sep=';')
# Revisar la raza si se agrupan las razas con menos ocurrencias
```

```
agrupar_razas = {93 : 93, 85 : 93, 90 : 93, 95 : 93, 94 : 93, 82 : 93, 80 : 80, 96 : 80, 88 : 88, 0 : 0, 23 : 0, 84 : 0, 66 : 0, 18 : 0, 68 : 88, 7 : 7, 89 : 7, 65 : 7, 15 : 15, 97 : 7, 69 : 69, 81 : 81}
df.replace({'ct_raza' : agrupar_razas}, inplace=True)
df["bajas"] = df["NumBajas"] / (df["NumAnimales"] * df["DiasMedios"])
# Convertimos los tipos
df["ct_integra"] = df["ct_integra"].astype("category")
#df["ct_tipo"] = df["ct_tipo"].astype("category")
df["ct_raza"] = df["ct_raza"].astype("category")
df["ct_fase"] = df["ct_fase"].astype("category")
df['EntradaInicial']= pd.to_datetime(df['EntradaInicial'])
df['EntradaFinal']= pd.to_datetime(df['EntradaFinal'])
df["na_rega"] = df["na_rega"].astype("category")
df["NumBajas"] = df["NumBajas"].astype("int64")
df["gr_codpos"] = df["gr_codpos"].astype("category")
df["gr_poblacion"] = df["gr_poblacion"].astype("category")
df["na_nombre2"] = df["na_nombre2"].astype("category")

# Funcion para convertir en One Hot Encoding
def encode_and_bind(original_dataframe, feature_to_encode):
    dummies = pd.get_dummies(original_dataframe[[feature_to_encode]])
    res = pd.concat([original_dataframe, dummies], axis=1)
    res = res.drop([feature_to_encode], axis=1)
    return(res)

# Cargamos las variables objetivo y las usadas (15 variables seleccionadas, una de ellas categórica con 8 valores).
y = df['GMD']
x0 = df[['ct_integra', 'ct_tipo', 'ct_raza', 'IncPeso', 'NumAnimales', 'na_rega', 'PesoEntMedio', 'PesoRecMedio', 'bajas', 'GPS_Longitud', 'GPS_Latitud', 'semanaEntrada', 'añoEntrada', 'PorcHembras', 'PiensoCerdaDia']]
features_to_encode = ['ct_raza'] # , 'na_rega']
x1 = x0.copy()
x1.drop(['ct_integra', 'na_rega'], inplace=True, axis=1)
for feature in features_to_encode:
    x1 = encode_and_bind(x1, feature)

<IPython.core.display.HTML object>
Saving gmd_02.csv to gmd_02.csv

[3]: X_train, X_test, y_train, y_test = train_test_split(x1, y, test_size = 0.2, random_state = 123)
## Vemos de escalar las variables para que no se vean influenciadas por los outliers.
scaler = RobustScaler()
scaler.fit(X_train)
X_train_s = scaler.transform(X_train)
```

```
X_test_s = scaler.transform(X_test)

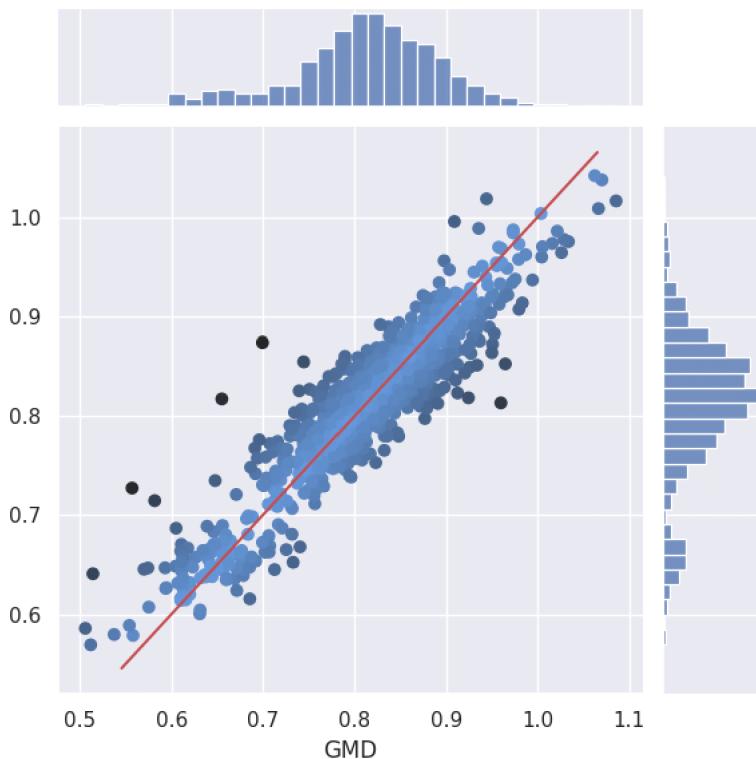
[4]: # Lanzamos el modelo con los parámetros por defecto
      modelo = HistGradientBoostingRegressor(
          loss = "squared_error",
          learning_rate=0.1,
          max_iter=100,
          max_leaf_nodes=31,
          random_state = 123
      )
      modelo.fit(X_train_s, y_train)

[4]: HistGradientBoostingRegressor(random_state=123)

[5]: # Función para Graficar diferencias entre valor predicho y real en datos de test del modelo pasado
      def graficoDiferencias(modelo, X_test_s, y_test):
          y_pred = modelo.predict(X_test_s)
          diferencia = abs(y_pred - y_test)
          g = sns.jointplot(x=y_test, y=y_pred)
          # Draw a line of x=y
          x0, x1 = g.ax_joint.get_xlim()
          y0, y1 = g.ax_joint.get_ylim()
          lims = [max(x0, y0), min(x1, y1)]
          g.ax_joint.plot(lims, lims, '-r')
          g.ax_joint.scatter(x=y_test, y=y_pred, c=diferencia.values, cmap=sns.
          dark_palette("#69d", reverse=True, as_cmap=True))
          plt.show()

      # Graficar las diferencias
      print('Score R2:', modelo.score(X_test_s, y_test))
      graficoDiferencias(modelo, X_test_s, y_test)
```

Score R2: 0.8476511539323192



```
[6]: # Analizamos otros errores del método
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import max_error
y_pred = modelo.predict(X_test_s)
# Definimos la función con las métricas a mostrar
def mostrar_metricas(y_test, y_pred):
    print("Metr.\tValor\t\tDescripción")
    print("R^2 \t", r2_score(y_test, y_pred), "\t (Coeficiente de Determinación)")
    print("RMSE\t", mean_squared_error(y_test, y_pred, squared=True), "\t (Raíz de error cuadrático medio)")
```

```
    print("MAE \t", mean_absolute_error(y_test, y_pred), "\t (Error absoluto\u20acmedio)")
    print("MAX \t", max_error(y_test, y_pred), "\t (Error Máximo)")
# Pedimos que muestre las métricas para el modelo de RandomForest
print("Métricas para RandomForest v1")
mostrar_metricas(y_test, y_pred)
```

Métricas para RandomForest v1

Metr.	Valor	Descripción
R^2	0.8476511539323192	(Coeficiente de Determinación)
RMSE	0.0010916783631631634	(Raíz de error cuadrático medio)
MAE	0.02515203975776052	(Error absoluto medio)
MAX	0.17427550398957703	(Error Máximo)

## 1 Optimización de Hiperparámetros

```
[7]: # Hacemos una optimización de los hiperparámetros básica para RandomForest y\u20acvemos de probar de forma aleatoria con la combinación de 100 de estos\u20acmodelos a ver cuál es el que mejor ajusta el MSRE.
from sklearn.model_selection import RandomizedSearchCV
random_grid = {'learning_rate': [0.0001, 0.001, 0.01, 0.1, 0.2, 0.25, 0.3, 0.5],
               'max_iter': [10, 30, 50, 100, 200, 500, 1000],
               'max_leaf_nodes': [5, 10, 20, 30, 50, 100, 200]}
modelo_random = RandomizedSearchCV(scoring="neg_mean_squared_error", estimator\u20ac= modelo,
                                      param_distributions = random_grid, n_iter = 100,
                                      cv = 3, verbose=1, random_state=123, n_jobs = -1)
modelo_random.fit(X_train_s, y_train)
```

Fitting 3 folds for each of 100 candidates, totalling 300 fits

