ESCUELA POLITÉCNICA SUPERIOR DE CÓRDOBA
Universidad de Córdoba

EPSC

MEMORIA TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

"Asistente para la configuración de Pipelines de Aprendizaje Supervisado mediante LLMs"

"Assistant for configuring Supervised Learning Pipelines using LLMs"

Autor: Isaac Manuel Cejudo Alfaro

Directores: Dr. José Raúl Romero Salguero

Dr. Carlos García Martínez

UNIVERSIDAD D CÓRDOBA

Declaro que el presente Trabajo Fin de Grado es original y no ha sido presentado con anterioridad para la obtención de ningún otro título. Asimismo, se han citado debidamente todas las fuentes utilizadas conforme a lo establecido en el articulo 18 del Reglamento 24/2024 de la EPSC.

CEJUDO ALFARO ISAAC MANUEL - 31030157Y

Firmado digitalmente por CEJUDO ALFARO ISAAC MANUEL - 31030157Y
Fecha: 2025.09.10 20:26:30 +02'00'

Resumen

Este Trabajo Fin de Grado presenta el desarrollo de un **asistente para la configuración de *pipelines* de aprendizaje supervisado**, estos se podrían definir como una secuencia bien definida de operaciones de preprocesamiento de datos seguido por la selección y entrenamiento de un modelo supervisado.

Este asistente se basa en modelos de lenguaje de gran tamaño (LLMs). Lo que se busca principalmente es **simplificar la creación de flujos de trabajo en aprendizaje automático**, para que un usuario pueda cargar un conjunto de datos, analizarlo, implementar técnicas de preprocesamiento, entrenar un modelo y evaluar su rendimiento mediante instrucciones en lenguaje natural.

La propuesta se basa en una **arquitectura modular**, organizada alrededor de un agente coordinador que supervisa distintas etapas del proceso. Para cada una de estas, se implementan agentes especializados — análisis de datos, preprocesamiento de datos, selección de instancias y características, entrenamiento de modelos y evaluación — que funcionan con *tools* para facilitar la interacción con librerías de *machine learning* en *Python*. Como capa de interacción, se ha desarrollado una interfaz: un ***dashboard*** usando el framework *Panel.*

Los experimentos realizados sobre conjuntos de datos de prueba muestran que el sistema puede crear *pipelines* completos de forma sencilla mediante interacciones en lenguaje natural. En esta línea, el asistente supone un avance en la **democratización del aprendizaje automático** porque disminuye la barrera de entrada técnica y agiliza la construcción de modelos.

Durante el desarrollo se comprobó que la interacción en lenguaje natural resultaba especialmente útil en la fase de preprocesamiento, ya que simplificaba las tareas que normalmente requieren más tiempo y conocimientos técnicos.

Palabras clave: Aprendizaje supervisado, LLMs, AutoML, agentes inteligentes, *pipelines* de datos.

Abstract

In this Final Degree Project, a large language model (LLM) based helper **for configuring supervised learning pipelines** is developed. This could be defined as a well-defined secuence of data preprocessing operations followed by the selection and training of a supervised model.

The primary goal is **to make it easier to create machine learning workflows** so that a user may load a dataset, analyze it, apply preprocessing techniques, train a model and evaluate its performance using natural language instructions.

The plan is based on a **modular architecture** that is structured around a coordinator agent that coordinates the different stages of the process. Specialized agents —data analysis, instance and feature selection, model training and evaluation— as well as tools that allow interaction with *Python* machine learning libraries are created for each of them. An interface —a dashboard built using the *Panel* framework — has been created as the interaction layer.

Experiments show that the system can easily provide complete pipelines using natural language interactions. Since the assistants speed up the process of building models and reduce technical obstacles to entry, they can be seen as a step towards **the democratization of machine learning.**

One observation I was able to confirm during development was the natural language interaction proved especially useful in the preprocessing phase, since it simplified tasks that normally require more time and technical knowledge.

Keywords: Supervised learning, LLMs, AutoML, intelligent agents, data pipelines

Asistente para la configuración de Pipelines de Aprendizaje Supervisado mediante LLMs

Autor: Isaac Manuel Cejudo Alfaro
Directores: José Raúl Romero Salguero y Carlos García Martínez.

Este trabajo de fin de grado es parte de la Cátedra Internacional ENIA AgriculturaUniversidad de Córdoba (TSI-100921-2023-3), financiada por la Secretaría de Estado de Digitalización e Inteligencia Artificial y por la Unión Europea-Next Generation EU. Plan de Recuperación, Transformación y Resiliencia.

Tabla de Contenidos

Índice de Figuras

Capítulo 1. Introducción y Contexto

1.1. Justificación académica y práctica

El incremento acelerado en la generación de datos, tanto a nivel empresarial como personal, ha promovido el avance y la implementación de **técnicas de aprendizaje automático** (ML, *machine learning* [1]) para obtener conocimiento y tomar decisiones basadas en estos datos. No obstante, la configuración y entrenamiento de métodos de aprendizaje supervisado continúan siendo tareas complejas, que requieren no solo un conocimiento profundo en estadística y programación, sino también experiencia en la selección y preprocesamiento de los datos.

El término *aprendizaje automático* se refiere al "*área de la ciencia computacional que se centra en el análisis y la interpretación de patrones y estructuras de datos que hacen posible el aprendizaje, el razonamiento y la toma de decisiones sin interacción humana*" [2]. Esta definición muestra la extensión del campo, que combina métodos algorítmicos, estadísticos y de programación para crear sistemas que puedan aprender a partir de los datos.

Uno de los elementos fundamentales en ML es el **pipeline de aprendizaje supervisado**, este se podría definir como una secuencia bien definida de operaciones de preprocesamiento de datos —normalización [3], limpieza [4] y balanceo [5]— seguido por la selección y entrenamiento de un modelo supervisado. Por ejemplo, un pipeline puede incluir técnicas de preprocesamiento y finalizar con un clasificador basado en *Random Forest* [6]. La ventaja de definir e implementar pipelines de aprendizaje supervisado, frente a aplicar manualmente operaciones de limpieza de datos y otras técnicas, es que la secuencia de operaciones de preprocesamiento queda definida en el mismo y su aplicación sobre los datos originales es por tanto fácilmente reproducible.

El reto principal radica en convertir los datos brutos en conocimiento útil y novedoso. Este procedimiento incluye desde el análisis inicial de los datos hasta la implementación de técnicas específicas de limpieza, normalización y balanceo, todo ello antes de seleccionar y afinar el modelo de ML más apropiado para el problema a solucionar. Este flujo de trabajo requiere que las decisiones estén bien fundamentadas y que haya

experiencia práctica, lo cual restringe su accesibilidad para quienes no tienen una formación avanzada en ciencia de datos.

La necesidad de tener expertos en ciencia de datos limita, por tanto, la entrada de otros profesionales y organizaciones, que podrían aprovechar la capacidad del aprendizaje automático, particularmente en situaciones donde la experiencia técnica y en técnicas computacionales es limitada. Esta circunstancia supone un reto tanto académico como social: ¿Cómo se puede asegurar la calidad y rigurosidad en los resultados a la vez que se propicia el acceso a herramientas de ML supervisado para usuarios con menos conocimiento técnico?

En los últimos años, la irrupción de los modelos de lenguaje de gran tamaño (LLMs, *Large Language Models* [7]) ha cambiado profundamente la forma en que interactuamos con los computadores. Gracias a ellos, podemos realizar consultas, buscar información en la web, generar imágenes, audio y video, e incluso automatizar tareas complejas y todo ello mediante instrucciones en lenguaje natural. Ejemplos destacados de estos modelos incluyen *ChatGPT* [8], *Gemini* [9] y *Copilot* [10].

Un *modelo de lenguaje de gran tamaño* es un sistema diseñado para procesar y generar texto de manera avanzada, pudiendo transformar entradas textuales en diferentes tipos de contenido, como texto estructurado, código o incluso imágenes y audio. Estos modelos están entrenados sobre grandes volúmenes de datos y permiten la automatización de tareas relacionadas con el procesamiento del lenguaje natural (PLN) [7].

Conscientes de estos avances, este Trabajo de Fin de Grado (TFG) sugiere **la creación de una aplicación que permita la configuración de *pipelines* de aprendizaje supervisado a través de interacciones en lenguaje natural**. La aplicación utilizará modelos de lenguaje de gran tamaño para interpretar y convertir instrucciones de lenguaje natural en operaciones técnicas, tales como la exploración de datos, la aplicación de técnicas de preprocesamiento y la selección y entrenamiento de modelos.

Esta solución busca no solo simplificar el proceso de generación de *pipelines*, sino también universalizar el acceso al aprendizaje automático, facilitando a expertos de

diferentes campos la configuración y uso de herramientas de ML sin requerir habilidades avanzadas en programación o análisis de datos. La comunicación a través del lenguaje natural se transforma en el instrumento esencial para que los usuarios puedan manifestar sus requerimientos y obtener respuestas inmediatas, ajustadas al contexto del problema que buscan solucionar.

1.2. Relación con la Ingeniería Informática

Este proyecto nace justo en ese punto donde varias ramas del Grado en Ingeniería Informática se cruzan y se nutren entre sí.

- **Inteligencia Artificial** [11]: Aquí entra en juego el aprendizaje supervisado, con sus algoritmos que "aprenden" de ejemplos y con las métricas que permiten medir su desempeño, ajustando el modelo siempre que sea necesario para alcanzar la precisión deseada.

- **Ingeniería del Software** [12]: El desarrollo se apoya en un diseño modular basado en agentes y *tools*, pensado para crecer sin romperse, con una arquitectura flexible y se valida mediante pruebas unitarias e integradas que actúan como una red de seguridad frente a errores invisibles.

- **Interacción Persona-Ordenador**: El sistema debe sentirse accesible. Por esto, se ofrece un *dashboard* interactivo con *Panel* [13], que hace que la experiencia sea mucho más intuitiva, donde el sistema se comunicará con el usuario en lenguaje natural.

En conjunto, este proyecto no es solo una lista de apartados técnicos: es la síntesis real de las competencias adquiridas durante la titulación. Y lo más interesante es que no se limita a lo académico, sino que mira de frente a problemas actuales de la práctica profesional, como lograr que la inteligencia artificial sea accesible de verdad o integrar nuevas tecnologías —como los LLMs— en sistemas de software que tienen que funcionar en el mundo real.

1.3. Estructura de la memoria

La memoria se organiza siguiendo la lógica habitual de los proyectos de ingeniería, aunque cada capítulo tiene su propio propósito.

- El **Capítulo 2** abre el camino con los objetivos del proyecto, tanto los generales que marcan la dirección como los específicos que concretan el rumbo.

- En el **Capítulo 3** se define el problema: primero desde su cara más tangible, la que existe en la realidad, y después desde su dimensión técnica. También se describe cómo debería comportarse el sistema una vez desarrollado, a modo de hoja de ruta.

- El **Capítulo 4** invita a mirar atrás y alrededor: revisa los antecedentes y el estado del arte, analiza enfoques que ya se han explorado y presenta herramientas afines que ayudan a situar el proyecto en su contexto.

- Los **Capítulos 5 y 6** son más prácticos: aquí se detallan las restricciones que condicionan el trabajo y los recursos con los que se cuenta.

- El **Capítulo 7** se centra en especificar los requisitos, para conocer qué debe ser capaz de realizar el sistema y definir el modelado de casos de uso, para conocer las funcionalidades del sistema desde un punto de vista más detallado.

- El **Capítulo 8** da un paso más hacia dentro y se centra en el análisis del sistema, descomponiéndolo en módulos funcionales para entender mejor sus piezas.

- En el **Capítulo 9** se pasa al terreno del diseño: se plantea la arquitectura y los componentes principales, cuidando tanto la visión global como los detalles que sostienen la estructura.

- El **Capítulo 10** recoge la implementación, es decir, la parte donde las ideas empiezan a tomar forma concreta en código y soluciones reales.

- El **Capítulo 11** documenta las pruebas realizadas, mientras que el **Capítulo 12** se centra en la experimentación y los resultados, mostrando lo que funciona, lo que sorprende y lo que invita a mejorar.

- El **Capítulo 13** cierra con las conclusiones y traza posibles líneas de trabajo futuro, porque todo proyecto deja siempre puertas abiertas para seguir explorando.

Finalmente, se incorpora la bibliografía, junto con varios apéndices o anexos que sirven como apoyo técnico y manuales de referencia, pensados para quien quiera profundizar más allá del cuerpo principal del documento.

Capítulo 2. Objetivos

2.1. Objetivo general

El objetivo principal del Trabajo de Fin de Grado es desarrollar una aplicación que asista a los usuarios en la configuración, entrenamiento y evaluación de pipelines de aprendizaje supervisado mediante interacciones en lenguaje natural.

2.2. Objetivos específicos

Los objetivos específicos que permiten alcanzar este objetivo son:

1. **Integrar Procesamiento del Lenguaje Natural basado en *LLMs*:** Implementar un módulo que transforme las instrucciones del usuario en lenguaje natural en acciones técnicas relacionadas con los datos y la configuración del pipeline.

2. **Facilitar la carga y exploración de datos:** Permitir la importación y visualización de conjuntos de datos, proporcionando mecanismos que

simplifiquen el análisis exploratorio de datos y la detección de problemas en los datos como valores atípicos y datos faltantes.

3. **Automatizar el preprocesamiento de datos**: Desarrollar módulos que realicen tareas como normalización, el manejo de valores nulos y el balanceo de clases, ofreciendo alternativas de personalización de acuerdo con las características del conjunto de datos.

4. **Configurar y entrenar pipelines de aprendizaje supervisado**: Integrar librerías de ML para la selección, configuración y entrenamiento de modelos y automatizar la generación de pipelines que integren todo el proceso.

5. **Implementar un sistema de evaluación de modelos**: Desarrollar un módulo para evaluar el rendimiento de los modelos entrenados, permitiendo la comparación entre distintas configuraciones de pipelines.

6. **Diseñar una interfaz intuitiva**: Crear un entorno de usuario que permita la entrada de órdenes mediante lenguaje natural y permita la visualización clara de las distintas etapas del pipeline (exploración, preprocesamiento, entrenamiento y evaluación).

Capítulo 3. Definición del problema

3.1. Problema real

Cuando se trata de problemas de regresión o problemas de clasificación de elementos en sectores tan variados como la salud, las finanzas, la industria, el comercio electrónico o incluso el transporte, el **aprendizaje automático supervisado** se ha vuelto casi indispensable. Y es que, por lo general, tras cada sugerencia de producto, diagnóstico asistido por inteligencia artificial o sistema que prevé fallo en una máquina, existe un modelo que ha sido capacitado con este enfoque.

Un problema de **regresión** tiene como objetivo "*predecir un valor continuo a partir de los datos de entrada. Esto se utiliza cuando se desea predecir cifras como los ingresos, la altura, el peso o incluso la probabilidad de que algo ocurra*" [14].

Por el contrario, un problema de **clasificación** tiene como objetivo "*categorizar datos en diferentes clases o grupos. Por ejemplo, clasificar correos electrónicos como "spam" o "no spam" o predecir si un paciente padece una enfermedad específica según sus síntomas*" [14].

Como vimos en el Capítulo 1, un *pipeline* es una secuencia bien definida de operaciones sobre un conjunto de datos (ver Capítulo 1). Ahora bien, armar un *pipeline* completo no es tan sencillo como parece. Esto implica una cadena de decisiones técnicas: decidir qué variables son realmente relevantes, qué hacer con los huecos en los datos, cómo normalizar valores que no tienen el formato correcto, cómo equilibrar clases desproporcionadas, qué modelo probar primero y con qué métrica juzgarlo. Son pasos que, para quienes no tienen una formación sólida en estadística o programación, pueden sentirse como un laberinto.

El problema es que esa complejidad termina levantando un muro. Muchas personas y organizaciones intuyen el poder del aprendizaje automático, pero no logran aprovecharlo porque carecen de equipos especializados. Y al final, ese muro se traduce en un coste alto —no solo en dinero, sino también en oportunidades perdidas.

3.2. Problema técnico (PDS)

Desde el punto de vista técnico, los retos principales de este TFG no han estado tanto en los problemas clásicos del *machine learning*, sino en poder realizar **la integración entre modelos de lenguaje y bibliotecas de aprendizaje supervisado.**

Los desafíos concretos han sido los siguientes:

- **Procesamiento de Lenguaje Natural con LLMs**. Conseguir que el sistema entienda instrucciones expresadas en lenguaje natural, aunque estén formuladas de maneras distintas ("*describe la variable*", "*reduce al 30% el dataset*"), y que sea capaz de traducirlas con fiabilidad a operaciones concretas de preprocesamiento, selección o entrenamiento.

- **Mapeo entre lenguaje natural y funciones de ML**. No es suficiente con "entender" la frase, el agente debe conectar cada instrucción con la *Tool* de Python adecuada, validar que los parámetros tengan sentido (por ejemplo, que la columna exista) y ejecutar la transformación sin errores.

- **Gestión del estado de la interacción**. Una de las dificultades más notables es mantener la coherencia entre pasos sucesivos, es decir, que los cambios aplicados al *dataset*, se conserven en memoria y que el agente "recuerde" lo hecho al responder nuevas peticiones.

- **Interfaz accesible.** El último reto es encapsular todo lo anterior en una interfaz que elimine la necesidad de programar y sea lo suficientemente accesible como para mantener una conversación con el sistema.

Todos estos aspectos convergen en lo que se conoce como un **Problema de Diseño del Sistema (PDS):** cómo construir un entorno que asista al usuario en la configuración de pipelines supervisados de manera sencilla, reproducible y eficaz.

3.3. Funcionamiento esperado

El sistema propuesto funciona como un **asistente conversacional**. El usuario solo necesita cargar un *dataset* en formato CSV y expresar su necesidad en lenguaje natural. Una petición podría ser tan directa como: "*quiero predecir la variable precio aplicando normalización y usando un clasificador Random Forest*".

A partir de ahí, el asistente se encarga de orquestar todo el proceso:

1. **Interpretación de la orden**. Un modelo de lenguaje de gran tamaño (LLM) traduce la petición del usuario en un tipo de acción (análisis, preprocesamiento, selección de características, selección de instancias, entrenamiento y evaluación) y activa la *tool* para delegar al agente correspondiente.

2. **Activación de agentes especializados**. Cada etapa es gestionada por módulos diseñados para esa función.

3. **Resultados de los agentes especializados.** Los agentes especializados devuelven la respuesta al agente coordinador.

4. **Resultados con trazabilidad**. El coordinador devuelve los resultados del entrenamiento, asegurando transparencia y posibilidad de réplica.

5. **Interacción iterativa**. El usuario puede ajustar, refinar o modificar su petición mediante nuevas instrucciones en lenguaje natural, sin necesidad de escribir código directamente.

En esencia, el asistente convierte un diálogo sencillo en un *pipeline* completo de aprendizaje supervisado, reduciendo la complejidad técnica y acercando el poder del ML a más personas.

3.4. Entorno y vida esperada

- **Usuarios destinatarios**. El sistema está orientado a distintos perfiles: estudiantes de informática que busquen una herramienta didáctica, investigadores que necesiten realizar prototipados rápidos y profesionales de áreas no técnicas que requieran apoyo en la configuración de modelos sin entrar en los detalles del código.

- **Entorno de ejecución.** La aplicación se desarrolla sobre *Python 3.10*, haciendo uso de librerías de código abierto ampliamente adoptadas, como *scikit-learn* [15], *pandas* [16], *Panel* [12] y *CrewAI* [17].

- **Requisitos de hardware.** Para un funcionamiento fluido basta con un ordenador personal sin tener muchos requisitos.

- **Vida esperada**. El sistema está concebido inicialmente como un **prototipo de TFG**, aunque su diseño deja abierta la posibilidad de evolucionar hacia una herramienta de mayor alcance, ya sea en el ámbito educativo o en el de investigación aplicada.

Capítulo 4. Antecedentes y Estado del Arte

4.1. *AutoML* y pipelines en *machine learning*

La dificultad del aprendizaje automático no está solo en entrenar un modelo; el reto empieza mucho antes. Hay que preparar los datos, transformar las variables, decidir cómo normalizarlas y, finalmente, evaluar el resultado. Todo esto exige orquestar un proceso complejo y, a menudo, delicado. En este contexto, los *pipelines* en *scikit-learn* se han convertido casi en el estándar por defecto, porque permiten encadenar transformadores y algoritmos en un flujo ordenado y reproducible.

La llegada de *AutoML* supuso un salto más. Su promesa es tentadora: automatizar la selección de modelos y la optimización de hiperparámetros sin que el usuario tenga que intervenir en cada decisión. Frameworks como *Auto-sklearn* [18] y *TPOT* [19] han demostrado que pueden lograr configuraciones muy competitivas en escenarios reales. Sin embargo, y aquí está la otra cara de la moneda, estos enfoques priorizan el rendimiento automático frente a la accesibilidad. Requieren ciertos conocimientos técnicos para configurarse correctamente y no ofrecen una interacción natural con el usuario.

En este sentido, *AutoML* supone un gran avance en automatización, pero mantiene pendiente el desafío central que motiva este TFG: **reducir la barrera de entrada al aprendizaje automático mediante interfaces más intuitivas y conversacionales**.

4.2. Herramientas de referencia

Para poder fundamentar el desarrollo del TFG, se han analizado diversas herramientas con funcionalidades que han inspirado a la propuesta actual.

4.3.1. Amazon SageMaker

Amazon SageMaker es una plataforma de aprendizaje automático en la nube que permite a los desarrolladores y científicos de datos la construcción, entrenamiento y despliegue de modelos de ML sin necesidad de gestionar la infraestructura.

Entre sus características sobresalen la provisión de entornos preconfigurados que admiten múltiples frameworks de ML, la automatización de tareas como la selección y ajuste de hiperparámetros y la capacidad para escalar tanto el entrenamiento como la inferencia de modelos de forma más sencilla. Además, *SageMaker* incorpora herramientas para la preparación de datos, lo que facilita la ingesta, transformación y validación de grandes cantidades de información, lo que acelera el avance de soluciones de ML y disminuye los tiempos de implementación [20].

Si bien ofrece un entorno robusto y escalable, está orientado a usuarios con conocimientos técnicos avanzados y se integra de manera estrecha con el ecosistema AWS. En contraste, la solución propuesta en este TFG se centra en simplificar la configuración de *pipelines* a través de interacciones en lenguaje natural, haciendo el proceso accesible para usuarios con menos experiencia técnica.

4.3.2. InstructPipe

InstructPipe es un *framework* innovador que se centra en la generación automática de pipelines de aprendizaje automático a partir de instrucciones expresadas en lenguaje natural. Su enfoque permite que usuarios sin conocimientos técnicos avanzados puedan definir flujos de procesamiento complejos, ya que el sistema interpreta los comandos y traduce estas indicaciones en configuraciones técnicas precisas. Esto no solo simplifica la adopción de ML, sino que también optimiza el proceso de creación de soluciones personalizadas, permitiendo una rápida iteración y experimentación en el desarrollo de modelos [21].

A diferencia de la aplicación propuesta en este TFG, *InstructPipe* transforma instrucciones en pipelines de manera automática, pero bajo un enfoque cerrado, es decir, el usuario puede dar una orden inicial en lenguaje natural, pero después no tiene margen para intervenir ni ajustar el flujo según sus necesidades. En otras palabras, el sistema interpreta y ejecuta, pero no mantiene una interacción conversacional. Tampoco emplea un planteamiento multiagente, sino que está diseñado para interpretar comandos de forma secuencial sin permitir interacciones dinámicas en tiempo real.

En cambio, el asistente desarrollado en este TFG apuesta por una interacción conversacional continua y por una arquitectura multiagente flexible, donde cada módulo puede responder, delegar y adaptarse al contexto. Esto convierte la construcción de *pipelines* en un proceso más personalizable, transparente y abierto a la exploración que el que ofrece *InstructPipe*.

4.3.3. DSPy

DSPy representa una aproximación moderna al desarrollo de flujos de datos y pipelines de ML mediante el uso de *LLMs*. Esta herramienta está diseñada para automatizar tareas de exploración y preprocesamiento de datos, integrando técnicas avanzadas de procesamiento de lenguaje natural para interpretar y ejecutar instrucciones en lenguaje cotidiano. Al aprovechar la capacidad de los *LLMs*, *DSPy* facilita la configuración de pipelines complejos, desde la identificación de patrones en los datos hasta la selección de algoritmos adecuados para el entrenamiento, mejorando la eficiencia y reduciendo la dependencia del conocimiento técnico especializado [22].

Sin embargo, su enfoque se orienta a la automatización directa, es decir, el usuario introduce una instrucción en lenguaje natural y el sistema genera un pipeline completo, pero lo hace de manera cerrada y sin etapas intermedias de validación o ajuste. La interacción, por tanto, se limita al punto de partida, sin que exista un dialogo continuo que permita revisar o modificar la configuración sobre la marcha.

En cambio, en este TFG, se potenciará la interacción del usuario, permitiendo ajustes en tiempo real y ofreciendo un sistema más flexible que se pueda adaptar a distintas necesidades.

4.3. CrewAI

CrewAI [17] es un *framework* en Python diseñado para simplificar la construcción de sistemas multiagente basado en modelos de lenguaje de gran tamaño.
En el contexto de este TFG, *CrewAI* se adopta como una de las bases tecnológicas, ya que proporciona un modelo flexible para organizar agentes, asignarles herramientas concretas y establecer dinámicas de colaboración.

Dado que forma una de las piezas más importantes del proyecto, se analizará en detalle en el Capítulo 8 (ver Capítulo 8).

4.4. Metodología de búsqueda y recopilación de información

Para dar forma a este TFG no bastó con una única revisión bibliográfica. Fue un proceso largo, casi constante, de búsqueda, prueba y ajuste. Hubo que investigar, recopilar y contrastar información de distintas fuentes, y ese camino se convirtió en la base sobre la que se apoyaron las decisiones técnicas y metodológicas que marcaron el proyecto.

Al principio, el foco estuvo en comprender los fundamentos de los *LLMs*. Para ello, resultaron de gran ayuda artículos divulgativos y guías técnicas, como el trabajo de *Cryptoniche* [23], que permitió visualizar la arquitectura general y entender como encajan sus diferentes piezas. A la par, se empezó a instalar y realizar pruebas en local de algunos *LLMs* mediante *LM Studio* [24]. Sin embargo, pronto apareció una dificultad muy concreta: los requisitos de GPU superaban el hardware disponible. Esa limitación empujó a explorar otras vías y, finalmente, a optar por soluciones en la nube. En este caso, la API de Gemini ofreció justo lo que se necesitaba: un acceso más ligero, sencillo y, además, gratuito para experimentar.

Mientras tanto, también se abrió otra línea de investigación: la construcción de la interfaz de usuario. En un primer momento se valoraron frameworks web como *Flutter* [25], que habrían permitido un despliegue multiplataforma. Sin embargo, se tomó la decisión

consciente de priorizar lo esencial —el núcleo del sistema — y se apostó por un enfoque local con Panel, sacrificando algo de alcance a cambio de simplicidad y foco.

Más adelante, la atención se desplazó hacia la arquitectura multiagente. Aquí la guía práctica de *Bhavik Jikadara* [26] resultó especialmente valiosa. Mostraba como aplicar *CrewAI* a la automatización de flujos de ciencia de datos y sirvió como chispa para diseñar prototipos con agentes especializados en tareas muy concretas: carga de datos, análisis, preprocesamiento y entrenamiento. En esta etapa, el intercambio de correos con los tutores fue clave: ayudó a descartar la búsqueda automática de *datasets* por internet y a centrar el proyecto en la interacción directa con el usuario.

No obstante, el trabajo no se quedó solo en lo teórico. Hubo una fase interna de experimentación: se probaron *prompts* distintos para cada agente, se afinó el rol del agente coordinador y hasta se tanteó la posibilidad de delegación dinámica entre agentes subordinados. Muchos de estos avances vinieron de ejemplos compartidos por la comunidad y de fragmentos de código de referencia que, eso sí, hubo que adaptar para ajustarlos a las necesidades concretas del proyecto.

Finalmente, la filosofía del sistema quedó más clara: no se buscaba un asistente que planificara de forma automática toda la ejecución, sino uno capaz de responder de manera interactiva, contextual y flexible a cada petición del usuario. Y es que esa flexibilidad, aunque más exigente, se convirtió en el verdadero valor del proyecto. Para alcanzarla, fue necesario un esfuerzo adicional de investigación, sobre todo porque la documentación oficial de *CrewAI* era bastante limitada. Hubo que ir armando el puzle con piezas dispersas: artículos técnicos, foros, videos de *Youtube* y, por supuesto, pruebas directas en local.

Capítulo 5. Restricciones

Como todo proyecto real, este Trabajo Fin de Grado no se ha desarrollado en un vacío ideal, sino dentro de unos límites claros que han marcado tanto su alcance como su forma de ejecución. Entender estas restricciones es fundamental para situar la propuesta en su contexto y valorar sus resultados con realismo.

5.1. Restricciones humanas

El trabajo se realiza por un estudiante, guiado por dos tutores académicos, aunque él asume la carga de investigación, desarrollo y redacción.

En la práctica, los 12 créditos ECTS asignados determinaron la dedicación, que equivale aproximadamente a 300 horas de trabajo distribuidas durante el curso. Aunque puede parecer mucho tiempo, esas horas transcurren rápidamente cuando se trata de aprender nuevas técnicas, resolver problemas y documentar cada paso.

Además, el conocimiento previo del estudiante en aprendizaje automático y en el uso de librerías de *Python* fue un factor determinante ya que no disponía de un conocimiento muy amplio en este sector. No es lo mismo enfrentarse a un terreno ya explorado que abrirse camino casi desde cero; esa curva de aprendizaje ha marcado hasta donde pudo llegar la complejidad de las técnicas aplicadas.

5.2. Restricciones económicas

En cuanto a este punto, no hubo presupuestos extraordinarios ni ayudas externas. Todo se apoyó en recursos personales, en el ecosistema *open source*, con librerías como *scikit-learn*, *pandas*, *Panel* o *CrewAI* y en el uso de un *api key* gratuita con *Gemini.*

Por otro lado, tampoco se invirtió en licencias de software ni en servicios de computación en la nube. Dicho de otro modo: el proyecto se construyó con lo que se disponía, buscando siempre alternativas gratuitas y accesibles.

5.3. Restricciones temporales

El TFG tenía que completarse dentro del curso académico 2024/25. Esto implicaba un calendario claro, con fases bien delimitadas: **investigación, requisitos, diseño, implementación, pruebas y redacción.**

Las fechas de entrega y defensa fueron una especie de reloj de arena que limitaba las posibilidades de experimentación. No fue posible, por ejemplo, probar el sistema con

muchos diversos *datasets*, algo que habría sido enriquecedor pero irrealizable en el plazo marcado.

5.4. Restricciones técnicas y de hardware

El trabajo se realiza mediante un portátil personal con estas especificaciones:

- **CPU**: Intel Core i5-12450H 2.0GHz.
- **GPU**: NVIDIA RTX 3050.
- **RAM**: 16 GB.
- **Almacenamiento**: 512 GB.

Estas características han hecho posible la realización de pipelines supervisados de tamaño medio. Además, la implementación del asistente se ha restringido a entornos locales, sin que se extienda a la nube.

5.5. Restricciones de software

El sistema operativo utilizado es *Windows 11*, con un entorno de desarrollo en *Python 3.10* mediante el *IDE* de *Visual Studio*.

Las librerías elegidas son todas de libre distribución: *scikit-learn*, *pandas*, *CrewAI* y *LangChain*. El control de versiones se gestiona con *Git* y *GitHub*, lo que permite mantener el orden en proyectos complejos.

En cuanto al uso de *APIs* externas —por ejemplo, para los *LLMs* —, está condicionado por claves de acceso y, sobre todo, por las limitaciones de los planes gratuitos. Una realidad que cualquier estudiante reconoce al instante.

5.6. Restricciones reglamentarias

Por último, el marco institucional también marcó su huella. El trabajo debería cumplir con las normas de la Universidad de Córdoba para la elaboración de TFGs, lo que implicaba:

- Entregar la memoria en un formato académico especifico.
- Incluir un resumen en español e inglés.
- Garantizar la originalidad.

Y, más allá de la forma, también había un fondo: el proyecto debía respetar el código ético y las buenas prácticas de la titulación. En otras palabras, no se trataba solo de llegar a una solución técnica, sino de hacerlo con rigor y responsabilidad.

Capítulo 6. Recursos

El desarrollo de este Trabajo Fin de Grado ha necesitado apoyarse en distintos recursos, tanto humanos como materiales y de software. A continuación, se detallan los más relevantes, aquellos que realmente hicieron posible llevar el proyecto de la idea al resultado final.

6.1. Recursos humanos

- **Alumno**.
    - **Isaac Manuel Cejudo Alfaro**, estudiante del Grado en Ingeniería Informática de la Universidad de Córdoba. Es el principal responsable del diseño, implementación, pruebas y redacción de la memoria del TFG. En definitiva, quien ha llevado el peso del día a día del proyecto.

- **Directores**.

    - **Dr. José Raúl Romero Salguero,** Departamento de Informática y Análisis Numérico de la Universidad de Córdoba.

    - **Dr. Carlos García Martínez**, Departamento de informática y Análisis Numérico de la Universidad de Córdoba.

    El rol de los tutores será supervisar el trabajo, orientar metodológicamente cada fase y, sobre todo, garantizar que el resultado final esté alineado con los criterios académicos de la titulación.

6.2. Recursos Hardware

- **Ordenador Portátil.**

    - CPU: Intel Core i5-12450H 2.0GHz

    - GPU: NVIDIA RTX 3050

    - RAM: 16 GB.

    - Almacenamiento: 512 GB.

6.3. Recursos Software

- **Sistema operativo.**

    - Windows 11.

- **IDE.**

    - Visual Studio.

- **Lenguajes de programación.**
    - Python 3.10.
- **Librerías**.
    - TensorFlow.
    - Scikit-Learn.
    - Pandas y numpy.
    - Joblib
- **Control de versiones.**
    - Git.
    - GitHub.
- **Frameworks para agentes y LLMs.**
    - CrewAI
    - LangChain
- **Framework de interfaz de usuario:**
    - Panel
- **Gestión de entornos y dependencias:**

- Archivo *pyproject.toml*

- **Gestion de variables de entorno.**

  - Archivo. *env*

Capítulo 7. Especificación de requisitos

El sistema no puede construirse de cualquier manera: necesita cumplir un conjunto de requisitos que aseguren su verdadera utilidad, su facilidad de uso y, por supuesto, su solidez técnica.

Para organizar estos requisitos y mantener un enfoque claro, se han clasificado en dos grandes grupos —funcionales y no funcionales —, siguiendo las metodologías habituales en Ingeniería del Software. Esta división permite distinguir, por un lado, lo que el sistema debe hacer y, por otro, como debe hacerlo, garantizando así una visión completa y equilibrada del proyecto.

Para la elaboración de este capítulo, se ha utilizado como bibliografía un tema de la asignatura "*Diseño y Construcción del Software*" [27].

7.1. Listado de requisitos funcionales (RF)

Los requisitos funcionales definen lo que debería hacer el sistema. A continuación, se muestra el listado de requisitos funcionales identificados para el sistema, clasificados según su categoría.

- **Gestión de datos**

  - **RF1**. El sistema debería permitir la carga de *datasets* en formato CSV, validando su estructura antes de usarlos.

- **RF2.** El sistema debería ofrecer un módulo de análisis exploratorio de datos (EDA), por ejemplo, el cálculo de valores medios de variables numéricas.

- **RF3.** El sistema debería permitir operaciones de preprocesamiento de datos, por ejemplo, la limpieza de valores nulos.

- **RF4.** El sistema debería ofrecer mecanismos de selección de instancias y características, por ejemplo, reducir el *dataset* a un 30%.

- **RF5.** El sistema debe permitir la división del *dataset* en conjuntos de entrenamiento, validación y prueba, tanto de forma estratificada como temporal.

- **Modelado y evaluación.**

  - **RF6.** El sistema debería permitir la configuración y entrenamiento de modelos supervisados de clasificación y regresión, por ejemplo, usando un *Random Forest*.

  - **RF7.** El sistema debería integrar un módulo de evaluación de modelos donde se puedan utilizar distintas métricas según el tipo de problema.

  - **RF8.** El sistema debería permitir la exportación de modelos entrenados y de sus métricas.

  - **RF9**. El sistema debería permitir al usuario establecer la semilla de los generadores de números aleatorios internos.

- **Interacción con el usuario**

- **RF10.** El sistema debería incluir un asistente basado en *LLMs* que interprete ordenes en lenguaje natural y sea capaz de traducirlas en acciones técnicas.

- **RF11.** El sistema debería permitir la visualización de los datos del *dataset.*

7.2. Requisitos no funcionales (RNF)

Los requisitos no funcionales especifican o restringen cómo se debería implementar el sistema. A continuación, se muestra el listado de los requisitos no funcionales identificados.

- **RNF1**. El sistema debería responder a una petición en lenguaje natural en 30 segundos o menos.

- **RNF2**. El sistema debería garantizar la reproducibilidad de resultados mediante fijación de semillas y registro de parámetros.

- **RNF3**. La arquitectura debería ser modular y extensible.

- **RNF4**. El sistema debería ser escrito en *Python 3.10* y basarse únicamente en librerías de software libre.

- **RNF5**. En caso de error, el sistema debería detener el pipeline de forma controlada y notificar al usuario sin corromper los datos.

- **RNF6**. Todos los artefactos deben almacenarse en local.

7.3. Priorización de requisitos

Los requisitos funcionales y no funcionales definidos en este capítulo no tienen todos la misma urgencia ni el mismo peso dentro del proyecto. Para establecer un orden claro, se ha utilizado el criterio ***MoSCoW*** [28], una técnica ampliamente empleada en gestión de requisitos.

Este enfoque divide cada requisito en cuatro niveles de prioridad:

- **M (Must have):** Son los imprescindibles, aquellos sin los cuales el sistema simplemente no podría considerarse funcional.

- **S (Should have):** Muy importantes, aunque no críticos en una primera versión. Su ausencia no rompe el sistema, pero si limita parte de su utilidad.

- **C (Could have):** Opcionales. Aportan valor añadido y enriquecen la experiencia, siempre que haya tiempo y recursos disponibles para incluirlos.

- **W (Won´t have):** No se implementarán en esta versión, aunque podrían ser candidatos a futuro si el proyecto evoluciona.

De esta forma, no solo se define lo que el sistema debe cumplir, sino también el orden lógico en el que cada requisito debe abordarse. La tabla 1 y 2 recoge esta clasificación de manera detallada.

Requisito	Descripción	Prioridad
RF1	El sistema debería permitir la carga de datasets.	M
RF2	El sistema debería ofrecer un módulo de análisis exploratorio de datos (EDA).	M
RF3	El sistema debería permitir operaciones de preprocesamiento de datos.	M

RF4	El sistema debería ofrecer mecanismos de selección de instancias y características.	M
RF5	El sistema debe permitir la división del *dataset* en conjuntos de entrenamiento, validación y prueba, tanto de forma estratificada como temporal.	M
RF6	El sistema debería permitir la configuración y entrenamiento de modelos supervisados de clasificación y regresión.	M
RF7	El sistema debería integrar un módulo de evaluación de modelos	M
RF8	El sistema debería permitir al usuario establecer la semilla de los generadores de números aleatorios internos.	S
RF9	El sistema debería permitir la exportación de pipelines y modelos entrenados.	S
RF10	El sistema debería incluir un asistente basado en LLMs que interprete ordenes en lenguaje natural y sea capaz de traducirlas en acciones técnicas.	M
RF11	El sistema debería permitir la visualización de los datos del *dataset*	S

Tabla 1. Priorización de requisitos funcionales

Requisito	Descripción	Prioridad
RNF1	El sistema debería responder a una petición en lenguaje natural en 30 segundos o menos.	S
RNF2	El sistema debería garantizar la reproducibilidad de resultados mediante fijación de semillas y registro de parámetros.	M
RNF3	La arquitectura debería ser modular y extensible.	M
RNF4	El sistema debería ser escrito en Python 3.10 y basarse únicamente en librerías de software libre.	M
RNF5	En caso de error, el sistema debería detener el pipeline de forma controlada y notificar al usuario sin corromper los datos.	S

RNF6	Todos los artefactos deben almacenarse en local.	M

Tabla 2. Priorización de requisitos no funcionales

7.4. Modelado de Casos de Uso

El modelado de casos de uso muestra, de una forma bastante clara, como las personas o sistemas que interactúan con la aplicación logran cubrir lo que realmente necesitan, es decir, esos requisitos funcionales que se definieron anteriormente (ver Capítulo 7.1). En el fondo, es como un puente que conecta dos frentes: por un lado, la etapa donde descubrimos qué se espera del sistema, y por el otro, el análisis y diseño que harán posible darle forma real.

Para la elaboración de esta sección, se ha utilizado como bibliografía otro tema de la asignatura "*Diseño y Construcción del Software*" [29].

7.5. Identificación de los actores

Los actores son un papel (**rol**) que una entidad externa adopta cuando interactúa directamente con el sistema. En la tabla 3 se muestra la identificación de los actores que participaran de una forma u otra en el sistema.

Nombre del Actor	Breve descripción
Usuario	Persona que hace uso de toda la funcionalidad del sistema.
LLM	Actor que interpreta las instrucciones en lenguaje natural.
Almacenamiento	Actor "especial" almacenamiento

Tabla 3. Identificación de actores

7.6. Identificación de Casos de Uso

Un Caso de Uso se puede definir como "*especificación de una secuencia de acciones, incluyendo variante y errores, que un sistema puede realizar al interactuar con agentes externos*" [30]. La tabla 4 enumera los casos de uso del sistema y una breve descripción de estos.

Identificador	Nombre de los Casos de Uso	Breve descripción
CU1	CargarDataset	El usuario importa un *dataset* valido al sistema.
CU2	VisualizarDataset	El usuario solicita al sistema una previsualización de los datos del *dataset*.
CU3	RealizarEDA	El usuario solicita al sistema un análisis exploratorio de datos del *dataset*.
CU4	PreprocesarDatos	El usuario solicita al sistema un preprocesamiento de datos del *dataset.*
CU5	SeleccionarInstanciasCaracteristicas	El usuario solicita al sistema reducir el *dataset* mediante eliminación de algunos datos
CU6	EntrenarModelo	El usuario solicita al sistema que genere un pipeline mediante un algoritmo en concreto.
CU7	EvaluarModelo	El usuario solicita al sistema que calcule métricas de rendimiento sobre el modelo entrenado.
CU8	ExportarArtefactos	El usuario solicita al sistema guardar el modelo entrenado y sus métricas.
CU9	InterpretarOrdenLN	El LLM recibe la instrucción del sistema y decide que hacer.