```
[7]: RandomizedSearchCV(cv=3,
                        estimator=HistGradientBoostingRegressor(random_state=123),
                        n_iter=100, n_jobs=-1,
                        param_distributions={'learning_rate': [0.0001, 0.001, 0.01,
                                                               0.1, 0.2, 0.25, 0.3,
                                                               0.5],
                                             'max_iter': [10, 30, 50, 100, 200, 500,
                                                          1000],
                                             'max_leaf_nodes': [5, 10, 20, 30, 50,
                                                                100, 200]},
                        random_state=123, scoring='neg_mean_squared_error',
                        verbose=1)
```

```
[9]: modelo_random.best_params_
```

```
[9]: {'max_leaf_nodes': 5, 'max_iter': 1000, 'learning_rate': 0.1}
```

```
[8]: print("Estimadores para Train")
mostrar_metricas(y_train, modelo_random.best_estimator_.predict(X_train_s))
print("")
print("Estimadores para Test")
mostrar_metricas(y_test, modelo_random.best_estimator_.predict(X_test_s))
```

Estimadores para Train

Metr.	Valor	Descripción
R^2	0.9181345063853255	(Coeficiente de Determinación)
RMSE	0.0005632649816280457	(Raíz de error cuadrático medio)
MAE	0.01841218605638466	(Error absoluto medio)
MAX	0.1369394846401235	(Error Máximo)

Estimadores para Test

Metr.	Valor	Descripción
R^2	0.8438046323594022	(Coeficiente de Determinación)
RMSE	0.0011192411867944516	(Raíz de error cuadrático medio)
MAE	0.02530588487320156	(Error absoluto medio)
MAX	0.17435190880309026	(Error Máximo)

Como se puede apreciar el método parece mejorar en entrenamiento, pero es únicamente sobreajustándose al modelo, ya que si vemos el error en datos de test este no ha mejorado nada, más bien ha empeorado ligeramente.

Intentamos ahora hacerlo con early\_stopping. Este modelo permite ser lanzado con la opción de early stopping, y es esta buscar optimizar no los resultados para train si no para test (reserva un porcentaje de los datos de entrenamiento para realizar la validación con datos no usados). Podemos ponerle un número de iteraciones máximas muy grande pues se espera que acabe antes el proceso por no mejorar según las condiciones de early stopping.

```
[10]: modelo2 = HistGradientBoostingRegressor(
        loss = "squared_error",
        learning_rate=0.1,
        max_iter=50000,
        max_leaf_nodes=31,
        early_stopping=True,
        scoring='loss',
        validation_fraction=0.1,
        n_iter_no_change=10,
        tol=1e-07,
        verbose=1,
        random_state = 123
    )
modelo2.fit(X_train_s, y_train)
# Mostramos los errores mediante este método
print("Estimadores para Train")
mostrar_metricas(y_train, modelo2.predict(X_train_s))
print("")
print("Estimadores para Test")
```

```
mostrar_metricas(y_test, modelo2.predict(X_test_s))
## Mostramos la gráfica del entrenamiento
graficoDiferencias(modelo2, X_test_s, y_test)