Tabla 4. Identificación de Casos de Uso

7.7. Diagrama de Casos de Uso

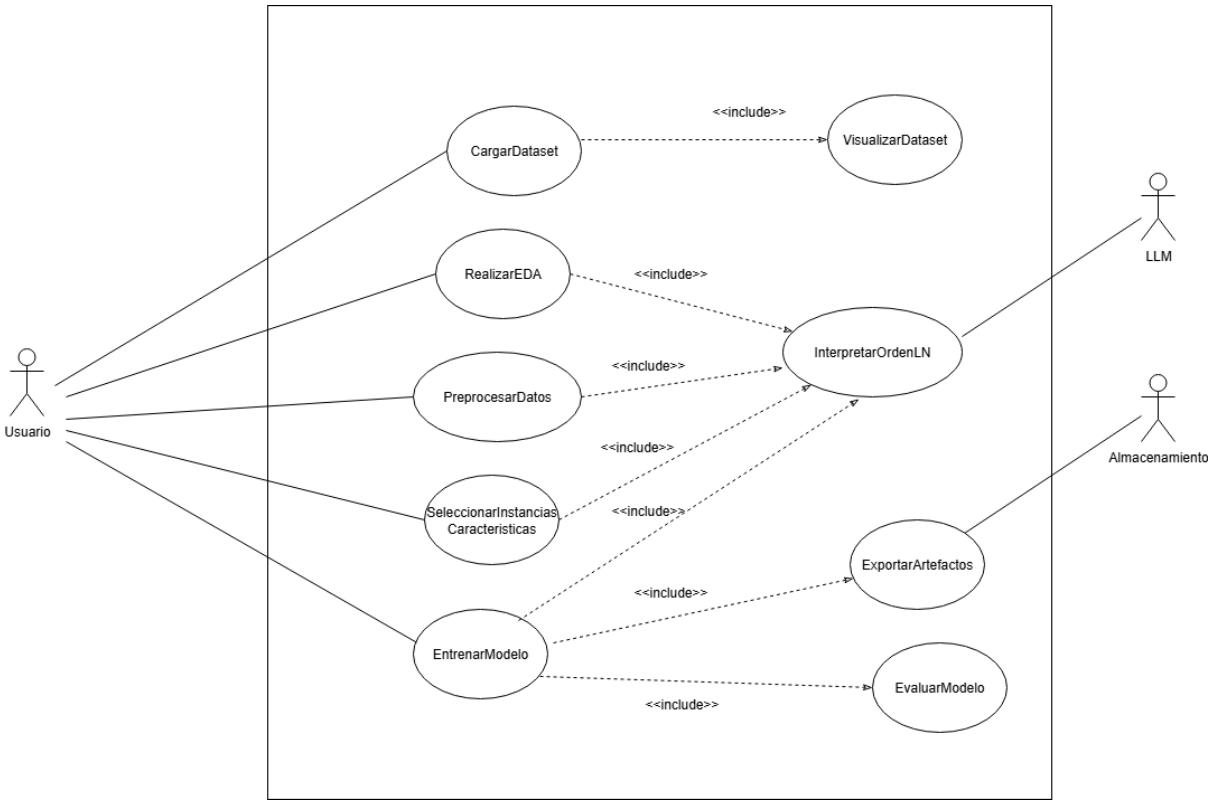

Figura 1. Diagrama de Casos de Uso

7.8. Casos de Uso

Nombre	CargarDataset
Identificador	CU-1
Descripción	El usuario importa un *dataset* valido al sistema.
Actor/es principal/es	Usuario
Actor/es secundario/s	*Ninguno*
Precondiciones	1. El sistema debe estar iniciado y el usuario disponer de un archivo valido.
Flujo principal	1. El caso de uso se activa cuando el usuario selecciona la opción “cargar dataset” 2. El usuario selecciona el *dataset* en su formato valido.

	3. El sistema valida que el formato sea correcto. 4. El sistema carga los datos en memoria. 5. El sistema confirma al usuario que el *dataset* está disponible. 6. Include(VisualizarDataset)
Postcondición	El *dataset* queda cargado en memoria.
Flujos alternativos	Si el archivo no es válido o está corrupto, el sistema muestra un error y solicita otro fichero.

Tabla 5. Especificación CU1

Nombre	VisualizarDataset
Identificador	CU-2
Descripción	El sistema muestra una previsualización de los datos del *dataset*.
Actor/es principal/es	Usuario
Actor/es secundario/s	*Ninguno*
Precondiciones	1. El *dataset* debe estar cargado previamente.
Flujo principal	1. El caso de uso se activa cuando el usuario selecciona "visualizar dataset" 2. El sistema muestra la información de todos los datos del *dataset*.
Postcondición	*Ninguno*.
Flujos alternativos	*Ninguno.*

Tabla 6. Especificación CU2.

Nombre	RealizarEDA
Identificador	CU-3

Descripción	El usuario solicita al sistema un análisis exploratorio de datos del *dataset*.
Actor/es principal/es	Usuario
Actor/es secundario/s	*Ninguno*
Precondiciones	1. El *dataset* debe estar cargado previamente.
Flujo principal	3. El caso de uso se activa cuando el usuario selecciona la opción "realizar eda" 4. El sistema genera las estadísticas de los datos pertinentes del *dataset*. 5. El sistema muestra los resultados en pantalla.
Postcondición	*Ninguno*
Flujos alternativos	*Ninguno*

Tabla 7. Especificación CU3.

Nombre	PreprocesarDatos
Identificador	CU-4
Descripción	El usuario solicita al sistema un preprocesamiento de datos del *dataset.*
Actor/es principal/es	Usuario
Actor/es secundario/s	*Ninguno*
Precondiciones	1. El *dataset* debe estar cargado previamente.
Flujo principal	1. El caso de uso se activa cuando el usuario selecciona "preprocesar datos" 2. El sistema aplica las transformaciones al *dataset.* 3. El sistema guarda el *dataset* modificado. 4. El sistema muestra un informe con los cambios realizados.

Postcondición	El *dataset* queda transformado.
Flujos alternativos	1. Si alguna técnica falla, el sistema notifica el error y sugiere opciones.

Tabla 8. Especificación CU4

Nombre	SeleccionarInstanciasCaracteristicas
Identificador	CU-5
Descripción	El usuario solicita al sistema reducir el *dataset* mediante eliminación de algunos datos.
Actor/es principal/es	Usuario
Actor/es secundario/s	*Ninguno*
Precondiciones	1. El *dataset* debe estar cargado previamente.
Flujo principal	1. El caso de uso se activa cuando el usuario selecciona “seleccionar instancias y características” 2. El usuario indica los criterios de selección. 3. El sistema filtra el *dataset* según las instrucciones. 4. El sistema almacena el *dataset* modificado. 5. El sistema muestra un informe con los cambios realizados en el *dataset*
Postcondición	El *dataset* queda transformado.
Flujos alternativos	1. Si se produce algún error, el sistema lo notifica y sugiere opciones.

Tabla 9. Especificación CU5

Nombre	EntrenarModelo
Identificador	CU-6

Descripción	El usuario solicita al sistema que genere un pipeline mediante un algoritmo en concreto.
Actor/es principal/es	Usuario
Actor/es secundario/s	*Ninguno*
Precondiciones	1. El *dataset* debe estar cargado previamente. 2. El *dataset* debe haber sido preprocesado previamente.
Flujo principal	1. El caso de uso se activa cuando el usuario selecciona "entrenar modelo" 2. El usuario elige un algoritmo de clasificación o regresión. 3. El sistema crea un *pipeline* con los pasos definidos. 4. El sistema entrena el modelo con los datos. 5. Include (EvaluarModelo) 6. Include (ExportarPipeline) 7. El sistema notifica los parámetros usados.
Postcondición	El modelo queda disponible.
Flujos alternativos	1. Si se produce algún error, el sistema lo notifica y sugiere opciones.

Tabla 10. Especificación CU6

Nombre	EvaluarModelo
Identificador	CU-7
Descripción	El usuario solicita al sistema que calcule métricas de rendimiento sobre el modelo entrenado.
Actor/es principal/es	Usuario
Actor/es secundario/s	*Ninguno*

Precondiciones	1. El modelo debe haber sido entrenado previamente.
Flujo principal	1. El caso de uso se activa cuando el usuario selecciona "evaluar modelo" 2. El sistema calcula las métricas. 3. El sistema muestra los resultados en pantalla.
Postcondición	*Ninguno*
Flujos alternativos	*Ninguno*

Tabla 11. Especificación CU7

Nombre	ExportarArtefactos
Identificador	CU-8
Descripción	El usuario solicita al sistema guardar el modelo entrenado y las metricas.
Actor/es principal/es	Usuario, Almacenamiento
Actor/es secundario/s	*Ninguno*
Precondiciones	1. El modelo debe haber sido entrenado previamente.
Flujo principal	1. El caso de uso se activa cuando el usuario o el almacenamiento selecciona "exportar pipeline" 2. El sistema genera ficheros en el formato correspondiente. 3. El sistema guarda los ficheros. 4. El sistema confirma la exportación con éxito.
Postcondición	Artefactos almacenados en el sistema.
Flujos alternativos	*Ninguno*

Tabla 12. Especificación CU8

Nombre	InterpretarOrdenLN

Identificador	CU-9
Descripción	El LLM recibe la instrucción del sistema y decide que hacer.
Actor/es principal/es	LLM
Actor/es secundario/s	*Ninguno*
Precondiciones	*Ninguno*
Flujo principal	1. El caso de uso se activa cuando el LLM selecciona "interpretar orden lenguaje natural" 2. El sistema envía el texto al LLM 3. El LLM interpreta la orden en lenguaje natural y la traduce a un conjunto de acciones a realizar 4. El LLM retorna las acciones a realizar.
Postcondición	*Ninguno*
Flujos alternativos	*Ninguno*

Tabla 13. Especificación CU9

7.9. Matriz de trazabilidad

La matriz de trazabilidad es un instrumento que permite relacionar los requisitos del sistema y los distintos casos de uso. Su finalidad es garantizar que cada requisito este cubierto por al menos un caso de uso.

	CU1	CU2	CU3	CU4	CU5	CU6	CU7	CU8	CU9
RF1	X								
RF2			X						
RF3				X					
RF4					X				
RF5					X	X			
RF6						X			
RF7							X		
RF8								X	

RF9					X	X			
RF10									X
RF11		X							

7.10. Glosario

El glosario es un conjunto de palabras (ordenadas alfabéticamente), junto con sus definiciones, que versan sobre una disciplina. El objetivo principal es definir los términos clave del proyecto y resolver sinónimos y homónimos.

Se ha definido los siguientes términos claves para el proyecto:

- **Artefactos**: Son los "productos" que el sistema genera durante su funcionamiento. Pueden ser modelos entrenados, pipelines listos o métricas exportadas en ficheros.

- **Asistente**: El corazón del sistema. Es el componente que coordina la ejecución de los agentes internos con las instrucciones enviadas por el usuario.

- **Dataset**: Conjunto estructurado de datos que se carga en el sistema.

- **EDA (Exporatory Data Analysis):** Permite describir, resumir y entender las características del dataset antes de dar el siguiente paso.

- **LLM (Large Language Model):** Modelos de lenguaje a gran escala capaces de entender instrucciones en lenguaje natural y responder con texto o planes de acción coherentes.

- **Pipeline**: Una cadena de pasos ordenados que forman un flujo reproducible dentro de un proceso de aprendizaje automático.

- **Almacenamiento**: Recurso externo donde se guardan los *datasets* de entrada y los resultados exportados.

- **Usuario**: El actor humano principal. Es quien da órdenes al sistema, lo pone en marcha y, al final, consume los resultados que este produce.

Capítulo 8. Análisis del sistema

El análisis de este TFG se centra en identificar y describir las tecnologías y librerías que constituyen la base del sistema, así como en estudiar la organización de los agentes que lo conforman. El objetivo es ofrecer una visión técnica solida que justifique las decisiones tomadas en fases posteriores de diseño e implementación.

A continuación, se detallan los pilares fundamentales del sistema junto con la propuesta organizativa de los agentes.

8.1. CrewAI

CrewAI [17] es un *framework* en *Python* pensado para hacer más sencilla la construcción de sistemas multiagente basados en LLMs. Su idea central es coordinar varios agentes especializados —cada uno con un rol y una función clara — para que trabajen en equipo y resuelvan tareas complejas de manera colaborativa.

8.1.1. Elementos principales

- **Agentes (Agents):** son el corazón de *CrewAI*. Cada agente se define con un *role* (rol), un *goal* (objetivo) y un *backstory* (una especie de historia o contexto narrativo que guía su comportamiento). Además, pueden contar con *tools*, es decir, herramientas que les permiten ejecutar operaciones concretas sobre datos.

- **Tareas (Tasks):** representan objetivos específicos que se asignan a los agentes. Una *task* describe qué debe hacerse, qué agente se encargará de ello y qué salida se espera obtener.

- **Herramientas (Tools)**: Son funciones o adaptadores que conectan a los agentes con el entorno o con librerías externas. Por ejemplo, una herramienta para cargar un *CSV* con *pandas* o para entrenar un modelo con *scikit-learn*.

- **Crews**: grupos de agentes que trabajan en conjunto, normalmente bajo la supervisión de un coordinador.

8.1.2. Funcionamiento interno

El funcionamiento interno de CrewAI depende del proceso de ejecución elegido para la *crew* (*sequential* o *hierarchical*). Sin embargo, siempre parte de los mismos elementos básicos: ***agents***, ***tasks*** y ***tools***.

- **Inicialización de la Crew.**

    1. Se definen los agentes (*Agents*), cada uno con su rol, objetivo y el conjunto de herramientas que tendrán disponibles.

    2. Se establecen las tareas (*Tasks*), que se asignan a un agente responsable.

    3. Se selecciona el proceso de orquestación:

        - **Sequential**: Las tareas se ejecutan en el orden exacto en el que fueron definidas.

        - **Hierarchical**: Un agente coordinador (*ManagerAgent*) recibe la petición y decide que agentes subordinados activar en cada momento.

- **Interpretación de la instrucción.**

La *Crew* recibe un input inicial, ya sea del usuario o del entorno.

- En modo secuencial, las tareas se ejecutan siguiendo la lista.
- En modo jerárquico, el *ManagerAgent* interpreta la instrucción y selecciona el agente más adecuado para comenzar.

- **Asignación y ejecución de tareas.**

  Cada *task* activa al agente correspondiente. Este puede apoyarse en sus *tools* para realizar operaciones concretas. Los agentes subordinados devuelven sus resultados intermedios al coordinador, que los consolida antes de decidir los siguientes pasos.

- **Delegación y validación.**

  El *ManagerAgent* revisa los resultados obtenidos. Si son correctos, puede decidir delegar nuevas tareas a otros agentes. Este ciclo se repite tantas veces como sea necesario hasta que la petición queda completamente resuelta.

- **Generación de la salida.**

  Una vez completado el proceso, el sistema devuelve el resultado final al usuario. Además, si la *crew* cuenta con memoria configurada, estos resultados se almacenan para futuras interacciones, lo que permite construir un historial de contexto útil para siguientes peticiones.

En la figura 2 se muestra el funcionamiento de *CrewAI*.

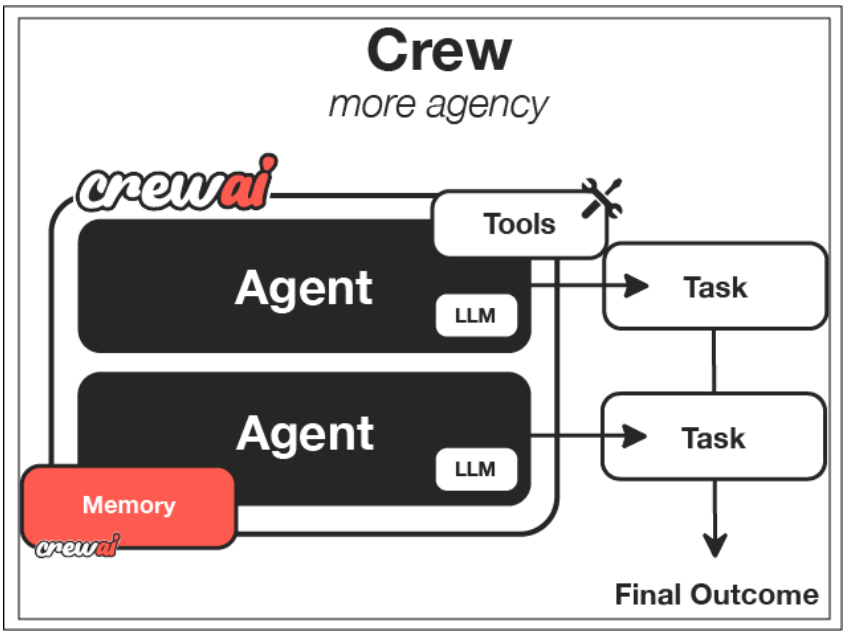

Figura 2. Funcionamiento de CrewAI

8.1.3. Modelo jerárquico

CrewAI permite trabajar con arquitecturas jerárquicas, en las que un **Coordinador** (*ManagerAgent*) actúa como supervisor y decide cuando y en qué momento delegar tareas a los agentes subordinados. Este patrón evita que todos los agentes se activen a la vez, lo que asegura un uso más eficiente: solo se ejecutan aquellos realmente necesarios para resolver la petición del usuario.

Un ejemplo práctico podría ser:

1. Usuario: “Normaliza los datos y entrena un Random Forest”.

2. El Coordinador invoca primero al Agente de Preprocesamiento, que se encarga de aplicar la normalización.

3. Después, delega en el Agente de Entrenamiento, que configura y entrena el modelo solicitado.

4. Finalmente, el sistema devuelve el resultado al usuario, acompañado de una explicación en lenguaje natural que facilita la comprensión de lo que ha ocurrido en segundo plano.

8.2. scikit-learn

scikit-learn [15] es la librería de Python para el aprendizaje automático clásico. Se caracteriza por ofrecer implementaciones robustas y eficientes de algoritmos de clasificación, regresión y *clustering* [31], junto con un amplio conjunto de utilidades para preprocesamiento, selección de características y evaluación.

En el contexto de este TFG, *scikit-learn* funciona como el motor central de los procesos de entrenamiento y evaluación:

- **Preprocesamiento**: normalización de variables, discretización, generación de variables dummy, etc.

- **Modelado**: Entrenamiento de algoritmos supervisados como *Random Forest*, *Regresion Lineal*, *SVM* y *Logistic Regression*.

- **Evaluación**: Calculo de métricas de rendimiento como *Accuracy*, *MAE*, o *RMSE*.

Esta librería tiene una gran relevancia debido a la madurez que conlleva, su amplia documentación y la reproducibilidad que ofrece, garantizando la validez de los experimentos.

8.3. Panel

Panel [12] es un *framework* en *Python* para crear interfaces interactivas basadas en *dashboards*. Destaca por su flexibilidad de composición, que permite organizar componentes con mayor libertad y personalizar el diseño de la interfaz. Estas características lo convierten en una opción muy adecuada para una arquitectura modular como la de este TFG.

En el contexto de este TFG, *Panel* actuará como la capa de interacción entre el usuario y el asistente:

- **Carga de *datasets***: permitirá subir archivos CSV y visualizar los datos de dichos *datasets*.

- **Comunicación en lenguaje natural**: Habilitará un chat interactivo donde el usuario podrá dar instrucciones al asistente sin necesidad de programar.

8.4. Organización de agentes en el sistema

El asistente se articula en torno a un *CoordinatorAgent* y un conjunto de agentes especializados, que se reparten las distintas fases del *pipeline*. La figura 3 ilustra esta arquitectura.

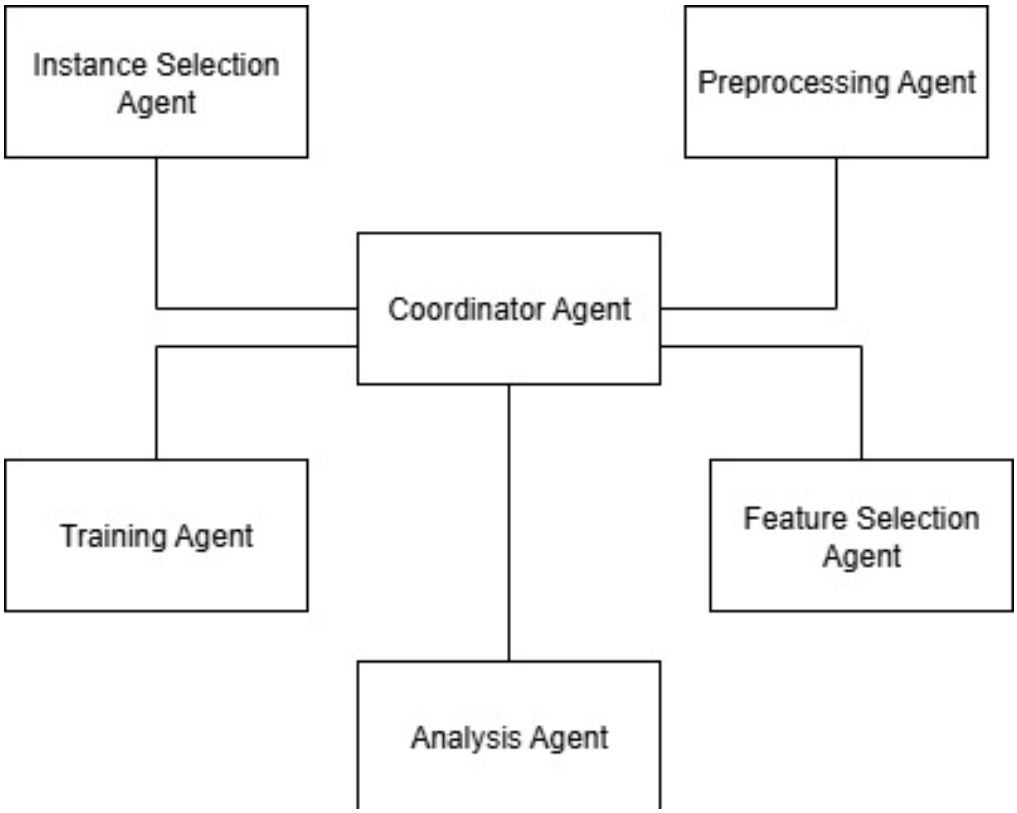

Figura 3. Organización de los agentes

- **CoordinatorAgent**: interpreta las órdenes del usuario en lenguaje natural y decide a que agente especializado derivar cada petición.

- **AnalysisAgent**: Obtiene estadísticas descriptivas del *dataset* y realiza análisis exploratorio.

- **PreprocessingAgent**: Aplica transformaciones sobre los datos, como discretización o codificación.

- **InstanceSelectionAgent**: Opera sobre las filas (instancias) reduciendo de esta forma el *dataset*.

- **FeatureSelectionAgent**: Reduce el número de variables manteniendo las más informativas o eliminado redundancias.

- **TrainingAgent**: Configura y entrena modelos supervisados a partir de los datos preparados y calcula las métricas de rendimiento del modelo entrenado.

Capítulo 9. Diseño del sistema

El diseño del sistema marca la transición natural entre la fase de análisis y la implementación. El objetivo es ofrecer una visión clara y ordenada de la estructura interna del asistente: sus módulos principales, la organización de las clases y las decisiones de diseño que condicionaron la implementación final.

9.1. Arquitectura general

El sistema se plantea como una **arquitectura multiagente de carácter modular**, organizada en torno a un agente coordinador que interpreta las solicitudes del usuario y delega su ejecución en agentes especializados.

Los componentes principales de esta arquitectura son:

- **CoordinatorAgent**: núcleo del sistema. Recibe instrucciones en lenguaje natural y decide qué agente especializado debe actuar.

- **Agentes especializados**: responsables de cada fase del pipeline (análisis exploratorio, preprocesamiento, selección, entrenamiento y evaluación).

- **Adaptadores**: funciones usadas por los agentes que encapsulan el acceso a librerías externas, reduciendo el acoplamiento y facilitando la extensibilidad.

- **Interfaz de usuario**: se trata de un *dashboard* gráfico con *Panel*.

- **Proveedor LLM**: servicio externo que interpreta las ordenes expresadas en lenguaje natural.

- **Sistema de archivos local**: repositorio donde se almacena tanto los *dataset* como los artefactos exportados.

Figura 4. Arquitectura Multiagente Modular

9.2. Diseño modular del sistema

En la Figura 5 se puede visualizar el diagrama de paquetes correspondiente al diseño del sistema.

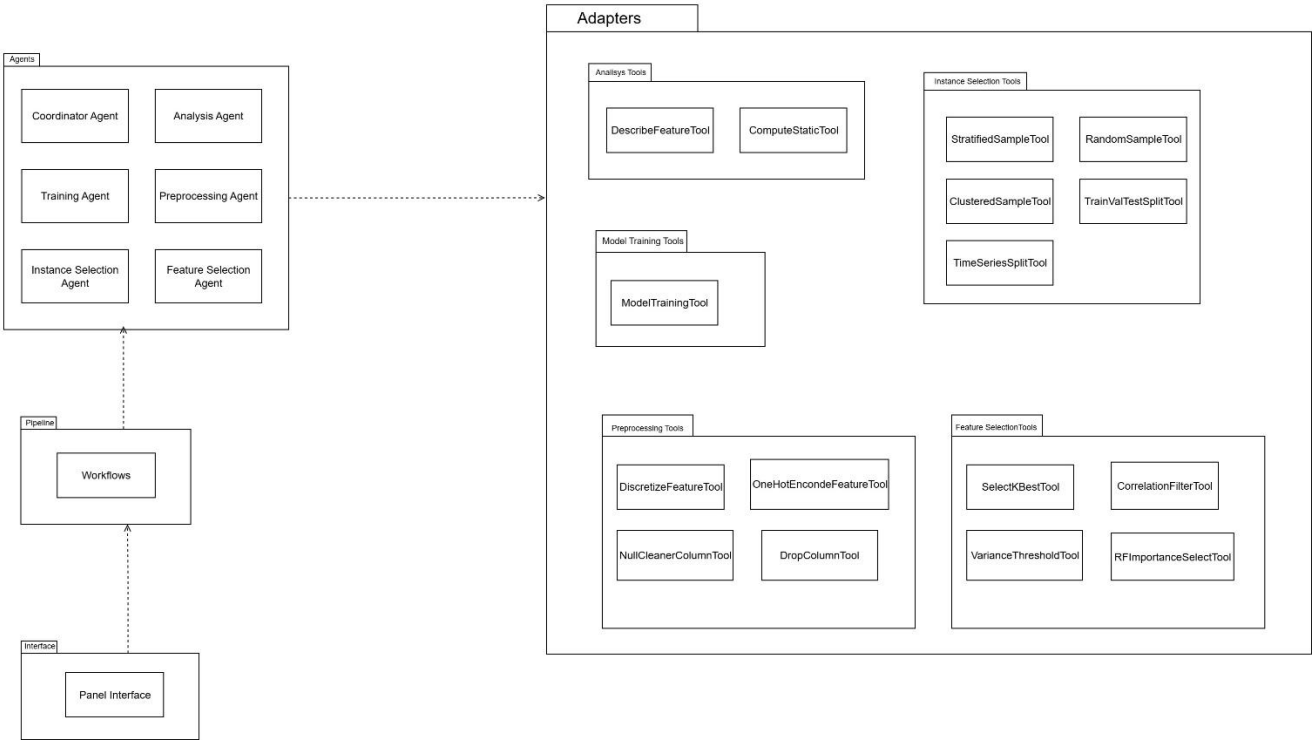

Figura 5. Diagrama de paquetes

9.2.1. Paquete *agents*

Contiene los distintos agentes que participan en el sistema.

- **CoordinatorAgent**.
    - **Entradas:** Instrucciones del usuario en lenguaje natural, historial de la conversación y *dataset* activo.

- **Funcionamiento**: Interpreta las órdenes del usuario, decide que agente especializado debe intervenir y redirige la petición mediante herramientas de delegación manual.

- **Salidas**: Respuesta generada por el agente especializado y registro en el historial para el contexto compartido.

- **AnalysisAgent**.

  - **Entradas**: *dataset* activo y una columna o conjunto de columnas sobre las que calcular estadísticas.

  - **Funcionamiento**: Invoca *tools* de análisis exploratorio, valida que las columnas existan y devuelve los resultados en un texto comprensible.

  - **Salidas**: Resumen estadístico (valores nulos, frecuencias, medias, etc.) y registro en el historial.

- **PreprocessingAgent**.

  - **Entradas**: *dataset* activo, columnas a transformar y parámetros de la operación.

  - **Funcionamiento**: activa herramientas de preprocesamiento que aplican discretización, codificación o imputación. Comprueba compatibilidad de tipos y genera transformaciones reproducibles.

  - **Salidas**: *dataset* modificado, resumen de la transformación aplicada y registro en el historial.

- **FeatureSelectionAgent**.

- **Entradas**: *dataset* activo, columnas de entrada, variable objetivo opcional y parámetros de selección.

- **Funcionamiento**: Interpreta la petición del usuario y selecciona únicamente la herramienta de selección de características adecuada.

- **Salidas**: lista de variables seleccionadas y descartadas, resumen del método utilizado y parámetros aplicados.

- **InstanceSelectionAgent**.

  - **Entradas**: *dataset* activo, variable objetivo, proporciones de muestreo o división, parámetros específicos de la técnica.

  - **Funcionamiento**: Interpreta la petición del usuario y selecciona únicamente la herramienta de selección de instancias adecuada

  - **Salidas**: subconjuntos generados con sus tamaños correspondientes, un resumen del método aplicado y los parámetros usados.

- **TrainingAgent**.

  - **Entradas**: *dataset* activo, variable objetivo definida, tipo de problema (clasificación o regresión), algoritmo seleccionado y parámetros básicos (semilla, profundidad, número de árboles, etc.).

  - **Funcionamiento**: utiliza la tool de entrenamiento para construir el modelo (*Random Forest*, *Logistic Regresion*, etc.), almacena el modelo entrenado y evalúa el modelo según unas métricas (*RMSE*, *MAE*, *F1*, etc.).

- **Salidas**: modelo entrenado, métricas, un informe con todo lo realizado y registro en el historial.

9.2.2. Paquete *adapters*

Contiene las distintas herramientas que usaran los agentes para llevar a cabo los *pipelines* solicitados.

- **AnalysisTools**.
  - **DescribeFeatureTool**.
    - **Entradas**: Nombre de la columna y, opcionalmente, número de categorías principales a mostrar.
    - **Funcionamiento**: Si la columna es numérica, calcula estadísticas básicas (media, mínimo, máximo, etc.); si es categórica, muestra las categorías más frecuentes y el porcentaje de valores nulos.
    - **Salidas**: Resumen en texto con las estadísticas descriptivas.
  - **ComputeStatisticTool**.
    - **Entradas:** Nombre de la columna, estadística a calcular y, de forma opcional, una columna de agrupación.
    - **Funcionamiento:** Calcula la estadística solicitada sobre la columna indicada y, si se especifica la agrupación, devuelve el valor por cada grupo.
    - **Salidas:** Un valor numérico acompañado de un texto explicativo.

- **PreprocessingTools**.

    - **DiscretizeFeatureTool**.

        - **Entradas**: Nombre de la columna numérica, numero de intervalos o cortes definidos manualmente y, opcionalmente, etiquetas personalizadas para los tramos.

        - **Funcionamiento**: Transforma una variable numérica continua en categorías discretas dividiendo su rango en intervalos.

        - **Salidas**: un nuevo *dataset* con la columna discretizada o con una nueva columna de intervalos, además de un resumen en texto con los tramos creados.

    - **OneHotEncodeFeatureTool**.

        - **Entradas:** nombre de la columna categórica y parámetros opcionales como prefijo o eliminación de la primera categoría.

        - **Funcionamiento:** Convierte una variable categórica en varias columnas binarias, una por categoría.

        - **Salidas:** un *dataset* con las nuevas columnas codificadas y un mensaje indicando todo lo que se ha realizado.

    - **NullCleanerColumnTool**

        - **Entradas**: nombre de la columna a limpiar, el modo de limpiar (imputación o borrado), la estrategia a seguir (media, mediana, cero) y otros parámetros opcionales.

- **Funcionamiento**: Realiza una limpieza de valores nulos en la columna seleccionada.

- **Salidas**: El dataset con la columna limpiada y un mensaje indicando que se ha realizado.

- **DropColumnTool**.

  - **Entradas**: una o varias columnas a eliminar.

  - **Funcionamiento**: Elimina una o varias columnas indicadas.

  - **Salidas**: El *dataset* con la/s columna/s eliminada/s y un mensaje indicando que se ha realizado.

- **InstanceSelectionTools**.

  - **StratifiedSampleTool**.

    - **Entradas**: columna objetivo para estratificar, tamaño de la muestra y semilla de aleatoriedad.

    - **Funcionamiento**: selecciona aleatoriamente un subconjunto del *dataset* respetando la misma proporción de clases que en el conjunto original.

    - **Salidas**: un *dataset* reducido almacenado en el disco y un informe con el tamaño final y la distribución de las clases.

  - **RandomSampleTool**.

- **Entradas**: tamaño de la muestra y semilla de aleatoriedad.
- **Funcionamiento**: selecciona filas al azar del *dataset* sin tener en cuenta la proporción de las clases.

- **Salidas**: un *dataset* reducido y un texto indicando el tamaño resultante.

- **ClassBalancedSampleTool**.

  - **Entradas:** columna objetivo y un numero de ejemplos por clase

  - **Funcionamiento:** construye una muestra equilibrada conteniendo el mismo número de instancias por clase.

  - **Salidas:** *dataset* balanceado y un informe con el número de ejemplos seleccionados por clase.

- **ClusteredSampleTool.**

  - **Entradas**: número de clústeres, numero de ejemplos representativos por *cluster*, semilla y, opcionalmente, las columnas a utilizar.

  - **Funcionamiento**: Reduce el *dataset* sin perder la esencia de la información, dividiendo los datos en los clústeres (grupos) pedidos, calculando el punto medio que resume los datos de esos clústeres y seleccionando las filas que están más cerca de ese punto ya que se consideran las más representativas del clúster.

  - **Salidas**: *dataset* reducido con los ejemplos más representativos y un texto explicativo con el número de *clústeres* y representantes por *cluster*.

- **TrainValTestSplitTool**

  - **Entradas:** proporciones para *train*, *validación* y *test*, columna objetivo y semilla.

  - **Funcionamiento:** divide el *dataset* en tres subconjuntos. Si se indica una variable objetivo y se activa la opción, mantiene las proporciones de clase en cada partición.

  - **Salidas:** tres ficheros (*train, val, test*) y un resumen con los tamaños de cada conjunto.

- **TimeSeriesSplitTool.**

  - **Entradas**: columna temporal, tamaño relativo y validación.

  - **Funcionamiento**: ordena el *dataset* cronológicamente y lo divide en entrenamiento, validación y prueba.

  - **Salidas**: tres ficheros (*train, val, test*) y un informe de texto con las fechas y tamaños resultantes.

- **FeatureSelectionTools**.

  - **SelectKBestTool**.

    - **Entradas**: la variable objetivo y el número de variables a conservar.

    - **Funcionamiento**: calcula que variables están más relacionadas con la variable objetivo y se queda con las *k* más importantes.

- **Salidas**: un nuevo *dataset* reducido solo a esas columnas, junto con un listado de cuáles han sido seleccionadas.

- **VarianceThresholdTool**.

  - **Entradas**: un valor umbral que indica la variabilidad mínima que debe tener una columna para mantenerse.

  - **Funcionamiento**: elimina columnas que practicante no cambian.

  - **Salidas**: un listado ordenado de las variables más influyente y un *dataset* filtrado con esas columnas.

- **RFImportanceSelectTool**

  - **Entradas**: El nombre de la variable objetivo y el número de variables a conservar o un nivel mínimo de importancia.

  - **Funcionamiento**: entrena un modelo rápido de bosque de árboles y usa ese modelo para ver que columnas influyen más en la predicción. Conserva solo las más relevantes.

  - **Salidas**: Un listado ordenado de las variables más influyentes y un *dataset* filtrado con esas columnas.

- **CorrelationFilterTool**

  - **Entradas:** Un nivel máximo de correlación permitido entre variables.

- **Funcionamiento:** Identifica pares de columnas que están demasiado relacionadas entre si y elimina una de ellas para evitar redundancias.

- **Salidas:** Un *dataset* con menos columnas repetitivas y un mensaje con las que fueron descartadas.

- **ModelTrainingTools**.
  - **ModelTrainingTool**.
    - **Entradas**: tipo de problema (clasificación o regresión), variable objetivo, algoritmo elegido (*Random Forest, SVM*, etc.), proporción de test y parámetros opcionales de cada modelo.

    - **Funcionamiento**: Realiza una separación de entrenamiento y test, construye el modelo especificado, lo entrena con los datos de entrenamiento y evalúa su rendimiento con el conjunto de test. Según el tipo de problema, calcula métricas adecuadas.

    - **Salidas**: Un modelo entrenado, un fichero JSON con las métricas de evaluación y un informe de texto con los resultados principales.

9.2.3. Paquete *pipelines*

Define la creación de los pipelines reproducibles que posteriormente se ejecutaran en el sistema.

- **Entradas**: *dataset* activo y mensaje del usuario.

- **Funcionamiento**: construye el *CoordinatorAgent*, asigna *dataset* a las Tools y crea una *Crew* que ejecuta la instrucción solicitada. Registra los mensajes en el historial del contexto compartido.

- **Salidas**: Respuesta en texto lista para la interfaz y ficheros de salida (modelos, métricas)

9.2.4. Paquete *interface*

Contiene la interfaz gráfica donde se comunicará el usuario con el sistema mediante lenguaje natural.

- **Entradas**: *dataset* en *CSV* y mensajes en lenguaje natural.

- **Funcionamiento**: permite cargar *datasets*, muestra una previsualización de los datos, ofrece indicadores básicos (número de filas/columnas) y gestiona la conversación con el asistente. Cada mensaje se envía al pipeline, se actualiza el histórico y se muestran los resultados.

- **Salidas**: chat con el asistente, tablas de resultados, métricas y notificaciones.

9.3. Definición de los *prompts*

En el diseño del sistema, uno de los aspectos mas importantes y complejos es la definición de los *prompts* [32] que guían tanto el comportamiento de los agentes como el de las tareas asociadas.

Un *prompt* se puede definir como "*una instrucción, pregunta o un texto que se utiliza para interactuar con sistemas de inteligencia artificial*" [32].

En *CrewAI*, estos *prompts* permiten establecer el rol esperado, marcan límites de actuación, condicionan el estilo de respuesta y fijan los criterios de ejecución. Su redacción se mantendrá en la misma línea con los requisitos funcionales (ver Capítulo 7).

9.3.1. Agente coordinador

El coordinador tendrá como rol (*role*) el de orquestador, con un *backstory* cuya misión es decidir si la instrucción del usuario corresponde a una conversación trivial (fuera del contexto de ML) o a una instrucción técnica. Su *goal* debe enfatizar que solo debe delegar en un agente especializado cada vez, evitando que se activen varios al mismo tiempo. Esta decisión es tomada para garantizar un enfoque conservador, donde se priorizará la flexibilidad del sistema y disminuir la capacidad de cometer errores por delegación.

9.3.2. Agente de análisis

Este agente tiene como misión aplicar solo las herramientas necesarias para calcular estadísticas o distribuciones de las variables. Su *goal* debe permitir pedir aclaraciones al usuario si faltan parámetros, y su *backstory* se define como un analista experto en datos exploratorios. De este modo, se asegura que la fase de análisis exploratorio proporcione información clara y relevante al usuario.

9.3.3. Agente de preprocesamiento

El rol de este agente queda definido como especialista en preparar los datos para el modelado. Su *goal* debe limitar las acciones en imputación, discretización y codificación. En el *backstory* se indicará que siempre deberá informar las columnas que se modifican, para que el usuario conozca las transformaciones que se han realizado.

9.3.4. Agente de selección de características

El rol de este agente debe ser claro: seleccionar las variables mas relevantes usando diferentes técnicas. Al igual que el anterior, deberá informar que atributos se conservan y cuales se eliminan. Finalmente, el *backstory* se definirá como un especialista en este tipo de técnicas.

9.3.5. Agente de selección de instancias

Su *goal* es aplicar métodos de muestreo o realizar particiones reproducibles. El *backstory* deberá ser el de un experto en mantener la representatividad e informar sobre el tamaño de los subconjuntos generados.

9.3.6. Agente de entrenamiento y evaluación del modelo

Este agente tiene como rol entrenar y evaluar los modelos supervisados. Su *goal* le obliga a validar los parámetros necesarios para posteriormente realizar el entrenamiento del modelo solicitado y finalmente retornar las métricas mas importantes. El *backstory* se definirá como un especialista en ML clásico, capaz de evaluar problemas de clasificación y de regresión.

9.3.7. Tareas de los agentes

Las tareas serán un elemento muy importante para obtener los resultados buscados. Para su definición, se indicará el historial de la conversación para obtener el contexto, la solicitud del usuario para conocer realmente que se esta pidiendo y el uso de una única herramienta para poder resolver esa solicitud.

9.4. Diseño de agentes y comunicación

El coordinador actúa como intermediario entre el usuario y los agentes especializados. El funcionamiento es el siguiente:

1. El usuario escribe una instrucción en lenguaje natural.

2. El coordinador recibe la instrucción y mediante el LLM asociado, invoca a una *tool* de delegación asociada.

3. La *tool* de delegación invoca al agente especializado correspondiente.

4. Se activa el agente especializado y realiza la instrucción pedida por el usuario, haciendo uso de sus *tools*.

5. El agente especializado retorna sus resultados.

6. El coordinador recoge la salida del agente especializado y se lo devuelve al usuario.

En la Figura 6 se ilustra el proceso de manera grafica.