Binning 0.001 GB of training data: 0.022 s
Binning 0.000 GB of validation data: 0.001 s
Fitting gradient boosted rounds:
[1/50000] 1 tree, 31 leaves, max depth = 9, train loss: 0.00293, val loss:
0.00315, in 0.021s
[2/50000] 1 tree, 31 leaves, max depth = 9, train loss: 0.00253, val loss:
0.00277, in 0.011s
[3/50000] 1 tree, 31 leaves, max depth = 9, train loss: 0.00220, val loss:
0.00247, in 0.010s
[4/50000] 1 tree, 31 leaves, max depth = 8, train loss: 0.00194, val loss:
0.00222, in 0.011s
[5/50000] 1 tree, 31 leaves, max depth = 9, train loss: 0.00172, val loss:
0.00200, in 0.013s
[6/50000] 1 tree, 31 leaves, max depth = 8, train loss: 0.00154, val loss:
0.00183, in 0.011s
[7/50000] 1 tree, 31 leaves, max depth = 8, train loss: 0.00139, val loss:
0.00168, in 0.012s
[8/50000] 1 tree, 31 leaves, max depth = 9, train loss: 0.00127, val loss:
0.00156, in 0.012s
[9/50000] 1 tree, 31 leaves, max depth = 8, train loss: 0.00116, val loss:
0.00146, in 0.012s
[10/50000] 1 tree, 31 leaves, max depth = 9, train loss: 0.00107, val loss:
0.00138, in 0.012s
[11/50000] 1 tree, 31 leaves, max depth = 10, train loss: 0.00100, val loss:
0.00132, in 0.011s
[12/50000] 1 tree, 31 leaves, max depth = 8, train loss: 0.00094, val loss:
0.00126, in 0.011s
[13/50000] 1 tree, 31 leaves, max depth = 8, train loss: 0.00088, val loss:
0.00120, in 0.012s
[14/50000] 1 tree, 31 leaves, max depth = 10, train loss: 0.00084, val loss:
0.00116, in 0.012s
[15/50000] 1 tree, 31 leaves, max depth = 9, train loss: 0.00080, val loss:
0.00112, in 0.011s
[16/50000] 1 tree, 31 leaves, max depth = 9, train loss: 0.00076, val loss:
0.00109, in 0.011s
[17/50000] 1 tree, 31 leaves, max depth = 10, train loss: 0.00073, val loss:
0.00106, in 0.011s
[18/50000] 1 tree, 31 leaves, max depth = 10, train loss: 0.00071, val loss:
0.00104, in 0.022s
[19/50000] 1 tree, 31 leaves, max depth = 9, train loss: 0.00068, val loss:
0.00101, in 0.017s
[20/50000] 1 tree, 31 leaves, max depth = 8, train loss: 0.00066, val loss:
0.00099, in 0.010s
[21/50000] 1 tree, 31 leaves, max depth = 11, train loss: 0.00064, val loss:
```

0.00097, in 0.010s  
[22/50000] 1 tree, 31 leaves, max depth = 12, train loss: 0.00063, val loss:  
0.00096, in 0.014s  
[23/50000] 1 tree, 31 leaves, max depth = 12, train loss: 0.00061, val loss:  
0.00094, in 0.011s  
[24/50000] 1 tree, 31 leaves, max depth = 10, train loss: 0.00059, val loss:  
0.00093, in 0.010s  
[25/50000] 1 tree, 31 leaves, max depth = 9, train loss: 0.00058, val loss:  
0.00091, in 0.013s  
[26/50000] 1 tree, 31 leaves, max depth = 9, train loss: 0.00057, val loss:  
0.00090, in 0.012s  
[27/50000] 1 tree, 31 leaves, max depth = 10, train loss: 0.00055, val loss:  
0.00089, in 0.010s  
[28/50000] 1 tree, 31 leaves, max depth = 7, train loss: 0.00054, val loss:  
0.00088, in 0.010s  
[29/50000] 1 tree, 31 leaves, max depth = 10, train loss: 0.00053, val loss:  
0.00087, in 0.026s  
[30/50000] 1 tree, 31 leaves, max depth = 11, train loss: 0.00052, val loss:  
0.00086, in 0.012s  
[31/50000] 1 tree, 31 leaves, max depth = 8, train loss: 0.00051, val loss:  
0.00085, in 0.011s  
[32/50000] 1 tree, 31 leaves, max depth = 14, train loss: 0.00050, val loss:  
0.00084, in 0.011s  
[33/50000] 1 tree, 31 leaves, max depth = 9, train loss: 0.00049, val loss:  
0.00084, in 0.010s  
[34/50000] 1 tree, 31 leaves, max depth = 12, train loss: 0.00049, val loss:  
0.00083, in 0.014s  
[35/50000] 1 tree, 31 leaves, max depth = 10, train loss: 0.00048, val loss:  
0.00083, in 0.011s  
[36/50000] 1 tree, 31 leaves, max depth = 8, train loss: 0.00047, val loss:  
0.00082, in 0.010s  
[37/50000] 1 tree, 31 leaves, max depth = 11, train loss: 0.00046, val loss:  
0.00081, in 0.013s  
[38/50000] 1 tree, 31 leaves, max depth = 7, train loss: 0.00046, val loss:  
0.00081, in 0.009s  
[39/50000] 1 tree, 31 leaves, max depth = 13, train loss: 0.00045, val loss:  
0.00080, in 0.011s  
[40/50000] 1 tree, 31 leaves, max depth = 11, train loss: 0.00044, val loss:  
0.00080, in 0.009s  
[41/50000] 1 tree, 31 leaves, max depth = 13, train loss: 0.00044, val loss:  
0.00079, in 0.013s  
[42/50000] 1 tree, 31 leaves, max depth = 14, train loss: 0.00043, val loss:  
0.00079, in 0.010s  
[43/50000] 1 tree, 31 leaves, max depth = 11, train loss: 0.00043, val loss:  
0.00079, in 0.011s  
[44/50000] 1 tree, 31 leaves, max depth = 9, train loss: 0.00042, val loss:  
0.00078, in 0.013s  
[45/50000] 1 tree, 31 leaves, max depth = 13, train loss: 0.00042, val loss:

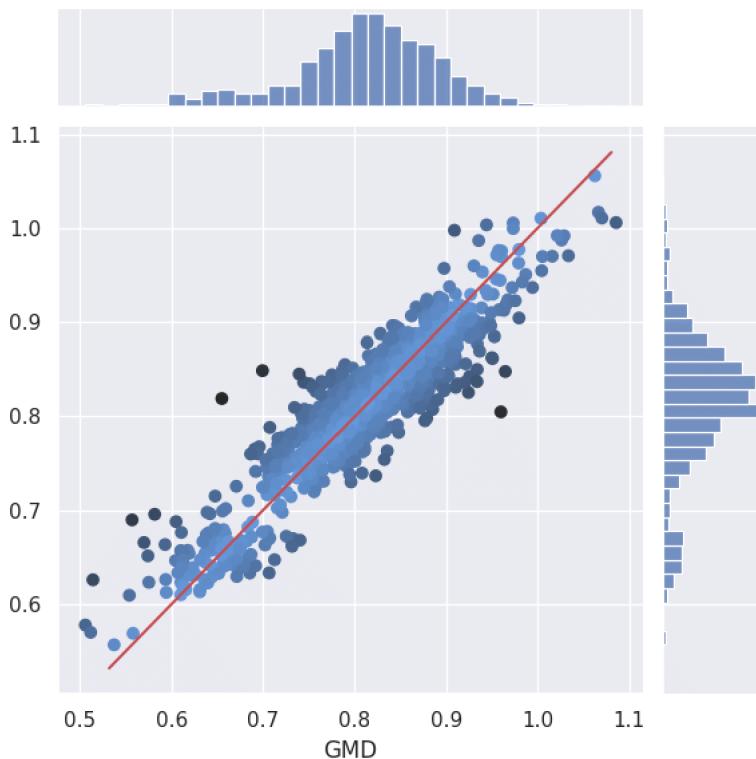
```
0.00078, in 0.010s
[46/50000] 1 tree, 31 leaves, max depth = 11, train loss: 0.00041, val loss:
0.00078, in 0.013s
[47/50000] 1 tree, 31 leaves, max depth = 10, train loss: 0.00040, val loss:
0.00077, in 0.015s
[48/50000] 1 tree, 31 leaves, max depth = 10, train loss: 0.00040, val loss:
0.00077, in 0.010s
[49/50000] 1 tree, 31 leaves, max depth = 13, train loss: 0.00039, val loss:
0.00077, in 0.010s
[50/50000] 1 tree, 31 leaves, max depth = 12, train loss: 0.00039, val loss:
0.00077, in 0.014s
[51/50000] 1 tree, 31 leaves, max depth = 8, train loss: 0.00038, val loss:
0.00077, in 0.011s
[52/50000] 1 tree, 31 leaves, max depth = 9, train loss: 0.00038, val loss:
0.00077, in 0.013s
[53/50000] 1 tree, 31 leaves, max depth = 12, train loss: 0.00038, val loss:
0.00077, in 0.012s
[54/50000] 1 tree, 31 leaves, max depth = 9, train loss: 0.00037, val loss:
0.00077, in 0.009s
[55/50000] 1 tree, 31 leaves, max depth = 12, train loss: 0.00037, val loss:
0.00076, in 0.011s
[56/50000] 1 tree, 31 leaves, max depth = 10, train loss: 0.00036, val loss:
0.00076, in 0.014s
[57/50000] 1 tree, 31 leaves, max depth = 15, train loss: 0.00036, val loss:
0.00076, in 0.012s
[58/50000] 1 tree, 31 leaves, max depth = 10, train loss: 0.00036, val loss:
0.00075, in 0.010s
[59/50000] 1 tree, 31 leaves, max depth = 16, train loss: 0.00035, val loss:
0.00075, in 0.010s
[60/50000] 1 tree, 31 leaves, max depth = 12, train loss: 0.00035, val loss:
0.00076, in 0.014s
[61/50000] 1 tree, 31 leaves, max depth = 10, train loss: 0.00035, val loss:
0.00075, in 0.011s
[62/50000] 1 tree, 31 leaves, max depth = 12, train loss: 0.00034, val loss:
0.00075, in 0.010s
[63/50000] 1 tree, 31 leaves, max depth = 12, train loss: 0.00034, val loss:
0.00075, in 0.009s
[64/50000] 1 tree, 31 leaves, max depth = 11, train loss: 0.00034, val loss:
0.00075, in 0.018s
[65/50000] 1 tree, 31 leaves, max depth = 10, train loss: 0.00033, val loss:
0.00075, in 0.009s
[66/50000] 1 tree, 31 leaves, max depth = 9, train loss: 0.00033, val loss:
0.00075, in 0.010s
[67/50000] 1 tree, 31 leaves, max depth = 11, train loss: 0.00033, val loss:
0.00074, in 0.014s
[68/50000] 1 tree, 31 leaves, max depth = 13, train loss: 0.00033, val loss:
0.00074, in 0.011s
[69/50000] 1 tree, 31 leaves, max depth = 15, train loss: 0.00032, val loss:
```

```
0.00074, in 0.012s
[70/50000] 1 tree, 31 leaves, max depth = 8, train loss: 0.00032, val loss:
0.00074, in 0.011s
[71/50000] 1 tree, 31 leaves, max depth = 13, train loss: 0.00032, val loss:
0.00074, in 0.010s
[72/50000] 1 tree, 31 leaves, max depth = 9, train loss: 0.00031, val loss:
0.00074, in 0.012s
[73/50000] 1 tree, 31 leaves, max depth = 15, train loss: 0.00031, val loss:
0.00074, in 0.020s
[74/50000] 1 tree, 31 leaves, max depth = 11, train loss: 0.00031, val loss:
0.00074, in 0.010s
[75/50000] 1 tree, 31 leaves, max depth = 11, train loss: 0.00031, val loss:
0.00074, in 0.016s
[76/50000] 1 tree, 31 leaves, max depth = 11, train loss: 0.00030, val loss:
0.00074, in 0.013s
[77/50000] 1 tree, 31 leaves, max depth = 10, train loss: 0.00030, val loss:
0.00073, in 0.010s
[78/50000] 1 tree, 31 leaves, max depth = 14, train loss: 0.00030, val loss:
0.00073, in 0.008s
[79/50000] 1 tree, 31 leaves, max depth = 10, train loss: 0.00030, val loss:
0.00073, in 0.012s
[80/50000] 1 tree, 31 leaves, max depth = 14, train loss: 0.00029, val loss:
0.00073, in 0.009s
[81/50000] 1 tree, 31 leaves, max depth = 14, train loss: 0.00029, val loss:
0.00073, in 0.016s
[82/50000] 1 tree, 31 leaves, max depth = 16, train loss: 0.00029, val loss:
0.00073, in 0.018s
[83/50000] 1 tree, 31 leaves, max depth = 13, train loss: 0.00029, val loss:
0.00073, in 0.011s
[84/50000] 1 tree, 31 leaves, max depth = 10, train loss: 0.00029, val loss:
0.00073, in 0.010s
[85/50000] 1 tree, 31 leaves, max depth = 11, train loss: 0.00028, val loss:
0.00073, in 0.009s
[86/50000] 1 tree, 31 leaves, max depth = 11, train loss: 0.00028, val loss:
0.00073, in 0.008s
[87/50000] 1 tree, 31 leaves, max depth = 13, train loss: 0.00028, val loss:
0.00073, in 0.008s
[88/50000] 1 tree, 31 leaves, max depth = 14, train loss: 0.00028, val loss:
0.00073, in 0.013s
[89/50000] 1 tree, 31 leaves, max depth = 9, train loss: 0.00027, val loss:
0.00073, in 0.010s
[90/50000] 1 tree, 31 leaves, max depth = 10, train loss: 0.00027, val loss:
0.00073, in 0.010s
[91/50000] 1 tree, 31 leaves, max depth = 12, train loss: 0.00027, val loss:
0.00073, in 0.009s
[92/50000] 1 tree, 31 leaves, max depth = 8, train loss: 0.00027, val loss:
0.00073, in 0.011s
[93/50000] 1 tree, 31 leaves, max depth = 13, train loss: 0.00027, val loss:
```



```
0.00073, in 0.010s
[94/50000] 1 tree, 31 leaves, max depth = 15, train loss: 0.00026, val loss:
0.00073, in 0.012s
[95/50000] 1 tree, 31 leaves, max depth = 11, train loss: 0.00026, val loss:
0.00073, in 0.012s
Fit 95 trees in 1.279 s, (2945 total leaves)
Time spent computing histograms: 0.202s
Time spent finding best splits: 0.117s
Time spent applying splits: 0.121s
Time spent predicting: 0.009s
Estimadores para Train
Metr.    Valor          Descripción
R^2      0.9101464221217427 (Coeficiente de Determinación)
RMSE     0.0006182259662542193 (Raíz de error cuadrático medio)
MAE      0.01879705055225904 (Error absoluto medio)
MAX     0.18431448005700957 (Error Máximo)

Estimadores para Test
Metr.    Valor          Descripción
R^2      0.842183850570004 (Coeficiente de Determinación)
RMSE     0.001130855140274013 (Raíz de error cuadrático medio)
MAE      0.025811140466109116 (Error absoluto medio)
MAX     0.16397140370451635 (Error Máximo)
```



Como se aprecia de todas las iteraciones permitidas ha parado tras 95 por no estar ya mejorando el error en validación, que se había estancado en 0.00073, a pesar que el error en train ya había bajado a 0.00026.

Vemos de lanzar la búsqueda de los mejores hiperparametros usando este último modo con una búsqueda aleatoria de hiperparámetros, y mostrando la gráfica de entrenamiento.

```
[11]: from sklearn.model_selection import RandomizedSearchCV
random_grid = {'learning_rate': [0.0001, 0.001, 0.01, 0.1, 0.2, 0.25, 0.3, 0.5, 1],
               'max_iter': [200, 500, 1000, 5000],
               'max_leaf_nodes': [5, 10, 20, 30, 50, 100, 200],
               'early_stopping': [True],
               'validation_fraction': [0.1, 0.15, 0.2],
```

```
        'n_iter_no_change': [10, 15],
        'tol': [1e-07]
    }
modelo_random = RandomizedSearchCV(scoring="neg_mean_squared_error", estimator_=_
    modelo2,
                           param_distributions = random_grid, n_iter = 200,
                           cv = 3, verbose=1, random_state=123, n_jobs = -1)
modelo_random.fit(X_train_s, y_train)

## Mostramos los resultados
print("Estimadores para Train")
mostrar_metricas(y_train, modelo_random.best_estimator_.predict(X_train_s))
print("")
print("Estimadores para Test")
mostrar_metricas(y_test, modelo_random.best_estimator_.predict(X_test_s))

## Mostramos la gráfica del entrenamiento
def mostrar_entrenamiento(modelo):
    # Muestro gráfica de accuracy y losses
    plt.style.use("ggplot")
    fig,ax = plt.subplots()
    filas = len(modelo.validation_score_)
    ax.plot(np.arange(0, filas), modelo.train_score_, label="train_loss",_
            color="red")
    ax.plot(np.arange(0, filas), modelo.validation_score_, label="val_loss",_
            color="orange")
    # Pongo en un eje secundario la Precisión, porque la escala no está en los_
    # mismos rangos
    ax2=ax.twinx()
    ax2.plot(np.arange(0, filas), H.history["accuracy"], label="train_acc",_
              color="blue")
    ax2.plot(np.arange(0, filas), H.history["val_accuracy"], label="val_acc",_
              color="green")
    ax.set_title("Training/Validation Loss and Accuracy")
    ax.set_xlabel("Epoch #")
    ax.set_ylabel("Loss", color="red")
    #ax.set_xlim(0., np.max(H.history["loss"]))
    ax2.set_ylabel("Accuracy", color="blue")
    #ax2.set_xlim(0., 1.)
    ax.legend(loc="upper right")
    #ax2.legend(loc="lower right")
    mostrar_entrenamiento(modelo_random.best_estimator_)
```

Fitting 3 folds for each of 200 candidates, totalling 600 fits  
Binning 0.001 GB of training data: 0.015 s  
Binning 0.000 GB of validation data: 0.001 s  
Fitting gradient boosted rounds:

```
[1/200] 1 tree, 20 leaves, max depth = 7, train loss: 0.00295, val loss:  
0.00301, in 0.008s  
[2/200] 1 tree, 20 leaves, max depth = 7, train loss: 0.00256, val loss:  
0.00265, in 0.009s  
[3/200] 1 tree, 20 leaves, max depth = 7, train loss: 0.00224, val loss:  
0.00237, in 0.010s  
[4/200] 1 tree, 20 leaves, max depth = 7, train loss: 0.00197, val loss:  
0.00212, in 0.009s  
[5/200] 1 tree, 20 leaves, max depth = 6, train loss: 0.00176, val loss:  
0.00193, in 0.009s  
[6/200] 1 tree, 20 leaves, max depth = 8, train loss: 0.00158, val loss:  
0.00177, in 0.008s  
[7/200] 1 tree, 20 leaves, max depth = 7, train loss: 0.00144, val loss:  
0.00164, in 0.008s  
[8/200] 1 tree, 20 leaves, max depth = 7, train loss: 0.00131, val loss:  
0.00153, in 0.008s  
[9/200] 1 tree, 20 leaves, max depth = 8, train loss: 0.00121, val loss:  
0.00144, in 0.008s  
[10/200] 1 tree, 20 leaves, max depth = 7, train loss: 0.00113, val loss:  
0.00137, in 0.010s  
[11/200] 1 tree, 20 leaves, max depth = 8, train loss: 0.00105, val loss:  
0.00130, in 0.009s  
[12/200] 1 tree, 20 leaves, max depth = 7, train loss: 0.00100, val loss:  
0.00125, in 0.009s  
[13/200] 1 tree, 20 leaves, max depth = 7, train loss: 0.00094, val loss:  
0.00120, in 0.008s  
[14/200] 1 tree, 20 leaves, max depth = 7, train loss: 0.00090, val loss:  
0.00117, in 0.008s  
[15/200] 1 tree, 20 leaves, max depth = 6, train loss: 0.00086, val loss:  
0.00113, in 0.008s  
[16/200] 1 tree, 20 leaves, max depth = 8, train loss: 0.00083, val loss:  
0.00110, in 0.009s  
[17/200] 1 tree, 20 leaves, max depth = 7, train loss: 0.00080, val loss:  
0.00107, in 0.008s  
[18/200] 1 tree, 20 leaves, max depth = 8, train loss: 0.00077, val loss:  
0.00105, in 0.008s  
[19/200] 1 tree, 20 leaves, max depth = 8, train loss: 0.00075, val loss:  
0.00104, in 0.008s  
[20/200] 1 tree, 20 leaves, max depth = 6, train loss: 0.00073, val loss:  
0.00102, in 0.008s  
[21/200] 1 tree, 20 leaves, max depth = 8, train loss: 0.00071, val loss:  
0.00100, in 0.008s  
[22/200] 1 tree, 20 leaves, max depth = 8, train loss: 0.00069, val loss:  
0.00099, in 0.011s  
[23/200] 1 tree, 20 leaves, max depth = 9, train loss: 0.00068, val loss:  
0.00098, in 0.008s  
[24/200] 1 tree, 20 leaves, max depth = 6, train loss: 0.00066, val loss:  
0.00096, in 0.010s
```