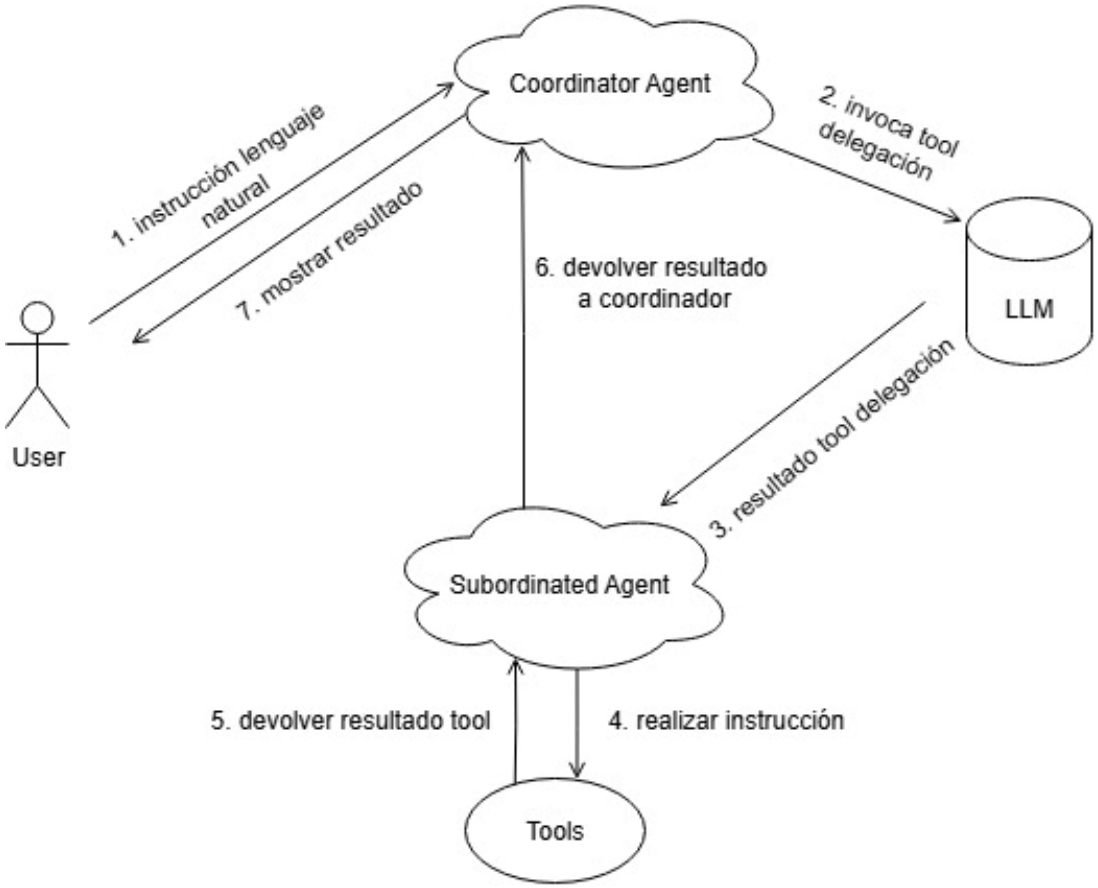

Figura 6. Diagrama de comunicación del sistema

9.5. Decisiones de diseño y limitaciones

En un inicio, se apostó por el modelo jerárquico de *CrewAI*, con un coordinador encargado de delegar dinámicamente las tareas en agentes subordinados especializados. Este enfoque ofrecía una gran flexibilidad ya que cada petición del usuario podía ser interpretada y enviada al agente más adecuado, sin necesidad de forzar un flujo predefinido, es decir, justo lo que se buscaba en este TFG.

Sin embargo, durante la implementación y las pruebas se identificaron limitaciones relevantes. En determinados casos la delegación no siempre se producía como se esperaba, en ocasiones se activaban más agentes de los necesarios y la configuración resultaba demasiado compleja y poco transparente.

Ante esta dificultad, se optó por una solución más controlada: implementar herramientas de delegación manualmente, de forma que el coordinador siempre derivase cada petición al agente especializado correspondiente de manera correcta.

Esta decisión permitió mantener el espíritu multiagente del diseño original, pero con una lógica de delegación más rígida y, al mismo tiempo, más predecible y fiable. La contrapartida fue perder parte de la flexibilidad prometida por el modelo jerárquico de *CrewAI*, aunque el sistema ganó en claridad y consistencia.

Por otro lado, en cuanto a la interfaz del usuario, se valoraron dos opciones principales: *Panel* y *Streamlit*, ambos frameworks en Python diseñados para construir *dashboard* interactivos.

Streamlit [33] tiene una gran ventaja de partida, ya que es muy popular y su curva de aprendizaje es casi inmediata. Aun así, se optó por Panel debido a que tiene mayor flexibilidad en la disposición de componentes, permitiendo diseñar interfaces más personalizadas. Además, tiene una compatibilidad superior en arquitecturas modulares ya que encaja mejor en proyectos con múltiples módulos, como es el caso de este TFG.

Capítulo 10. Implementación

La fase de implementación es, en cierto modo, el momento en que todo lo planeado empieza a tomar forma. Lo que antes eran diagramas y decisiones de diseño se transforma aquí en un sistema real.

En este capítulo se explica el entorno de programación empleado, como se organizan los paquetes y módulos, y también esos detalles prácticos que, aunque a veces parecen menores, terminan marcando la diferencia en el desarrollo.

10.1. Entorno de desarrollo

- **Lenguaje**: *Python 3.10*

- **Framework multiagente**: *CrewAI.*

- **Bibliotecas científicas**: *pandas*, *NumPy*, *SciPy*.

- **Bibliotecas de machine learning:** *scikit-learn 1.3.*

- **Persistencia de modelos:** *joblib*.

- **Interfaz gráfica:** *Panel*.

- **Control de versiones**: *Git* y *GitHub*

10.2. Organización del proyecto

El código se encuentra en el directorio *src/tfg_ml*, estructurado en distintos paquetes, de acuerdo con el diseño (ver Capítulo 9), reflejando la modularidad del sistema:

- *agents*/: Todos los distintos agentes que componen el sistema.

- *adapters/:* Son las herramientas o *tools* que usan los distintos agentes para utilizar las librerías externas.

- *pipelines*/: Definición de los pipelines.

- *interface*/: Donde se encuentra la interfaz de usuario.

10.3. Explicación del código

En este punto se abordará la explicación del código de cada archivo que dispone el sistema. Con motivo de no extender la memoria del TFG demasiado, el código completo se ubicará en el Anexo – Manual de Código.

10.3.1. Implementación de los *prompts*.

Los *prompts* diseñados en el Capitulo 10 han sido implementados mediante la librería *CrewAI*, que permite instanciar agentes a través de la definición de su rol (*role*), objetivo (*goal*) y contexto narrativo (*backstory*).

A continuación, se muestra un ejemplo representativo del *AnalysisAgent*, donde se puede observar la definición de dichos *prompts*.

```
return Agent(
        role="Exploratory Data Analyst",
        goal=(
            "Understand the user's request and call only the necessary tools. "
             "When calling a tool, fill the structured parameters according to the tool's schema. "
             "Ask a brief clarifying question if the request lacks required parameters (e.g., missing 'column')."
        ),
        backstory=(
            "You are precise and concise. You don't run unnecessary tools. "
            "You prefer structured inputs and return readable results."
        ),
        tools=[DescribeFeatureTool(), ComputeStatisticTool()],
        verbose=True,
        llm=llm,
        max_iter=3,
        max_execution_time=30,
    )
```

Al igual que el fragmento anterior, los demás agentes se codificaron siguiendo la misma estructura.

10.3.2. agents/coordinator_agent.py

Este módulo implementa el **agente coordinador,** la pieza central que orquesta toda la conversación. Al iniciarse, configura el modelo de lenguaje con un rol y un objetivo muy claros: gestionar las peticiones del usuario y decidir qué hacer con cada una. Para ello, carga un conjunto de herramientas de delegación. Estas no procesan datos por sí mismas, sino que actúan como pasarelas, forzando la invocación del subagente más adecuado en cada caso.

Cuando la interfaz le envía un mensaje, lo primero que hace el coordinador es un filtrado básico. Si detecta preguntas que no tienen nada que ver con *machine learning*, responde el mismo con un mensaje breve, en el idioma del usuario, y no activa más componentes. Pero si la petición sí está relacionada con ML (por ejemplo: "*describe Price column*"), el coordinador interpreta la intención a partir de las palabras clave y del contexto reciente. Entonces dispara una sola herramienta de delegación: análisis, preprocesamiento, selección de características, selección de instancias o entrenamiento/evaluación.

Esa herramienta se encarga de construir la llamada al sub-agente especializado, pasándole el *dataset* activo y los parámetros relevantes (bien porque el usuario los indico en el chat, bien porque el coordinador los dedujo del contexto). Luego espera su respuesta. Cuando el sub-agente devuelve el resultado, el coordinador lo entrega tal cual, al usuario, sin reescribir nada, para preservar la trazabilidad. Además, registra en el contexto compartido que acción se ejecutó, sobre que columnas o variables y con que resumen de resultados.
Si ocurre un error, el coordinador captura la excepción y responde de manera orientativa.

De esta forma, el coordinador convierte lo que empieza como una conversación en acciones concretas sobre el pipeline. Todo ello bajo un control de delegación manual, que compensa las limitaciones del modelo jerárquico nativo de *CrewAI*.

10.3.3. agents/analysis_agent.py

Este agente es el especialista en análisis exploratorio de datos (EDA), una especie de lupa inicial para entender mejor los *datasets*. Este agente interpreta preguntas típicas como "*descríbeme la variable A*" o "*dame estadísticas por categoría*" e invoca a las *Tools* definidas en *adapters/analysis_tools.py*

Cuando recibe una petición del coordinador, lo primero que hace es comprobar que los datos estén cargados y que la columna solicitada exista. Si detecta que falta algún detalle, pide una aclaración mínima antes de continuar. Con los parámetros ya definidos activa herramientas como **DescribeFeatureTool** o **ComputeStatisticTool** de las que se hablará más adelante.

El agente devuelve el resultado casi tal cual, aunque puede añadir una pequeña frase de contexto si lo cree necesario.

10.3.4. agents/preprocessing_agent.py

Este agente es el encargado del preprocesamiento, la etapa donde los datos se preparan para que los modelos puedan trabajar con ellos sin problemas.

Cuando recibe una orden del coordinador, del estilo "discretiza la edad en 5 *bins*", lo primero que hace es validar la compatibilidad. Por ejemplo, evita aplicar un *one-hot encoding* sobre una columna puramente numérica cuando no tiene sentido. Una vez verificado, invoca la herramienta adecuada.

La respuesta que devuelve incluye información concreta en función de la herramienta invocada y finalmente, se la pasara al coordinador.

10.3.5. agents/feature_selection_agent.py

Este agente actúa como el especialista en selección de características, el encargado de decidir que variables merecen quedarse en el modelo y cuales conviene descartar. Para

ello recurre a un conjunto de *Tools* de filtrado y ranking: desde los k-mejores por *chi-cuadrado* o *F-test*, hasta criterios como el umbral de varianza, la importancia medida con *Random Forest* o la correlación con la variable objetivo.

El funcionamiento es similar a los anteriores. El agente recibe una instrucción del coordinador, valida los parámetros y ejecuta la *tool* correspondiente.

A nivel práctico, el agente también actualiza el *DataFrame* activo, de manera que los siguientes pasos del pipeline trabajen ya con ese subconjunto optimo de variables.

10.3.6. agents/instance_selection_agent

Este agente es el responsable de la selección y partición de instancias, una pieza clave para garantizar que los experimentos sean consistentes y comparables. Su repertorio abarca desde submuestreos hasta particiones reproducibles en *train/val/test*, incluyendo variantes estratificadas cuando la distribución de clases lo exige. Todo esto se verá más en detalle más adelante.

10.3.7. agents/model_training_agent.py

Este agente es el encargado del entrenamiento y la evaluación de modelos supervisados, la fase donde el pipeline se convierte en resultados tangibles.

Recibe una petición, selecciona el algoritmo apropiado, ajusta unos hiperparámetros por defecto y entrena el modelo mediante su Tool de entrenamiento integrada con *scikit-learn*. Tras esto, calcula las métricas más adecuadas al caso: ***Accuracy*** y ***F1*** para clasificación; ***MAE***, ***RMSE*** y $\boldsymbol{R^2}$ para regresión. Además, se encarga de persistir el artefacto y guardar un archivo con las métricas obtenidas.

10.3.8. adapters/analysis_tools.py

Este archivo reúne las herramientas de análisis exploratorio de datos (EDA). Como comentamos anteriormente, su propósito es ayudar a entender que hay realmente en los datos antes de dar el siguiente paso.

En primer lugar, cada herramienta tiene clases con validación de entradas mediante *Pydantic*. La herramienta ***DescribeFeatureTool***, puede señalar si una variable es numérica o categórica, cuantos valores faltan, cual es el promedio, el valor más frecuente o incluso, en el caso de columnas de texto, listar las categorías que más se repiten.

En cambio, la herramienta ***ComputeStatisticTool***, permite calcular estadísticas concretas y, si se pide, hacerlo por grupos.

10.3.9. adapters/preprocessing_tools.py

Este archivo contiene las operaciones de preprocesamiento, esas transformaciones que dejan el *dataset* en un estado mucho más manejable y coherente.
Cada herramienta tiene también clases con validación de entradas mediante *Pydantic*.

La herramienta ***DiscretizeFeatureTool*** convierte valores continuos, como la edad, en grupos o intervalos fáciles de interpretar. Es como agrupar datos sueltos en cajones ordenados para poder compararlos mejor.

La herramienta ***OneHotEncodeTool*** transforma una columna de texto con categorías en varias columnas binarias, una por categoría. Así, los algoritmos reciben la información en un formato que realmente entienden, como si tradujésemos palabras en señales de encendido y apagado.

La herramienta ***NullCleanerColumnTool*** limpia los valores nulos de una columna.

La herramienta ***DropColumnTool*** es capaz de eliminar una o varias columnas del conjunto de datos.

10.3.10. adapters/feature_selection_tools.py

Este archivo reúne las herramientas de selección de características, cuyo objetivo es quedarse con las columnas del *dataset* que realmente aportan valor para el objetivo final y dejar fuera aquellas que añaden poco o nada.

La validación de entradas se realiza mediante *Pydantic*, como las herramientas anteriores.

La herramienta ***SelectKBestTool*** selecciona un numero fijo de columnas consideradas las mejores, en función de una puntuación calculada automáticamente.

La herramienta ***VarianceThresholdTool*** elimina aquellas columnas que apenas cambian y que, por tanto, aportan escasa información.

La herramienta ***RFImportanceSelectTool*** ordena las columnas según su importancia estimada por un modelo sencillo, lo que sirve como una guía práctica para decidir.

La herramienta ***CorrelationFilterTool*** retira columnas muy redundantes, porque están excesivamente correlacionadas con otras.

10.3.11. adapters/instance_selection_tools.py

Este archivo se encarga de las operaciones sobre las filas del *dataset*, es decir, sobre instancias.

En primer lugar, la validación de entradas también se realiza mediante *Pydantic*.

La herramienta ***RandomSampleTool*** selecciona una muestra aleatoria de las filas del *dataset*. Es muy útil para reducir el tamaño de los datos y así trabajar de manera más ágil.

La herramienta ***StratifiedSampleTool*** es parecida a la anterior con la diferencia que mantiene las proporciones de las clases. Por ejemplo, si el *dataset* tiene un 60% de hombres y un 40% de mujeres, la muestra respetara esa misma proporción. Esto es clave para evitar sesgos cuando se trabaja con categorías desbalanceadas.

La herramienta ***ClassBalancedSampleTool*** construye una muestra equilibrada eligiendo el mismo número de filas por clase o calculándolo a partir de un máximo total.

La herramienta ***ClusteredSampleTool*** selecciona una muestra diversa aplicando agrupación automática sobre las columnas numéricas con ***KMeans***. El usuario indica el número de grupos y, de cada uno, toma el ejemplo más representativo. Así se obtiene una muestra pequeña, pero que cubre bien la variedad del *dataset*.

La herramienta ***TrainValTestSplitTool*** parte el *dataset* de forma reproducible en entrenamiento, validación y prueba.

La herramienta ***TimeSeriesSplitTool*** realiza la división respetando el orden temporal de una columna de fecha/hora.

10.3.12. adapters/model_training_tools.py

Este archivo implementa el entrenamiento y la evaluación de modelos de aprendizaje supervisado, la etapa final donde los datos se convierten en resultados medibles.
En primer lugar, se realiza la validación de entrada mediante *Pydantic*, recogiendo tanto los datos como la variable objetivo a predecir.

A continuación, separa los datos en dos subconjuntos (entrenamiento y prueba).

Después construye el modelo solicitado dentro de los soportados: ***Random Forest*** (para clasificación o regresión), ***Regresion Logistica*** y ***SVM*** (solo clasificación), ***Regresion Lineal*** (solo regresión) y ***XGBoost***.

Una vez entrenado el modelo, calcula las métricas de calidad sobre el conjunto de prueba. Finalmente, guarda los artefactos, el modelo serializado y un JSON con las métricas.

10.3.13. pipelines/workflows.py

Este archivo encapsula la orquestación *end-to-end* de una interacción. Su función es recibir el mensaje del usuario junto con el *dataset* activo, construir el *CoordinatorAgent*, crear una *Crew* con la *task* que incluye el *prompt* del usuario y el contexto reciente de la conversación y ejecutar la interacción completa.

De esta forma se mantiene una separación clara entre la lógica de negocio y la presentación.

10.3.14. interface/ui_panel

Este archivo define la capa de presentación de la aplicación, que en este caso es *Panel*. Al iniciarse, carga extensiones como *Tabulator* para trabajar cómodamente con tablas y construye una vista compuesta principalmente por un elemento para subir el *dataset*, una previsualización del *dataset* y un chat interactivo.

El flujo es sencillo: cuando un usuario sube un *dataset*, la interfaz lee el CSV en un *DataFrame* y lo muestra como previsualización para que el usuario pueda conocer los datos.

Cada mensaje del chat activa el pipeline en pipelines/workflows.py y muestra la respuesta final del coordinador.

10.3.15. context.py

Este módulo implementa el contexto compartido, el punto único de memoria en proceso. Su misión es conservar todo lo que va ocurriendo durante la sesión, para que el sistema recuerde lo anterior y pueda tomar decisiones con un contexto ya definido.

Capítulo 11. Pruebas

El objetivo de la fase de pruebas es verificar que el sistema creado cumpla con los requerimientos funcionales y no funcionales establecidos en el proyecto (ver Capítulo 7).

Para ello, se han utilizado dos enfoques complementarios: las pruebas de caja blanca y las pruebas de caja negra. Las pruebas de caja blanca tienen como objetivo comprobar el funcionamiento interno de cada módulo tanto de manera independiente como integrada, mientras que las pruebas de caja negra se enfocan en explorar los casos de uso en su totalidad imitando una ejecución real de un usuario.

11.1. Pruebas de caja blanca

11.1.1. Diseño de las pruebas

Las pruebas unitarias se centraron en las Tools de los adaptadores, mientras que las de integración comprobaron la coordinación de los agentes a través del *CoordinatorAgent*. El diseño puede resumirse en la tabla 15.

Prueba	Descripción	Acción esperada
P1	*DescribeFeatureTool* sobre columna numérica	Devuelve estadísticas básicas iguales a las de *pandas*
P2	*ComputeStatisticTool* con agrupación	Calcula medias por grupos de forma coherente
P3	*DiscretizeFeatureTool* con 5 intervalos	Crea columna nueva con 5 tramos bien definidos.
P4	*OneHotEncodeFeatureTool* sobre columna categórica	Genera tantas columnas *dummy* como categorías existan.
P5	*NullCleanerColumnTool* mediante imputación con mediana	Imputa los valores nulos con la mediana de la columna.

P6	*DropColumnTool*	Elimina una columna del conjunto de datos
P7	*SelectKBestTool* con k=10	Devuelve exactamente 10 columnas, ordenadas por relevancia.
P8	*StratifiedSampleTool*	Genera muestra reducida manteniendo proporciones de clases.
P9	*TrainValTestSplitTool*	Divide *dataset* en *train/val/test* con tamaños correctos.
P10	*ModelTrainingTool* (Random Forest, clasificación)	Entrena modelo, calcula métricas y guarda artefactos.
P11	Integración: *CoordinatorAgent* con petición "discretiza edad"	Activa solo *PreprocessingAgent* y devuelve nueva columna.
P12	Integración: error al pedir columna inexistente	Devuelve mensaje orientativo sin bloquear ejecución.

Tabla 14. Pruebas de caja blanca

11.1.2. Ejecución de las pruebas

Los resultados obtenidos fueron los esperados en todos los casos. Estos resultados se pueden visualizar en la tabla 16.

Prueba	Descripción	Resultado
P1	*DescribeFeatureTool* sobre columna numérica	Correcto
P2	*ComputeStatisticTool* con agrupación	Correcto
P3	*DiscretizeFeatureTool* con 5 intervalos	Correcto
P4	*OneHotEncodeFeatureTool* sobre columna categórica	Correcto
P5	*NullCleanerColumnTool* mediante imputación con mediana	Correcto
P6	*DropColumnTool*	Correcto
P7	*SelectKBestTool* con k=10	Correcto
P8	*StratifiedSampleTool*	Correcto
P9	*TrainValTestSplitTool*	Correcto

P10	*ModelTrainingTool* (*Random Forest*, clasificación)	Correcto
P11	Integración: *CoordinatorAgent* con petición "discretiza edad"	Correcto
P12	Integración: error al pedir columna inexistente	Correcto

Tabla 15. Resultado Pruebas Caja Blanca

Las pruebas de caja blanca se superaron con éxito, lo que garantiza que las herramientas funcionan correctamente en solitario y que la coordinación entre agentes es estable.

11.2. Pruebas de caja negra.

11.2.1. Diseño de las pruebas.

Para simular la experiencia de un usuario final, se diseñaron pruebas *end-to-end* ejecutadas directamente desde la interfaz.
El diseño puede resumirse en la tabla 17.

Caso	Descripción	Acción esperada
CU1	Subir *dataset* CSV	Detectar delimitadores, cargar datos y mostrar primeras filas.
CU2	Pedir estadísticas sobre variable categórica	Mostrar frecuencias y valores nulos de la columna.
CU3	Aplicar preprocesamiento (discretización y *one-hot*)	Crear nuevas columnas con intervalos y *dummies*.
CU4	Seleccionar características (k=10)	Devolver listado de 10 variables más relevantes.
CU5	Entrenar modelo *Random Forest*	Entrenar, calcular métricas y mostrar informe
CU6	Exportar modelo y métricas	Guardar artefactos en *data/artifacts*
CU7	Interpretar ordenes en Lenguaje Natural	El sistema interpreta las ordenes y ejecuta las acciones correspondientes.

Tabla 16. Pruebas Caja Negra

11.2.2. Ejecución de las pruebas

Los resultados esperados fueron igualmente satisfactorios como se muestran en la tabla 18.