```
[25/200] 1 tree, 20 leaves, max depth = 8, train loss: 0.00065, val loss:  
0.00095, in 0.008s  
[26/200] 1 tree, 20 leaves, max depth = 8, train loss: 0.00064, val loss:  
0.00094, in 0.010s  
[27/200] 1 tree, 20 leaves, max depth = 8, train loss: 0.00063, val loss:  
0.00093, in 0.007s  
[28/200] 1 tree, 20 leaves, max depth = 7, train loss: 0.00062, val loss:  
0.00092, in 0.007s  
[29/200] 1 tree, 20 leaves, max depth = 8, train loss: 0.00061, val loss:  
0.00091, in 0.007s  
[30/200] 1 tree, 20 leaves, max depth = 11, train loss: 0.00060, val loss:  
0.00090, in 0.007s  
[31/200] 1 tree, 20 leaves, max depth = 8, train loss: 0.00059, val loss:  
0.00089, in 0.011s  
[32/200] 1 tree, 20 leaves, max depth = 10, train loss: 0.00058, val loss:  
0.00089, in 0.009s  
[33/200] 1 tree, 20 leaves, max depth = 6, train loss: 0.00057, val loss:  
0.00088, in 0.009s  
[34/200] 1 tree, 20 leaves, max depth = 9, train loss: 0.00056, val loss:  
0.00088, in 0.011s  
[35/200] 1 tree, 20 leaves, max depth = 8, train loss: 0.00056, val loss:  
0.00087, in 0.009s  
[36/200] 1 tree, 20 leaves, max depth = 8, train loss: 0.00055, val loss:  
0.00087, in 0.009s  
[37/200] 1 tree, 20 leaves, max depth = 9, train loss: 0.00054, val loss:  
0.00086, in 0.012s  
[38/200] 1 tree, 20 leaves, max depth = 7, train loss: 0.00054, val loss:  
0.00085, in 0.008s  
[39/200] 1 tree, 20 leaves, max depth = 8, train loss: 0.00053, val loss:  
0.00085, in 0.008s  
[40/200] 1 tree, 20 leaves, max depth = 8, train loss: 0.00052, val loss:  
0.00085, in 0.007s  
[41/200] 1 tree, 20 leaves, max depth = 7, train loss: 0.00052, val loss:  
0.00084, in 0.011s  
[42/200] 1 tree, 20 leaves, max depth = 9, train loss: 0.00051, val loss:  
0.00084, in 0.008s  
[43/200] 1 tree, 20 leaves, max depth = 8, train loss: 0.00051, val loss:  
0.00084, in 0.008s  
[44/200] 1 tree, 20 leaves, max depth = 10, train loss: 0.00050, val loss:  
0.00083, in 0.007s  
[45/200] 1 tree, 20 leaves, max depth = 7, train loss: 0.00049, val loss:  
0.00083, in 0.011s  
[46/200] 1 tree, 20 leaves, max depth = 9, train loss: 0.00049, val loss:  
0.00082, in 0.008s  
[47/200] 1 tree, 20 leaves, max depth = 10, train loss: 0.00048, val loss:  
0.00082, in 0.009s  
[48/200] 1 tree, 20 leaves, max depth = 8, train loss: 0.00048, val loss:  
0.00082, in 0.009s
```

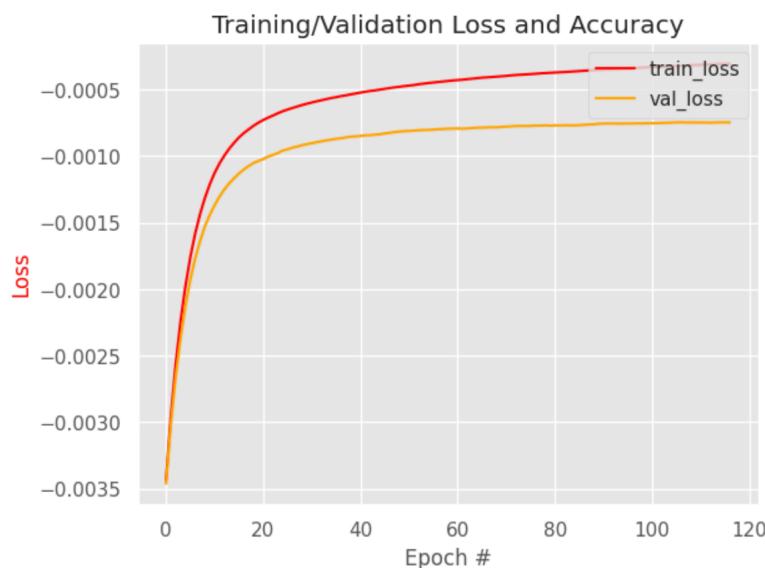
```
[49/200] 1 tree, 20 leaves, max depth = 8, train loss: 0.00047, val loss:  
0.00081, in 0.010s  
[50/200] 1 tree, 20 leaves, max depth = 8, train loss: 0.00047, val loss:  
0.00081, in 0.009s  
[51/200] 1 tree, 20 leaves, max depth = 12, train loss: 0.00047, val loss:  
0.00081, in 0.010s  
[52/200] 1 tree, 20 leaves, max depth = 7, train loss: 0.00046, val loss:  
0.00080, in 0.008s  
[53/200] 1 tree, 20 leaves, max depth = 7, train loss: 0.00045, val loss:  
0.00080, in 0.007s  
[54/200] 1 tree, 20 leaves, max depth = 12, train loss: 0.00045, val loss:  
0.00080, in 0.008s  
[55/200] 1 tree, 20 leaves, max depth = 7, train loss: 0.00045, val loss:  
0.00080, in 0.008s  
[56/200] 1 tree, 20 leaves, max depth = 8, train loss: 0.00044, val loss:  
0.00080, in 0.008s  
[57/200] 1 tree, 20 leaves, max depth = 11, train loss: 0.00044, val loss:  
0.00080, in 0.007s  
[58/200] 1 tree, 20 leaves, max depth = 9, train loss: 0.00043, val loss:  
0.00079, in 0.007s  
[59/200] 1 tree, 20 leaves, max depth = 8, train loss: 0.00043, val loss:  
0.00079, in 0.007s  
[60/200] 1 tree, 20 leaves, max depth = 8, train loss: 0.00043, val loss:  
0.00079, in 0.010s  
[61/200] 1 tree, 20 leaves, max depth = 8, train loss: 0.00042, val loss:  
0.00079, in 0.011s  
[62/200] 1 tree, 20 leaves, max depth = 7, train loss: 0.00042, val loss:  
0.00079, in 0.007s  
[63/200] 1 tree, 20 leaves, max depth = 7, train loss: 0.00042, val loss:  
0.00079, in 0.008s  
[64/200] 1 tree, 20 leaves, max depth = 9, train loss: 0.00041, val loss:  
0.00079, in 0.011s  
[65/200] 1 tree, 20 leaves, max depth = 8, train loss: 0.00041, val loss:  
0.00079, in 0.008s  
[66/200] 1 tree, 20 leaves, max depth = 12, train loss: 0.00041, val loss:  
0.00078, in 0.008s  
[67/200] 1 tree, 20 leaves, max depth = 11, train loss: 0.00040, val loss:  
0.00078, in 0.008s  
[68/200] 1 tree, 20 leaves, max depth = 10, train loss: 0.00040, val loss:  
0.00078, in 0.007s  
[69/200] 1 tree, 20 leaves, max depth = 8, train loss: 0.00040, val loss:  
0.00078, in 0.008s  
[70/200] 1 tree, 20 leaves, max depth = 7, train loss: 0.00040, val loss:  
0.00078, in 0.008s  
[71/200] 1 tree, 20 leaves, max depth = 8, train loss: 0.00039, val loss:  
0.00078, in 0.011s  
[72/200] 1 tree, 20 leaves, max depth = 9, train loss: 0.00039, val loss:  
0.00077, in 0.008s
```

```
[73/200] 1 tree, 20 leaves, max depth = 11, train loss: 0.00039, val loss:  
0.00077, in 0.008s  
[74/200] 1 tree, 20 leaves, max depth = 11, train loss: 0.00039, val loss:  
0.00077, in 0.007s  
[75/200] 1 tree, 20 leaves, max depth = 11, train loss: 0.00038, val loss:  
0.00077, in 0.012s  
[76/200] 1 tree, 20 leaves, max depth = 10, train loss: 0.00038, val loss:  
0.00077, in 0.008s  
[77/200] 1 tree, 20 leaves, max depth = 7, train loss: 0.00038, val loss:  
0.00077, in 0.009s  
[78/200] 1 tree, 20 leaves, max depth = 7, train loss: 0.00038, val loss:  
0.00077, in 0.009s  
[79/200] 1 tree, 20 leaves, max depth = 9, train loss: 0.00037, val loss:  
0.00077, in 0.008s  
[80/200] 1 tree, 20 leaves, max depth = 9, train loss: 0.00037, val loss:  
0.00077, in 0.008s  
[81/200] 1 tree, 20 leaves, max depth = 8, train loss: 0.00037, val loss:  
0.00077, in 0.008s  
[82/200] 1 tree, 20 leaves, max depth = 9, train loss: 0.00037, val loss:  
0.00077, in 0.008s  
[83/200] 1 tree, 20 leaves, max depth = 10, train loss: 0.00037, val loss:  
0.00077, in 0.007s  
[84/200] 1 tree, 20 leaves, max depth = 10, train loss: 0.00036, val loss:  
0.00077, in 0.011s  
[85/200] 1 tree, 20 leaves, max depth = 11, train loss: 0.00036, val loss:  
0.00077, in 0.009s  
[86/200] 1 tree, 20 leaves, max depth = 8, train loss: 0.00036, val loss:  
0.00076, in 0.007s  
[87/200] 1 tree, 20 leaves, max depth = 7, train loss: 0.00036, val loss:  
0.00076, in 0.008s  
[88/200] 1 tree, 20 leaves, max depth = 8, train loss: 0.00035, val loss:  
0.00076, in 0.012s  
[89/200] 1 tree, 20 leaves, max depth = 14, train loss: 0.00035, val loss:  
0.00076, in 0.008s  
[90/200] 1 tree, 20 leaves, max depth = 8, train loss: 0.00035, val loss:  
0.00075, in 0.008s  
[91/200] 1 tree, 20 leaves, max depth = 7, train loss: 0.00035, val loss:  
0.00075, in 0.008s  
[92/200] 1 tree, 20 leaves, max depth = 6, train loss: 0.00035, val loss:  
0.00075, in 0.009s  
[93/200] 1 tree, 20 leaves, max depth = 12, train loss: 0.00034, val loss:  
0.00076, in 0.008s  
[94/200] 1 tree, 20 leaves, max depth = 8, train loss: 0.00034, val loss:  
0.00075, in 0.008s  
[95/200] 1 tree, 20 leaves, max depth = 10, train loss: 0.00034, val loss:  
0.00075, in 0.009s  
[96/200] 1 tree, 20 leaves, max depth = 9, train loss: 0.00034, val loss:  
0.00075, in 0.008s
```

```
[97/200] 1 tree, 20 leaves, max depth = 10, train loss: 0.00034, val loss:  
0.00075, in 0.008s  
[98/200] 1 tree, 20 leaves, max depth = 7, train loss: 0.00033, val loss:  
0.00075, in 0.010s  
[99/200] 1 tree, 20 leaves, max depth = 8, train loss: 0.00033, val loss:  
0.00075, in 0.007s  
[100/200] 1 tree, 20 leaves, max depth = 12, train loss: 0.00033, val loss:  
0.00075, in 0.021s  
[101/200] 1 tree, 20 leaves, max depth = 8, train loss: 0.00033, val loss:  
0.00075, in 0.009s  
[102/200] 1 tree, 20 leaves, max depth = 9, train loss: 0.00033, val loss:  
0.00075, in 0.012s  
[103/200] 1 tree, 20 leaves, max depth = 11, train loss: 0.00033, val loss:  
0.00075, in 0.011s  
[104/200] 1 tree, 20 leaves, max depth = 10, train loss: 0.00032, val loss:  
0.00075, in 0.014s  
[105/200] 1 tree, 20 leaves, max depth = 12, train loss: 0.00032, val loss:  
0.00075, in 0.008s  
[106/200] 1 tree, 20 leaves, max depth = 10, train loss: 0.00032, val loss:  
0.00075, in 0.007s  
[107/200] 1 tree, 20 leaves, max depth = 12, train loss: 0.00032, val loss:  
0.00075, in 0.006s  
[108/200] 1 tree, 20 leaves, max depth = 9, train loss: 0.00032, val loss:  
0.00075, in 0.010s  
[109/200] 1 tree, 20 leaves, max depth = 10, train loss: 0.00031, val loss:  
0.00075, in 0.009s  
[110/200] 1 tree, 20 leaves, max depth = 11, train loss: 0.00031, val loss:  
0.00075, in 0.009s  
[111/200] 1 tree, 20 leaves, max depth = 10, train loss: 0.00031, val loss:  
0.00075, in 0.007s  
[112/200] 1 tree, 20 leaves, max depth = 8, train loss: 0.00031, val loss:  
0.00075, in 0.011s  
[113/200] 1 tree, 20 leaves, max depth = 11, train loss: 0.00031, val loss:  
0.00075, in 0.009s  
[114/200] 1 tree, 20 leaves, max depth = 11, train loss: 0.00031, val loss:  
0.00075, in 0.008s  
[115/200] 1 tree, 20 leaves, max depth = 10, train loss: 0.00031, val loss:  
0.00075, in 0.008s  
[116/200] 1 tree, 20 leaves, max depth = 8, train loss: 0.00031, val loss:  
0.00075, in 0.007s  
Fit 116 trees in 1.206 s, (2320 total leaves)  
Time spent computing histograms: 0.126s  
Time spent finding best splits: 0.110s  
Time spent applying splits: 0.088s  
Time spent predicting: 0.010s  
Estimadores para Train  
Metr.    Valor          Descripción  
R^2      0.8920809102796025  (Coeficiente de Determinación)
```

RMSE	0.0007425233930035075	(Raíz de error cuadrático medio)
MAE	0.020471776688240143	(Error absoluto medio)
MAX	0.22465169947440788	(Error Máximo)

Estimadores para Test		
Metr.	Valor	Descripción
R <sup>2</sup>	0.8378583629080998	(Coeficiente de Determinación)
RMSE	0.00116185006680291	(Raíz de error cuadrático medio)
MAE	0.025875958889140085	(Error absoluto medio)
MAX	0.18115629775007847	(Error Máximo)