Caso	Descripción	Resultado
CU1	Subir *dataset* CSV	Correcto
CU2	Pedir estadísticas sobre variable categórica	Correcto
CU3	Aplicar preprocesamiento (discretización y *one-hot*)	Correcto
CU4	Seleccionar características (k=10)	Correcto
CU5	Entrenar modelo *Random Forest*	Correcto
CU6	Exportar modelo y métricas	Correcto
CU7	Interpretar ordenes en Lenguaje Natural	Correcto

Tabla 17. Resultados Pruebas Caja Negra

También se evaluaron casos negativos, como subir un *CSV* vacío, pedir operaciones sobre columnas inexistentes o entrenar sin definir la variable objetivo. En todos ellos, el sistema respondió con un mensaje explicativo y sin bloquear la aplicación.

Capítulo 12. Experimentación

La fase de experimentación tiene como objetivo comprobar que el asistente multiagente desarrollado es realmente capaz de construir pipelines de aprendizaje supervisado completos a partir de instrucciones expresadas en lenguaje natural. En otras palabras, validar que el sistema entiende lo que se le pide, lo traduce a operaciones técnicas concretas y funciona con *datasets* de distinta naturaleza, incluso cuando las peticiones del usuario varían en su forma de expresarse.

Para ello, se seleccionaron tres conjuntos de datos con características muy diferentes, pensados para poner a prueba el asistente en contextos variados:

- **AmesHousing** [34] (regresión): un *dataset* clásico de predicción de precios de viviendas.

- **Iris** [35] (clasificación multiclase): un *dataset* de flores de iris medidos en varios atributos para predecir la especie de la flor.

- **Bank Marketing** [36] (clasificación binaria desbalanceada): datos de campañas de marketing bancario, con muchas variables categóricas y gran diversidad de valores.

12.1. Materiales y configuración experimental

Como se ha comentado anteriormente, se han utilizado tres conjuntos de datos de diferentes características. En la Tabla 19 se detalla las características de cada uno de ellos.

Dataset	Nº Instancias	Nº Variables	Target
AmesHousing	2.930	82	*SalePrice*
Iris	150	5	*Species (setosa/versicolor /virginica)*
Bank Marketing	5.521	17	*y (yes/no)*

Tabla 18. Características de los datasets empleados

12.2. Resultados

En este apartado, se mostrará los distintos resultados para cada uno de los experimentos realizados.

12.2.1. Experimento 1

En este experimento se probará a realizar un pipeline de aprendizaje supervisado clásico, donde se realizarán los pasos típicos para poder entrenar un modelo de la manera más eficiente posible.

1. Cargar dataset y previsualizar los datos.

2. Realizar análisis estadístico de la variable objetivo.

3. Realizar preprocesamiento de algunos datos.

4. Reducir el dataset a través de la selección de características.

5. Realizar el entrenamiento del modelo.

Caso 1 – AmesHousing

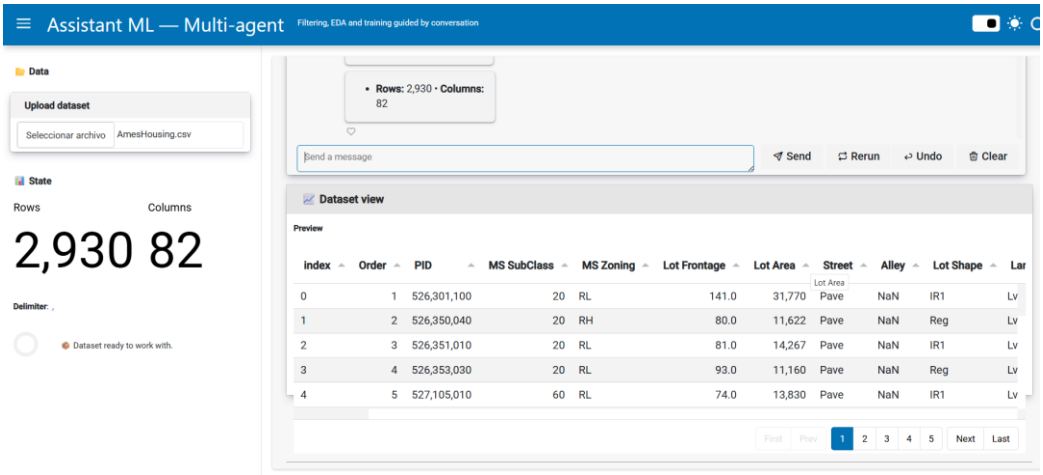

Figura 7. Previsualización de los datos (AmesHousing)

En la Figura 7 se observa la carga del *dataset* y la previsualización de los datos de este. Se observa un total de 2930 instancias y 82 variables, tal y como se comentó en la Tabla 19.

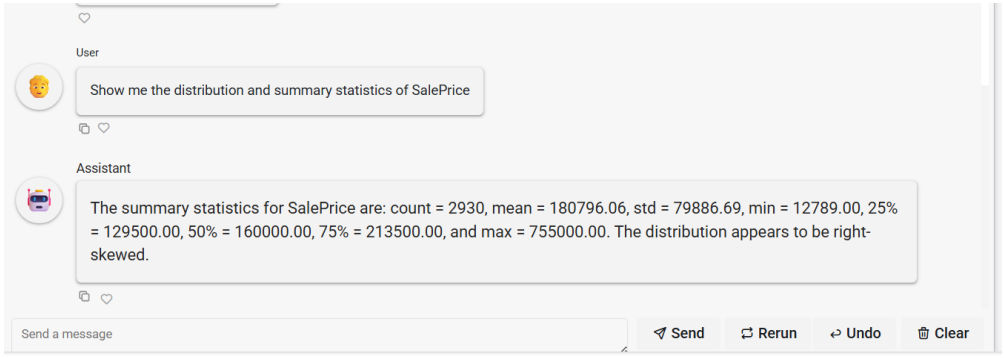

Figura 8. Evidencia EDA Experimento 1 (AmesHousing)

En la Figura 8 se observa la interacción para realizar un análisis estadístico de la variable "*SalePrice*". Con él, se descubre que la distribución de los datos parece estar sesgada a la derecha, por lo que se tendrá en cuenta para las siguientes interacciones.

Figura 9. Evidencia Preprocessing Data 1 Experimento 1 (AmesHousing)

En la figura 9 se observa cómo se crean diferentes variables numéricas para la variable "*Neighborhood*" ya que al transformarse en valores continuos será mucho más sencillo trabajar con ellos.

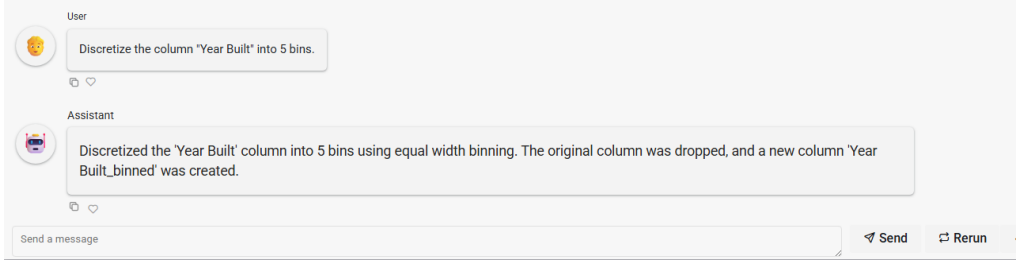

Figura 10. Evidencia Preprocessing Data 2 Experimento 1 (AmesHousing)

En la Figura 10 se observa que la variable "*Year Built*" se discretiza en 5 *bins*, es decir, se separa en 5 intervalos con el objetivo de que posteriormente los modelos puedan trabajar mejor.

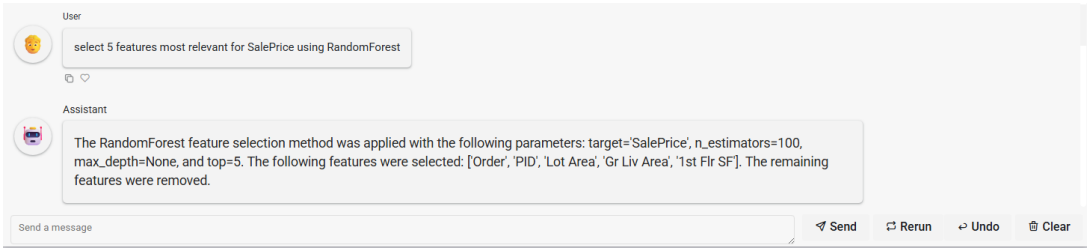

Figura 11. Evidencia Feature Selection (AmesHousing)

En la Figura 11 se observa que se ha seleccionado las 5 características más relevantes para la variable objetivo "*SalePrice*" utilizando *Random Forest* como método. Esto servirá para realizar el entrenamiento del modelo mucho más eficiente y completo.

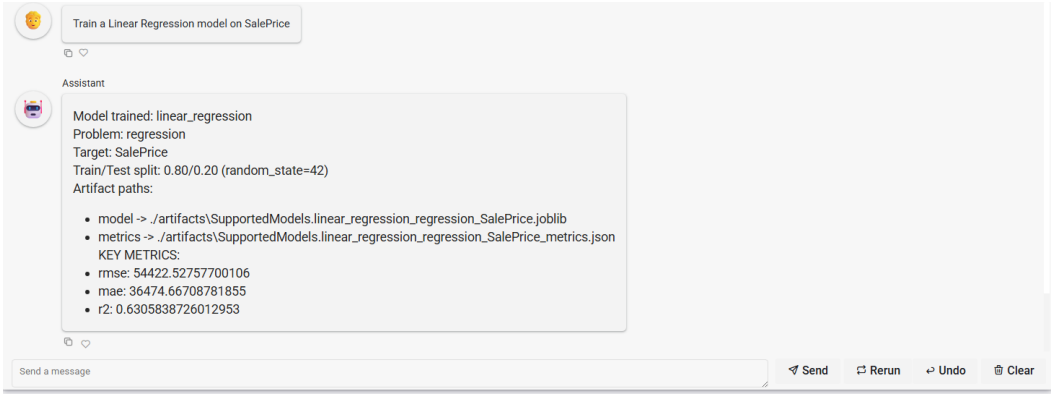

Figura 12. Evidencia Training Model Linear Regresion Experimento 1 (AmesHousing)

En la Figura 12 se detalla el entrenamiento del modelo usando *Linear Regression* con la variable objetivo que se quiere predecir (en este caso *SalePrice*). Además, se observa un mensaje donde se explica todo lo que se ha utilizado, es decir, el modelo que se ha utilizado, el tipo de problema, cuanta proporción de entrenamiento y test se ha utilizado, los artefactos generados y, por último, la evaluación del modelo con las métricas generadas.

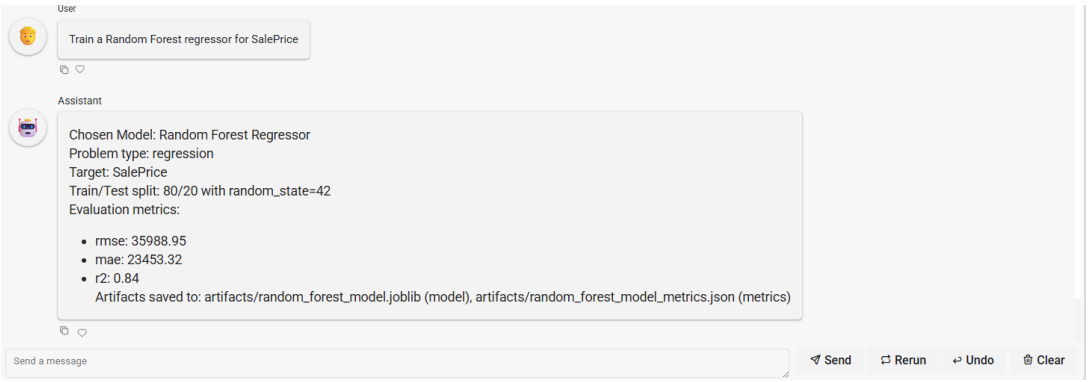

Figura 13. Evidencia Training Model Random Forest Regressor Experimento 1 (AmesHousing)

Por otro lado, en la Figura 13 se detalla el entrenamiento del modelo usando *Random Forest Regressor* con la variable objetivo. En este caso, también se observa un mensaje donde se explica todo lo que se ha utilizado.

Modelo	RMSE	MAE	R^2
Linear Regression	54422.53	36474.66	0.63
Random Forest	35988.95	23453.32	0.84

Tabla 19. Métricas de Experimento 1 (AmesHousing)

En la Tabla 19 se observa las distintas métricas obtenidas en función del modelo usado. Con estas métricas podemos llegar a la conclusión de que *Random Forest* supera con un gran margen a *Logistic Regression* para este modelo. Esto es debido a que el RMSE y MAE de *Logistic Regression* es bastante más alto que el *Random Forest* (cuanto más alto sea, más errores de precisión tiene el modelo). Además, el R^2 de *Random Forest* muestra un valor bastante alto en cuanto a la variabilidad en contraposición al de *Linear Regression*, que tiene un valor más bajo.

Caso 2 – Iris

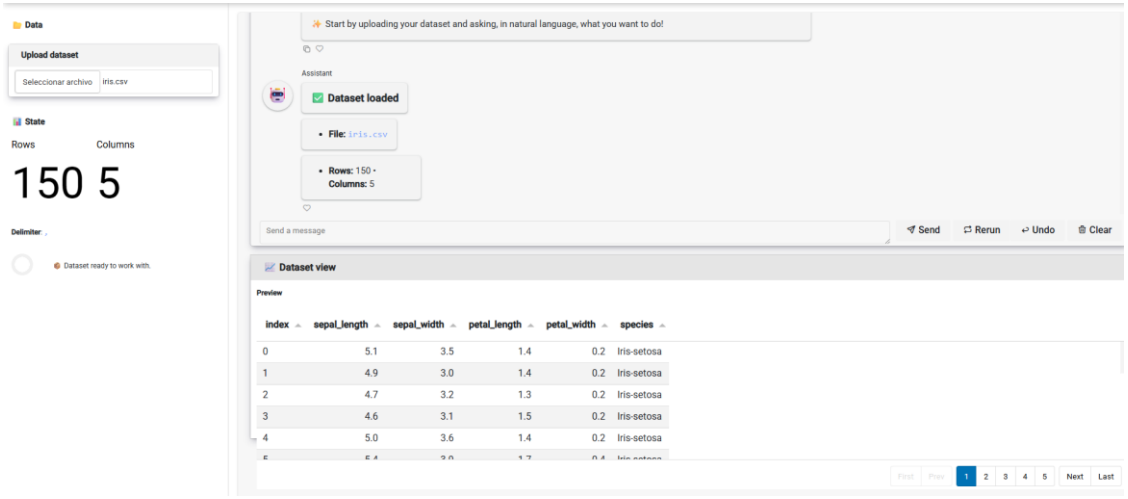

Figura 14. Evidencia Previsualización de los datos (Iris)

En la Figura 14, se observa que se ha cargado el *dataset* y una previsualización de los datos. Este *dataset* tiene 150 instancias y 5 características, tal y como se comentó en la Tabla 18.

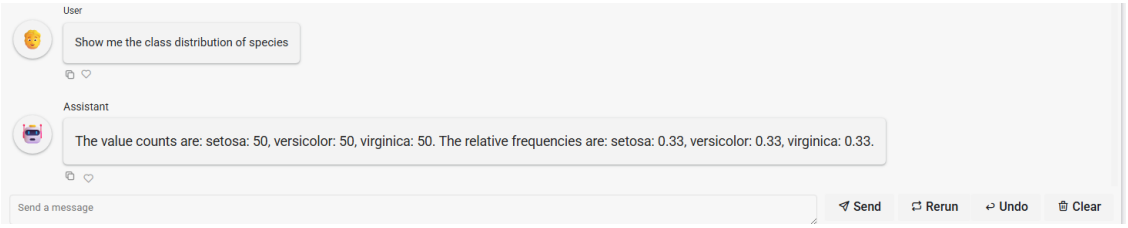

Figura 15. Evidencia EDA 1 Experimento 1 (Iris)

En la Figura 15, se muestra la distribución de las clases de la variable "*species*". Esto se realiza ya que este *dataset* es un problema de clasificación multiclase, es decir, la variable objetivo dispone de distintos valores (en este caso tres valores).

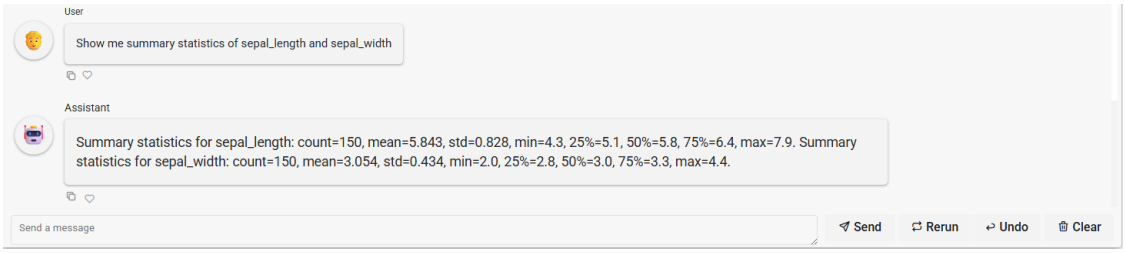

Figura 16. Evidencia EDA 2 Experimento 1 (AmesHousing)

En la Figura 16, se muestra un análisis estadístico de las características “sepal_length” y “sepal_width” con el objetivo de conocer un poco más los datos con los que se va a trabajar.

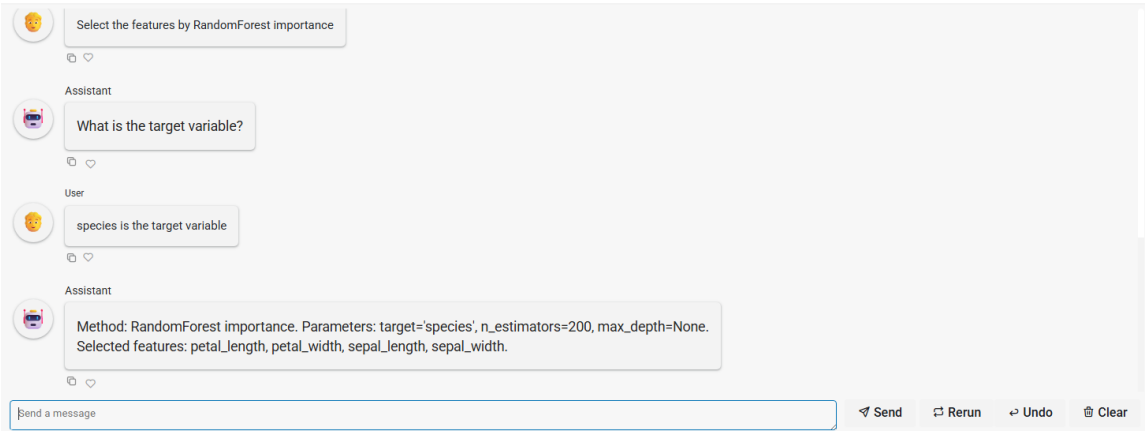

Figura 17. Evidencia Preprocessing Data Experimento 1 (Iris)

En la Figura 17, se solicita seleccionar las características según lo importantes que son para *Random Forest*. Tras esto, el sistema detecta que no se ha enviado la variable objetivo y la solicita. Una vez se ha enviado la variable objetivo (en este caso “*species*”), realiza la selección de las características. Con esto, el sistema demuestra que tiene memoria ya que no ha sido necesario volver a escribir la orden.

En los resultados, se puede observar que las 4 características restantes tienen bastante importancia para el algoritmo *Random Forest*.

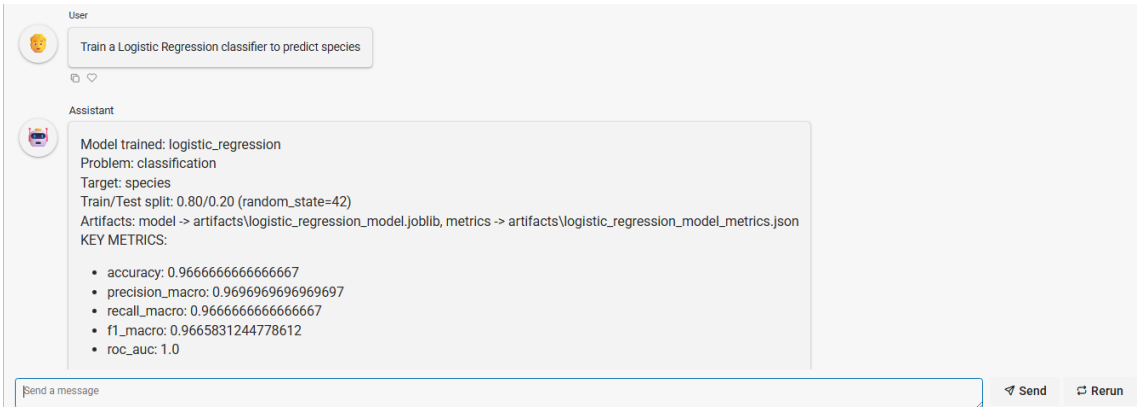

Figura 18. Evidencia Training Model Logistic Regression Experimento 1 (Iris)

En la Figura 18, se solicita entrenar el modelo usando *Logistic regression classifier* para predecir la variable "*species*". El sistema informa de todos los detalles de dicho entrenamiento.

Figura 19. Evidencia Training Model Random Forest Experimento 1 (Iris)

Al igual que en la Figura 19, en la Figura 16 se solicita el entrenamiento del modelo, pero en este caso usando *Random Forest classifier*. El sistema también informa de todos los detalles de dicho entrenamiento.

Modelo	Accuracy	Precision_macro	Recall_macro	F1_macro	Roc_auc
Logistic Regression	0.967	0.97	0.967	0.967	1.0
Random Forest	0.933	0.933	0.84	0.933	0.99

Tabla 20. Métricas de Experimento 1 (Iris)

En la Tabla 20 se observa las distintas métricas obtenidas en función del modelo usado. A diferencia del Caso 1, este problema es de clasificación por lo que las métricas usadas son diferentes.

Con estas métricas podemos llegar a la conclusión de que *Logistic Regression* supera con un ligero margen a *Random Forest* para este modelo. Aunque *Random Forest* también obtiene muy buenos resultados, presenta una mayor debilidad en detectar correctamente

los positivos (*Recall_macro*). En cuanto a la separación de clases (*Roc_auc*), ambos lo realizan muy bien, pero *Logistic Regression* llega a la separación perfecta.

En definitiva, se podrían usar cualquiera de los dos modelos porque son muy precisos.

Caso 3 – Bank

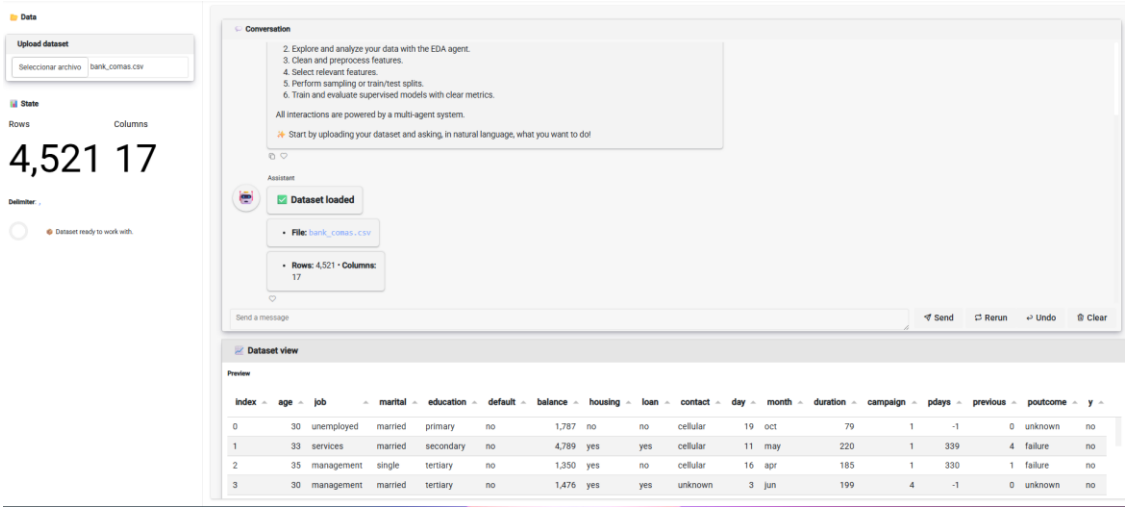

Figura 20. Evidencia Previsualización de los datos Experimento 1 (Bank)

En la Figura 20 se observa que se ha cargado correctamente el *dataset* y se muestra una previsualización de los datos. Como se comentó en la Tabla 18, este *dataset* contiene 4.521 instancias y 17 características.

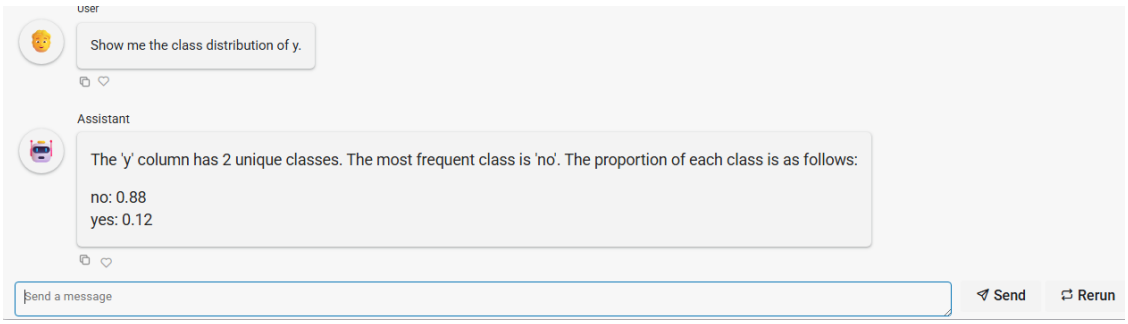

Figura 21. Evidencia EDA Experimento 1 (Bank)

En la Figura 21 se ha solicitado la distribución de clases de la variable objetivo "*y*" debido a que este *dataset* es un problema de clasificación binaria, es decir, la variable objetivo dispone de dos únicos valores.

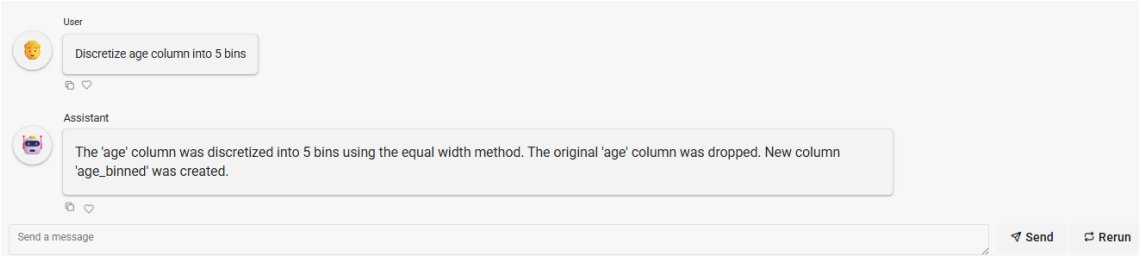

Figura 22. Evidencia Preprocessing Data Experimento 1 (Bank)

En la Figura 22, se aplica discretización para transformar la variable "*age*" en distintos intervalos.

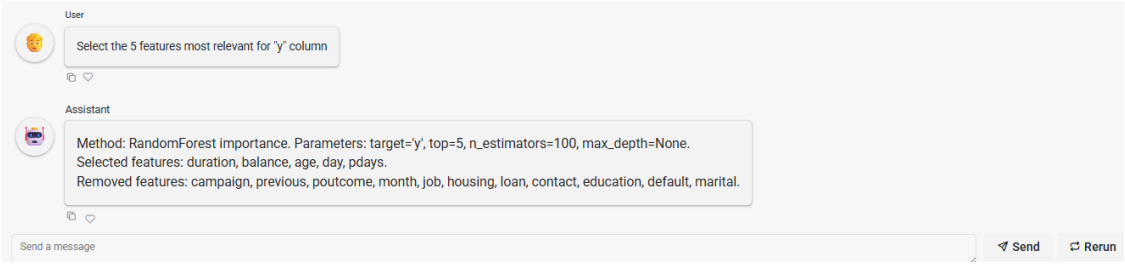

Figura 23. Evidencia Feature Selection Experimento 1 (Bank)

En la Figura 23, se realiza una selección de las 5 características más relevantes para la columna objetivo y con ello se reduce el *dataset* a ellas.

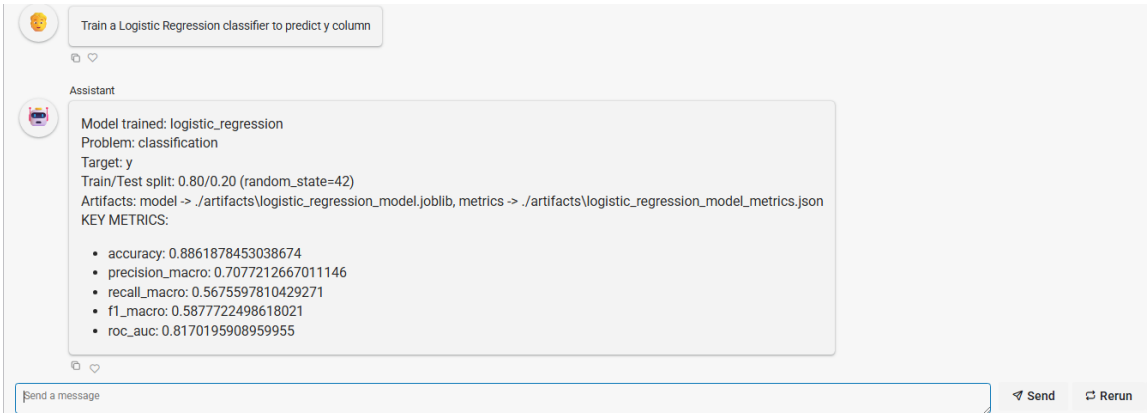

Figura 24. Evidencia Model Training Logistic Regression Experimento 1 (Bank)

En la Figura 24, se realiza el entrenamiento del modelo usando *Logistic Regression classifier* para predecir la variable "*y*". Se observa que se han generado los artefactos y las métricas asociadas al problema de clasificación.

Figura 25. Evidencia Model Training Random Forest Experimento 1 (Bank

En la Figura 25 se comprueba que se ha entrenado el modelo usando *Random Forest* para predecir la variable objetivo "*y*".

Modelo	Accuracy	Precision_macro	Recall_macro	F1_macro	Roc_auc
Logistic Regression	0.886	0.707	0.568	0.588	0.817
Random Forest	0.884	0.701	0.617	0.642	0.812

Tabla 21. Métricas de Experimento 1 (Bank)

En la Tabla 21 se observa la comparación entre las distintas métricas del modelo entrenado en función del modelo usado. Se puede comprobar que ambos modelos tienen un rendimiento muy parecido en *Accuracy* y *Roc_auc*, siendo este muy correcto. El *Random Forest* destaca por un mejor *Recall* y *F1_macro*, logrando ser más preciso que *Logistic Regression*. No obstante, *Logistic Regression* ofrece una ligera mejora en cuanto a la métrica *Precision_macro*, lo que indica que se equivoca menos cuando afirma que un caso es positivo.

12.2.2. Experimento 2

En este experimento, se cambiarán algunos pasos de orden para demostrar que el sistema no sigue un flujo preestablecido y, además, no se reducirá el *dataset*.

1. Cargar el *dataset* y previsualizar los datos

2. Preprocesar algunos de los datos del *dataset*.

3. Analizar alguna característica del conjunto de datos.

4. Realizar el entrenamiento del modelo

En este experimento y en los próximos, se reutilizarán las Figuras que muestran exactamente lo mismo que debería de mostrar en dicho paso. Esto se realiza para evitar la redundancia en Figuras.

Caso 1 – AmesHousing

En primer lugar, se carga el *dataset* y se previsualizan los datos del conjunto (ver Figura 4).

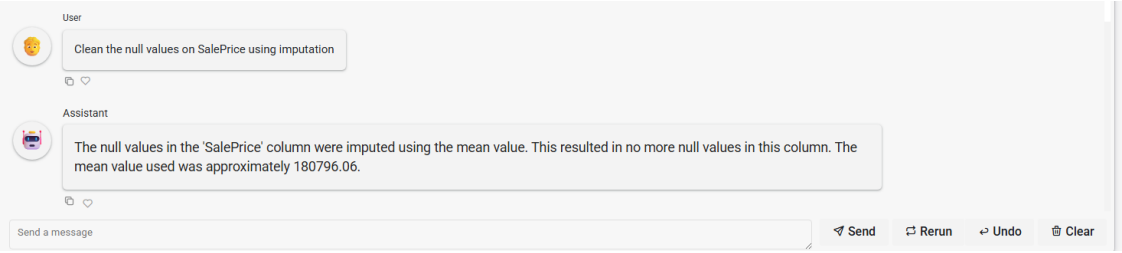

Figura 26. Evidencia Preprocessing Data Experimento 2 (AmesHousing)

En la figura 26 se muestra una limpieza de valores nulos en la columna "*SalePrice*". Esto será de gran ayuda a la hora de obtener la mayor precisión en el entrenamiento del modelo.

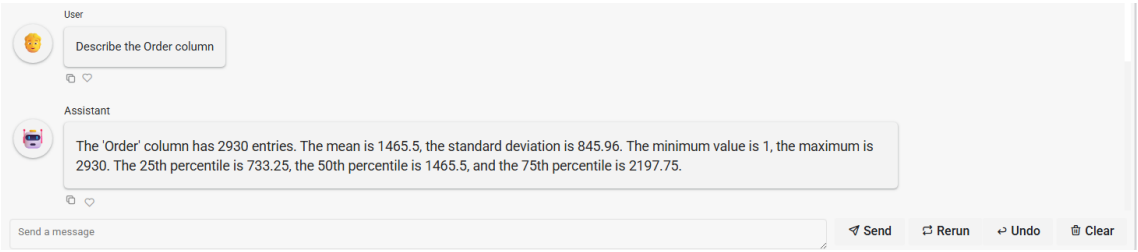

Figura 27. Evidencia EDA Experimento 2 (AmesHousing)

En la Figura 27 se muestra un análisis de la columna "Order". Aquí se comprueba que se puede preprocesar los datos antes de analizarlos (aunque no es lo recomendable).

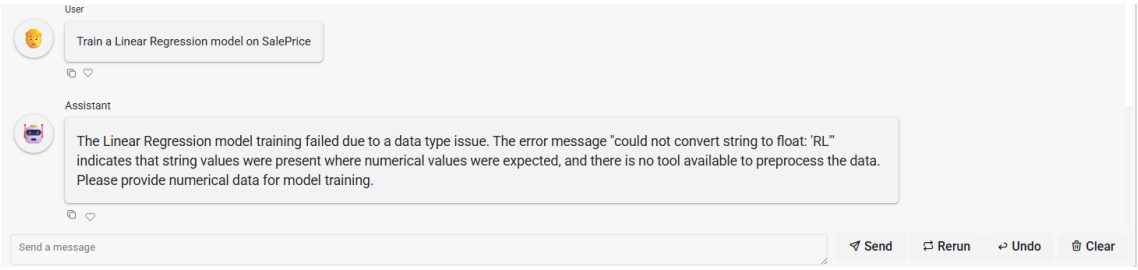

Figura 28. Evidencia Model Training Linear Regression Experimento 2 (AmesHousing)

En la Figura 28 se intenta entrenar el modelo usando *Linear Regression*, pero da error debido a que existen variables categóricas en el *dataset* y este algoritmo no puede trabajar con dichas variables. Para solucionarlo, se debería convertir todas las variables categóricas a numéricas.

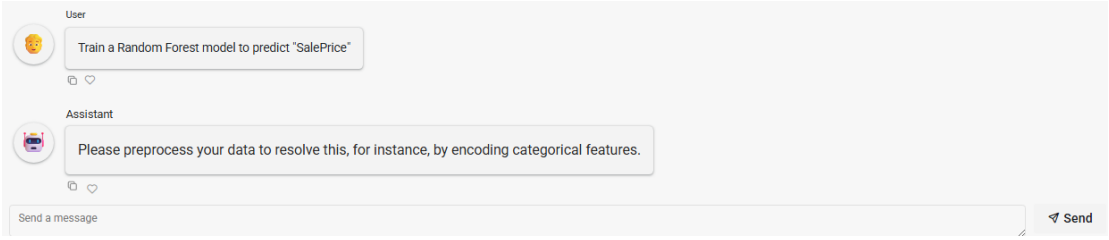

Figura 29. Evidencia Model Training Random Forest Experimento 2 (AmesHousing)

En la Figura 29 ocurre lo mismo que en la Figura 25. En esta ocasión, se intenta entrenar el modelo usando *Random Forest*, pero no retorna la salida correspondiente e indica, a modo de sugerencia, que se debería de preprocesar los datos para poder realizar el entrenamiento.

En este caso, al disponer de tantas variables en este *dataset* (ver Tabla 19), se ha optado por no entrenar el modelo.

Caso 2 – Iris

Primero, se carga el dataset y se previsualizan los datos del conjunto (ver Figura 11).

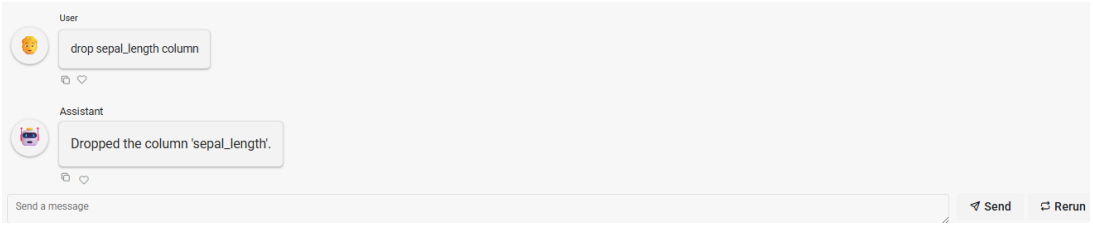

Figura 30. Evidencia Preprocessing Data Experimento 2 (Iris)

En la Figura 30, se observa que se ha solicitado eliminar la columna "*sepal_length*".

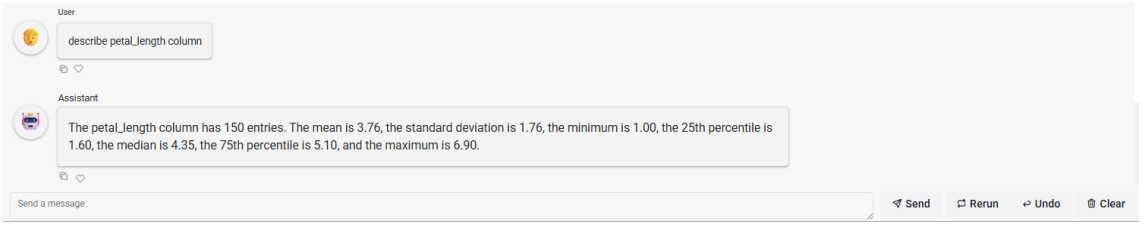

Figura 31. Evidencia EDA Experimento 2 (AmesHousing)

En la Figura 31, se detalla un análisis estadístico de la columna "petal_length".

Figura 32. Evidencia Model Training Logistic Regression Experimento 2 (AmesHousing)

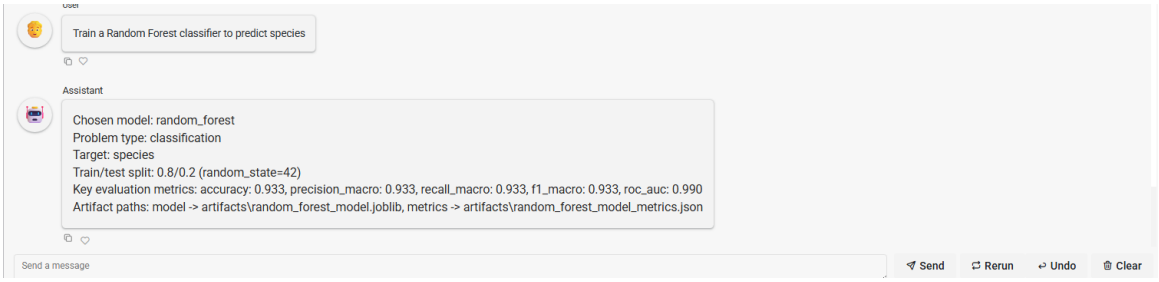

Figura 33. Evidencia Model Training Random Forest Experimento 2 (AmesHousing)

En la Figura 32 y 33 se comprueba que se ha realizado el entrenamiento del modelo correctamente usando *Logistic Regression* y *Random Forest* respectivamente.

Modelo	Accuracy	Precision_macro	Recall_macro	F1_macro	Roc_auc
Logistic Regression	0.933	0.933	0.933	0.933	0.997
Random Forest	0.933	0.933	0.933	0.933	0.99

Tabla 22. Métricas de Experimento 2 (Iris)

En la Tabla 22 se muestra la comparación entre ambos modelos. Las métricas de ambos son muy parecidas a las del experimento 1 (ver Experimento 1. Caso 2 –Iris).

En conclusión, se podrían utilizar cualquiera de los dos modelos.

Caso 3 – Bank

En primer lugar, se carga el *dataset* y se previsualizan los datos del conjunto (ver Figura 17).

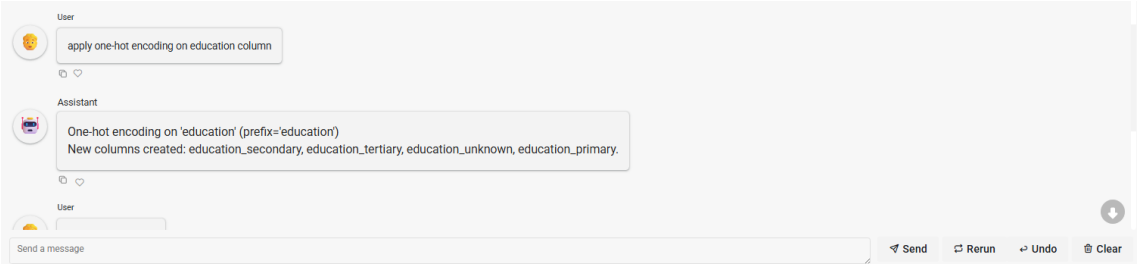

Figura 34. Evidencia Preprocessing Data Experimento 2 (Bank)

En la Figura 34 se puede observar que se ha convertido a numérica la columna "*education*".

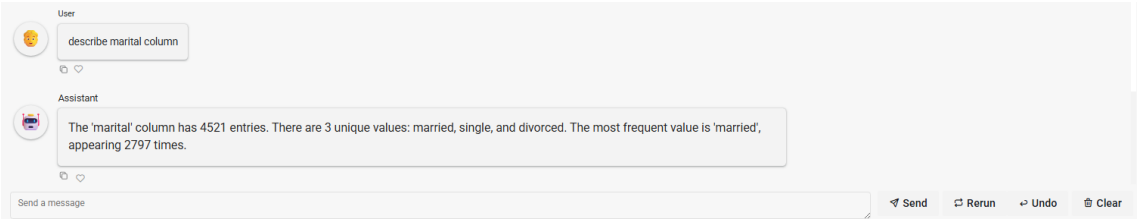

Figura 35. Evidencia EDA Experimento 2 (Bank)

En la Figura 35 se observa que se ha obtenido las estadísticas de la variable "*marital*".