```
[12]: modelo_random.best_params_
```

```
[12]: {'validation_fraction': 0.15,
      'tol': 1e-07,
      'n_iter_no_change': 10,
      'max_leaf_nodes': 20,
      'max_iter': 200,
      'learning_rate': 0.1,
      'early_stopping': True}
```



## 1.5. KNeighborsRegressor

Enlace al cuaderno de trabajo en “Jupyter Notebook” para poder probar y generar los datos de preparación del dataset.

Se puede consultar el fichero en la siguiente URL: [KNeighborsRegressor.ipynb](#)

Se adjunta el contenido del cuaderno como anexo.



# KNeighborsRegressor

April 2, 2023

## 0.1 #Introducción

TFM: Aplicación de ciencia de datos en el sector de producción animal para la predicción y explicación de óptimos en ganado porcino.

*Titulo:* KNeighborsRegressor

*Autor:* Jose Eduardo Cámará Gómez

---

## 0.2 Importar paquetes

```
[1]: # Importación de paquetes
import numpy as np
import pandas as pd
import seaborn as sns

import matplotlib.pyplot as plt
import matplotlib.mlab as mlab
import matplotlib
from matplotlib.pyplot import figure

from sklearn.preprocessing import OneHotEncoder
from sklearn.datasets import make_regression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import RobustScaler

from sklearn.neighbors import KNeighborsRegressor

sns.set(style="darkgrid")

[2]: from google.colab import files
# Cargamos el fichero del dataset gmd_02.csv
uploaded = files.upload()
# Leemos el fichero csv con los datos
df = pd.read_csv('gmd_02.csv', sep=';')
# Revisar la raza si se agrupan las razas con menos ocurrencias
agrupar_razas = {93 : 93, 85 : 93, 90 : 93, 95 : 93, 94 : 93, 82 : 93, 80 : 80, 96 : 80, 88 : 88, 0 : 0, 23 : 0,
```

```

84 : 0, 66 : 0, 18 : 0, 68 : 88, 7 : 7, 89 : 7, 65 : 7, 15 : 15, 97 : 7, 69 : 69, 81 : 81}
df.replace({'ct_raza' : agrupar_razas}, inplace=True)
df["bajas"] = df["NumBajas"] / (df["NumAnimales"] * df["DiasMedios"])
# Convertimos los tipos
df["ct_integra"] = df["ct_integra"].astype("category")
#df["ct_tipo"] = df["ct_tipo"].astype("category")
df["ct_raza"] = df["ct_raza"].astype("category")
df["ct_fase"] = df["ct_fase"].astype("category")
df['EntradaInicial']= pd.to_datetime(df['EntradaInicial'])
df['EntradaFinal']= pd.to_datetime(df['EntradaFinal'])
df["na_rega"] = df["na_rega"].astype("category")
df["NumBajas"] = df["NumBajas"].astype("int64")
df["gr_codpos"] = df["gr_codpos"].astype("category")
df["gr_poblacion"] = df["gr_poblacion"].astype("category")
df["na_nombre2"] = df["na_nombre2"].astype("category")

# Funcion para convertir en One Hot Encoding
def encode_and_bind(original_dataframe, feature_to_encode):
    dummies = pd.get_dummies(original_dataframe[[feature_to_encode]])
    res = pd.concat([original_dataframe, dummies], axis=1)
    res = res.drop([feature_to_encode], axis=1)
    return(res)

# Cargamos las variables objetivo y las usadas (15 variables seleccionadas, una de ellas categórica con 8 valores).
y = df['GMD']
x0 = df[['ct_integra', 'ct_tipo', 'ct_raza', 'IncPeso', 'NumAnimales', 'na_rega', 'PesoEntMedio', 'PesoRecMedio', 'bajas', 'GPS_Longitud', 'GPS_Latitud', 'semanaEntrada', 'añoEntrada', 'PorcHembras', 'PiensoCerdaDia']]
features_to_encode = ['ct_raza'] # , 'na_rega']
x1 = x0.copy()
x1.drop(['ct_integra', 'na_rega'], inplace=True, axis=1)
for feature in features_to_encode:
    x1 = encode_and_bind(x1, feature)

<IPython.core.display.HTML object>
Saving gmd_02.csv to gmd_02.csv

[3]: X_train, X_test, y_train, y_test = train_test_split(x1, y, test_size = 0.2, random_state = 123)
## Vemos de escalar las variables para que no se vean influenciadas por los outliers.
scaler = RobustScaler()
scaler.fit(X_train)
X_train_s = scaler.transform(X_train)
X_test_s = scaler.transform(X_test)

```

## 1 Lanzar modelo por defecto

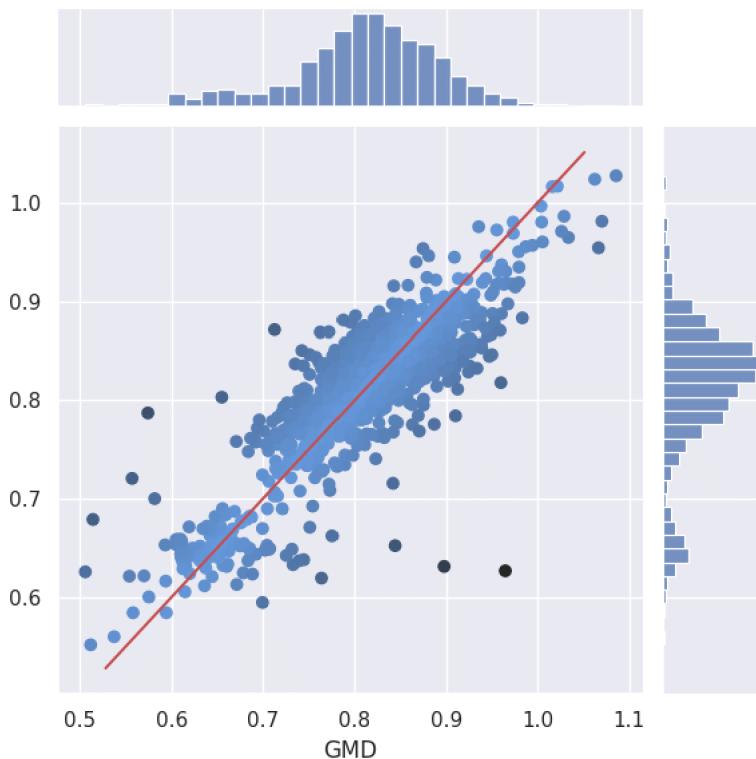
```
[5]: # Lanzamos el modelo con los parámetros por defecto
modelo = KNeighborsRegressor (
    n_neighbors=5,
    weights='uniform',
    algorithm='auto',
    leaf_size=30,
    p=2,
    metric='minkowski',
    metric_params=None,
    n_jobs=None
)
modelo.fit(X_train_s, y_train)

[5]: KNeighborsRegressor()

[6]: # Función para Graficar diferencias entre valor predicho y real en datos de test del modelo pasado
def graficoDiferencias(modelo, X_test_s, y_test):
    y_pred = modelo.predict(X_test_s)
    diferencia = abs(y_pred - y_test)
    g = sns.jointplot(x=y_test, y=y_pred)
    # Draw a line of x=y
    x0, x1 = g.ax_joint.get_xlim()
    y0, y1 = g.ax_joint.get_ylim()
    lims = [max(x0, y0), min(x1, y1)]
    g.ax_joint.plot(lims, lims, '-r')
    g.ax_joint.scatter(x=y_test, y=y_pred, c=diferencia.values, cmap=sns.
    dark_palette("#69d", reverse=True, as_cmap=True))
    plt.show()

# Graficar las diferencias
print('Score R2:', modelo.score(X_test_s, y_test))
graficoDiferencias(modelo, X_test_s, y_test)
```

Score R2: 0.7459138720387797

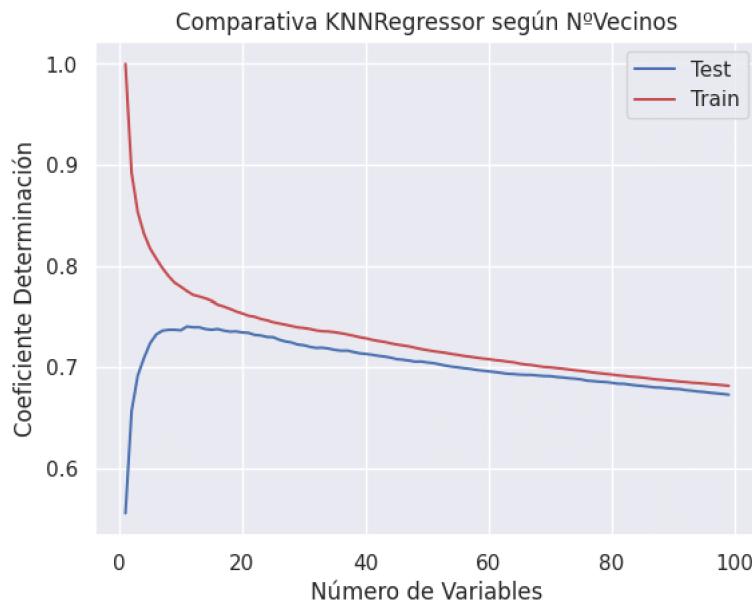


Vemos como se comporta con menos variables, por ejemplo vamos a estudiar que pasa para las 7 más relevantes según RandomForest.

```
[10]: x2 =  
    x1[['PesoRecMedio','ct_raza_69','ct_tipo','PesoEntMedio','IncPeso','PorcHembras','semanaEnt'  
    .copy()  
X_train, X_test, y_train, y_test = train_test_split(x2, y, test_size = 0.2,  
    random_state = 123)  
## Vemos de escalar las variables para que no se vean influenciadas por los  
    outliers.  
scaler = RobustScaler()  
scaler.fit(X_train)  
X_train_s = scaler.transform(X_train)
```

```
X_test_s = scaler.transform(X_test)
```

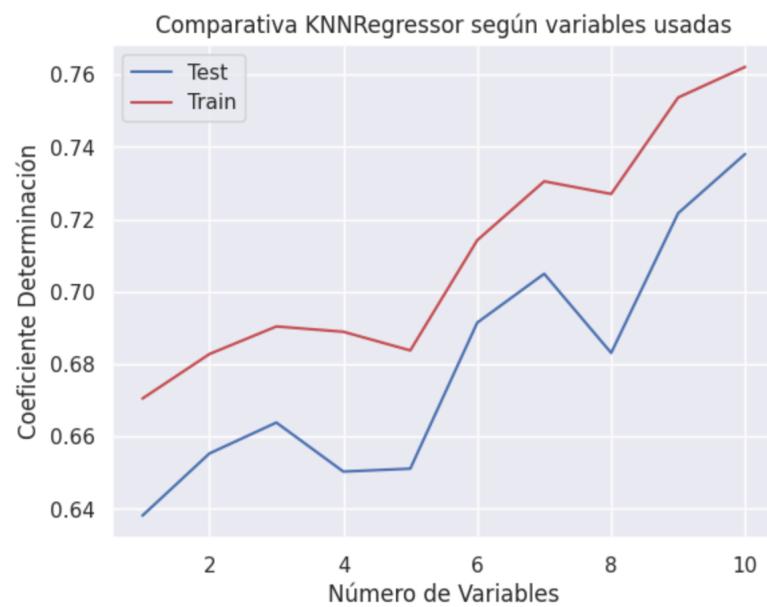
```
[29]: # Lanzamos el modelo con los parámetros por defecto, para los 100 primeros valores de vecinos más cercanos
from sklearn.metrics import r2_score
r2_train = list()
r2_test = list()
for i in range(1,100):
    modelo = KNeighborsRegressor (
        n_neighbors=1,
        weights='uniform',
        algorithm='auto',
        leaf_size=30,
        p=2,
        metric='minkowski',
        metric_params=None,
        n_jobs=None
    )
    modelo.fit(X_train_s, y_train)
    # Graficar las diferencias
    r2_train.append(r2_score(y_train, modelo.predict(X_train_s)))
    r2_test.append(r2_score(y_test, modelo.predict(X_test_s)))
    # print('Score R2 para n_neighbors=',i, 'Train:', r2_score(y_train, modelo.predict(X_train_s)), 'Test:', r2_score(y_test, modelo.predict(X_test_s)))
sns.lineplot(x=range(1,100), y=r2_test, label="Test")
sns.lineplot(x=range(1,100), y=r2_train, c='r', label="Train")
plt.legend()
plt.xlabel("Número de Variables")
plt.ylabel("Coeficiente Determinación")
plt.title("Comparativa KNNRegressor según NºVecinos")
plt.show()
```



Para esas variables el valor más alto es el de n=16, que obtiene un R<sup>2</sup>=0.7050. Veo ahora como se comporta si modificamos el número de variables desde 1 a 21 para los datos y orden que tenemos según RandomForest.

```
[28]: top10_variables = ['PesoRecMedio','ct_raza_69','ct_tipo','PesoEntMedio','IncPeso','PorcHembras','semanaEntrada','edad','ct_raza_1','ct_raza_2','ct_raza_3','ct_raza_4','ct_raza_5','ct_raza_6','ct_raza_7','ct_raza_8','ct_raza_9','ct_raza_10','ct_raza_11']
r2_train_v = list()
r2_test_v = list()
for n_var in range(1,11):
    x3 = x1[top10_variables[:n_var]].copy()
    X_train, X_test, y_train, y_test = train_test_split(x3, y, test_size = 0.2, random_state = 123)
    ## Vemos de escalar las variables para que no se vean influenciadas por los outliers.
    scaler = RobustScaler()
    scaler.fit(X_train)
    X_train_s = scaler.transform(X_train)
    X_test_s = scaler.transform(X_test)
    # Entremos el modelo con esas variables
```

```
modelo = KNeighborsRegressor (n_neighbors=16, weights='uniform',  
                             algorithm='auto', leaf_size=30, p=2, metric='minkowski', metric_params=None,  
                             n_jobs=None)  
modelo.fit(X_train_s, y_train)  
r2_test_v.append(r2_score(y_test, modelo.predict(X_test_s)))  
r2_train_v.append(r2_score(y_train, modelo.predict(X_train_s)))  
print('Score R2 para variables=',n_var, 'Train:', r2_score(y_train, modelo.  
predict(X_train_s)), '\t', 'Test:', r2_score(y_test, modelo.predict(X_test_s)))  
  
# Graficamos los datos  
sns.lineplot(x=range(1,11), y=r2_test_v, label="Test")  
sns.lineplot(x=range(1,11), y=r2_train_v, c='r', label="Train")  
plt.legend()  
plt.xlabel("Número de Variables")  
plt.ylabel("Coeficiente Determinación")  
plt.title("Comparativa KNNRegressor según variables usadas")  
plt.show()  
  
Score R2 para variables= 1 Train: 0.6705166094575745 Test:  
0.6381669196044853  
Score R2 para variables= 2 Train: 0.6828041807738229 Test:  
0.6553354693811115  
Score R2 para variables= 3 Train: 0.690438873499253 Test:  
0.6638840821849186  
Score R2 para variables= 4 Train: 0.6889679181955092 Test:  
0.6503348539634992  
Score R2 para variables= 5 Train: 0.683807580504175 Test:  
0.6511477204934475  
Score R2 para variables= 6 Train: 0.7142649850034557 Test:  
0.6914857532835079  
Score R2 para variables= 7 Train: 0.7305353480639281 Test: 0.704989781208365  
Score R2 para variables= 8 Train: 0.727049674919438 Test:  
0.6831335621543003  
Score R2 para variables= 9 Train: 0.7536903649786754 Test:  
0.7217233354036477  
Score R2 para variables= 10 Train: 0.7621505380817271 Test:  
0.7380769691403543
```



## Anexos I

*!!!Por implementar!!! (quitar)*

Los anexos también contienen información adicional que se considera relevante para justificar las conclusiones del trabajo, pero, por lo general, el autor de contenido del anexo es distinto al autor del trabajo. Suele ser un documento independiente del trabajo. Pueden ser tablas de datos, imágenes, etc. Es necesario incluir las referencias de los documentos de donde procedan.