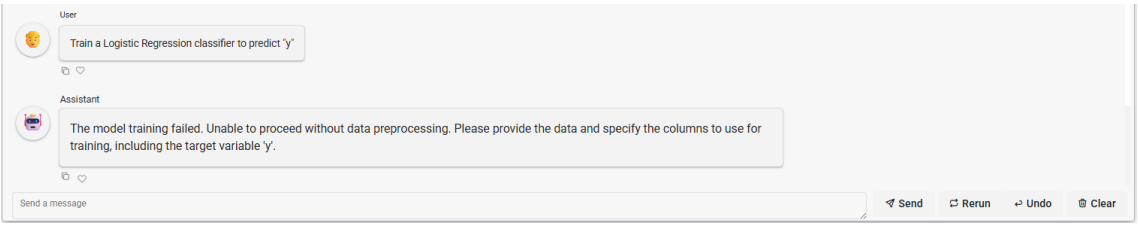

Figura 36. Evidencia Model Training Logistic Regression Experimento 2 (Bank)

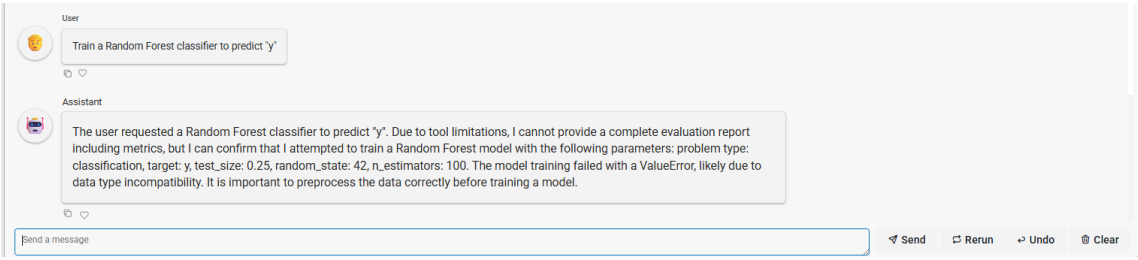

Figura 37. Evidencia Model Training Experimento 2 (Bank)

En las Figuras 36 y 37 ocurre lo mismo que ha ocurrido en el Caso 1 de este experimento (ver Experimento 2. Caso 1 –AmesHousing). No se ha podido realizar el entrenamiento debido a que existen valores categóricos en otras columnas, por lo que la solución debería de ser la misma.

12.2.3. Experimento 3

En este experimento, se realizará el pipeline muy similar al Experimento 1, con la única diferencia de que en este se reducirá el *dataset* a través de una selección de instancias.

1. Cargar *dataset* y previsualizar los datos.

2. Realizar análisis estadístico de la variable objetivo.

3. Realizar preprocesamiento de algunos datos.

4. Reducir el *dataset* a través de la selección de instancias.

5. Realizar el entrenamiento del modelo.

Caso 1 – AmesHousing

Como en los anteriores experimentos, se carga el *dataset* y se previsualizan los datos del conjunto (ver Figura 4).

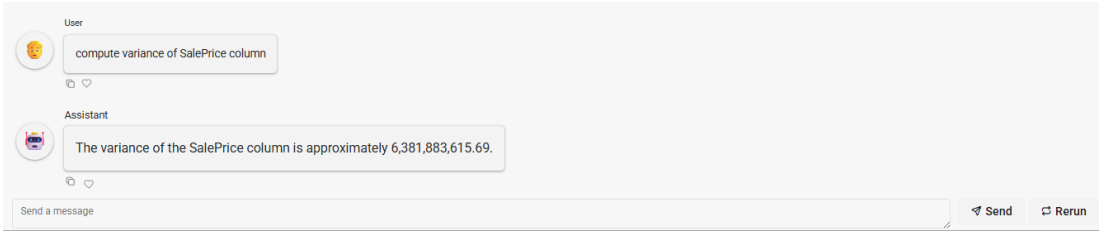

Figura 38. Evidencia EDA Experimento 3 (AmesHousing)

En la figura 38 se observa el cálculo de la varianza de la variable objetivo. Con ello, se conoce que los datos están muy dispersos ya que es un número muy alto.

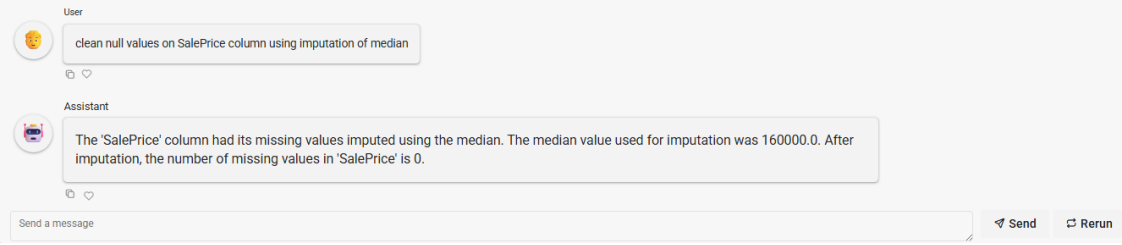

Figura 39. Evidencia Preprocessing Data Experimento 3 (AmesHousing)

En la Figura 39, se realiza una limpieza de valores nulos sobre la variable objetivo. Tras esto, ya no existe ningún valor nulo en dicha variable.

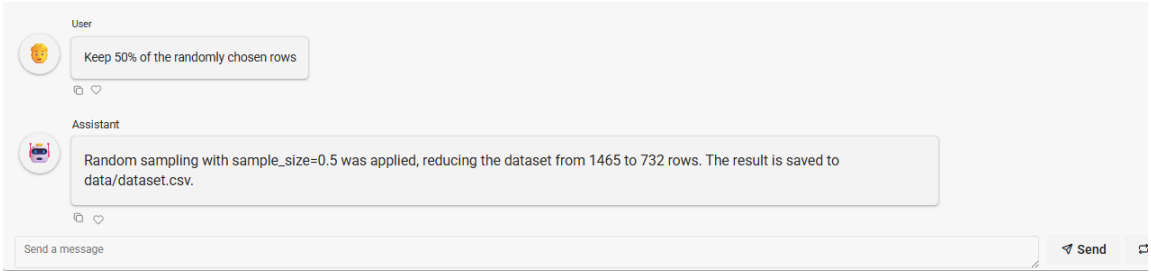

Figura 40. Evidencia Instance Selection Experimento 3 (AmesHousing)

En la Figura 40 se realiza una selección de instancias del 50% de manera aleatoria sin tener en cuenta la distribución de las clases.

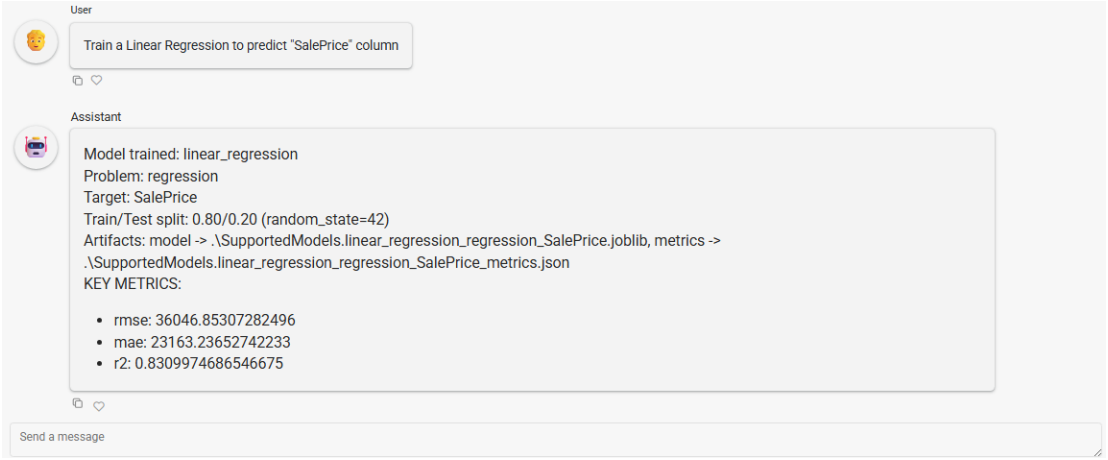

Figura 41. Evidencia Model Training Linear Regression Experimento 3 (AmesHousing)

En la Figura 41 se realiza el entrenamiento del modelo utilizando Linear Regression.

Figura 42. Evidencia Model Training Random Forest Experimento 3 (AmesHousing)

En la Figura 42 se puede observar que se ha realizado correctamente el entrenamiento del modelo usando Random Forest.

Modelo	RMSE	MAE	R^2
Linear Regression	36046.853	23163.237	0.831
Random Forest	40041.248	20179.557	0.791

Figura 43. Métricas de Experimento 3 (AmesHousing)

En la Tabla 40, se observa la comparación en métricas de ambos modelos. En este caso, se puede comprobar que el *Linear Regression* explica ligeramente mejorar la variabilidad de los precios de las casas (R^2) y tiene errores globales más bajos (*RMSE*).
El *Random Forest* comete menos errores medios por predicción (*MAE*), aunque explica un poco menos los datos.

Caso 2 – Iris

Como en los anteriores experimentos, se carga el dataset y se previsualizan los datos del conjunto (ver Figura 11).

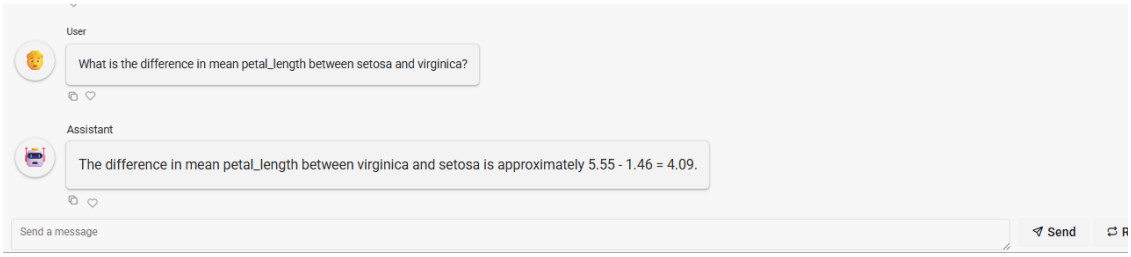

Figura 44. Evidencia EDA Experimento 3 (Iris)

En la Figura 44, se ha solicitado la diferencia de medias entre dos valores de la columna objetivo con respecto a "*petal_length*".

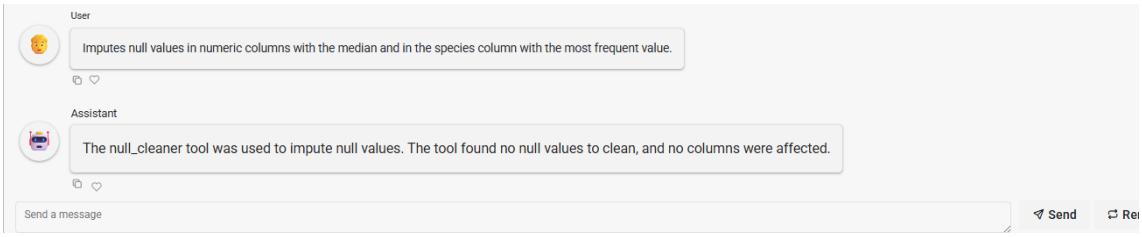

Figura 45. Evidencia Preprocessing Data Experimento 3 (Iris)

En la Figura 45, se ha intentado limpiar los valores nulos de las columnas, pero este *dataset* no tienen ningún valor nulo, por lo que lo indica el asistente.

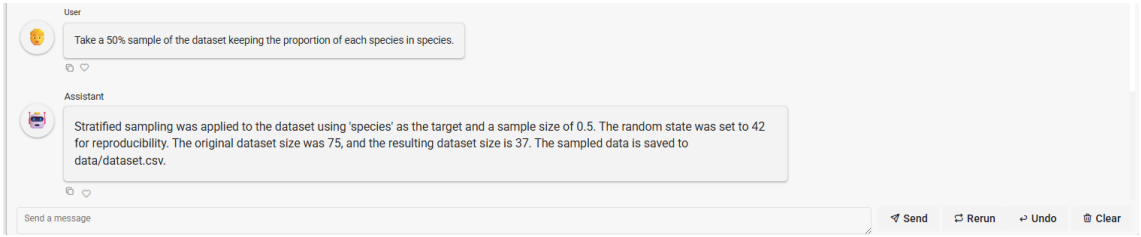

Figura 46. Evidencia Instance Selection Experimento 3 (Iris)

En la Figura 46 se observa que se ha reducido el *dataset* un 50% manteniendo la proporción de clases de "*species*"

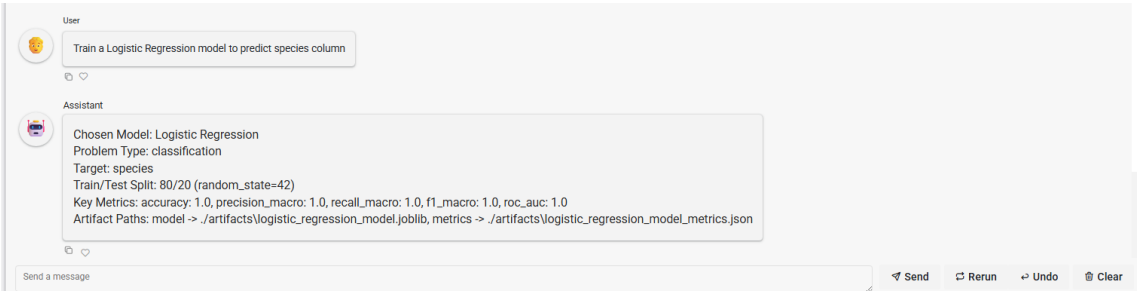

Figura 47. Evidencia Model training Logistic Regression Experimento 3 (Iris)

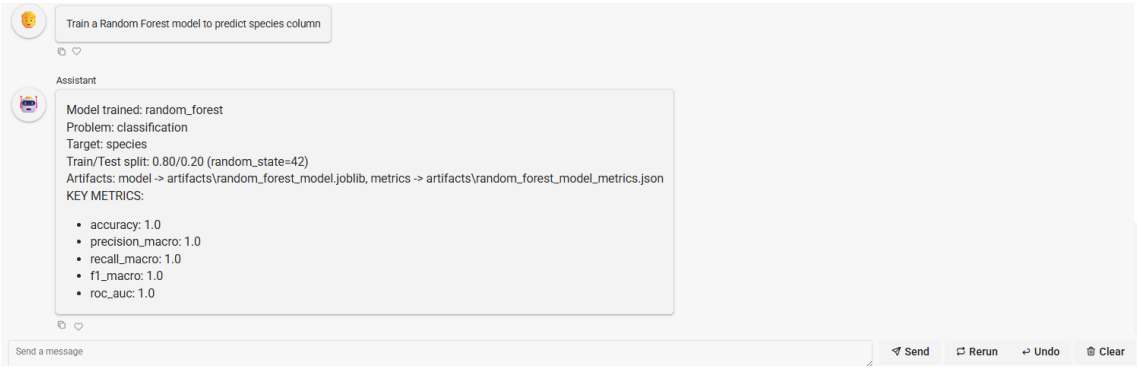

Figura 48. Evidencia Model Training Random Forest Experimento 3 (Iris)

En las Figuras 47 y 48 se puede observar cómo se ha realizado el entrenamiento del modelo usando *Logistic Regressor* y *Random Forest* respectivamente.

Modelo	Accuracy	Precision_macro	Recall_macro	F1_macro	Roc_auc
Logistic Regression	1.0	1.0	1.0	1.0	1.0
Random Forest	1.0	1.0	1.0	1.0	1.0

Tabla 23. Métricas de Experimento 3 (Iris)

En la Tabla 23 se observa que se han entrenado los modelos de manera perfecta ya que ambos tienen en todas las métricas 1.0.

Caso 3 – Bank

Como en los anteriores experimentos, se carga el *dataset* y se previsualizan los datos del conjunto (ver Figura 4).

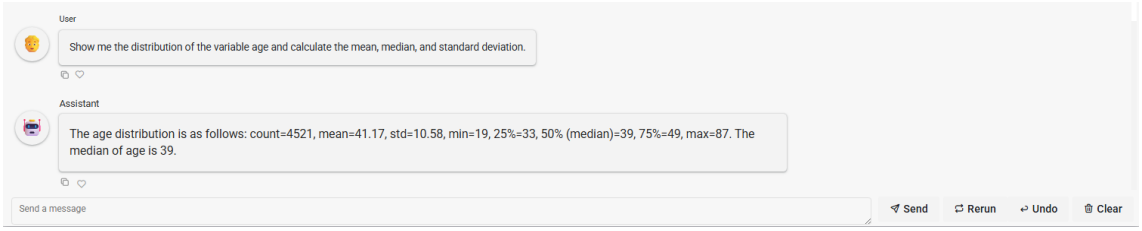

Figura 49. Evidencia EDA Experimento 3 (Bank)

En la Figura 49 se muestra la distribución de la variable "*age*".

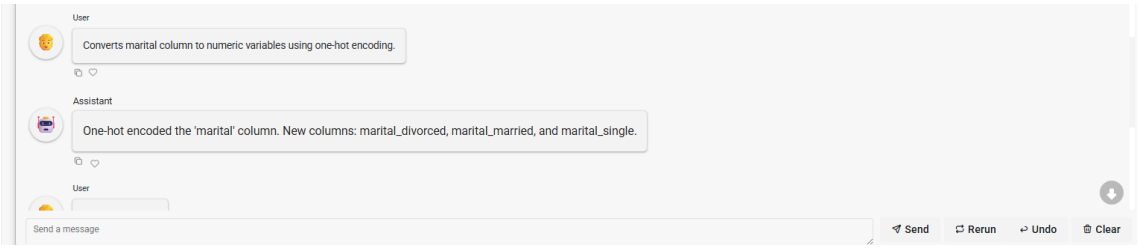

Figura 50. Evidencia Preprocessing Data Experimento 3 (Bank)

En la Figura 50 se ha convertido la variable "*marital*" a variables numéricas.

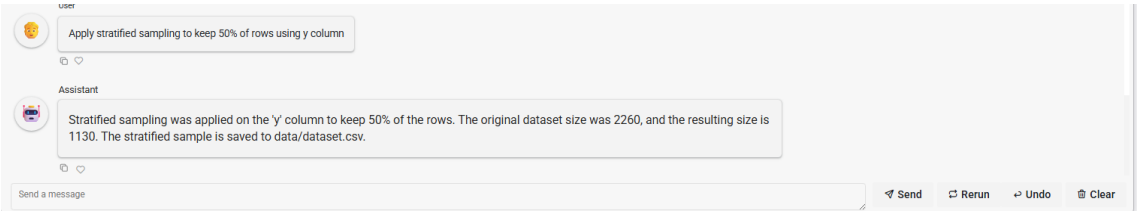

Figura 51. Evidencia Instance Selection Experimento 3 (Bank)

En la Figura 51 se ha reducido las instancias al 50% teniendo en cuenta la variable objetivo.

Figura 52. Evidencia Model Training Logistic Regression Experimento 3 (Bank)

Figura 53. Evidencia Model Training Random Forest Experimento 3 (Bank)

En las Figuras 52 y 53 se puede comprobar que se ha realizado el entrenamiento del modelo usando Logistic Regression y Random Forest.

Modelo	Accuracy	Precision_macro	Recall_macro	F1_macro	Roc_auc
Logistic Regression	0.996	0.981	0.998	0.989	1.0
Random Forest	1.0	1.0	1.0	1.0	1.0

Tabla 24 Métricas de Experimento 3 (Bank)

En la Tabla 24 se puede ver la comparación entre los dos modelos usados. Se puede ver que prácticamente los dos modelos están entrenados muy bien.

12.2.4. Experimento 4

En este experimento se busca comprobar que el sistema no solo entiende peticiones relacionadas con *machine learning*, sino que también es capaz de mantener una conversación coherente con el usuario cuando se le plantean temas ajenos al ML.

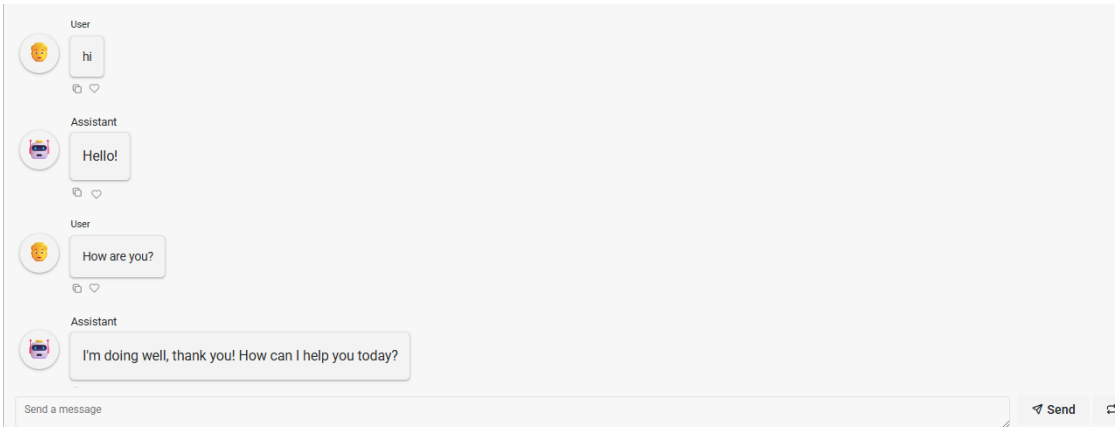

Figura 54. Evidencia Conversación 1 Experimento 4

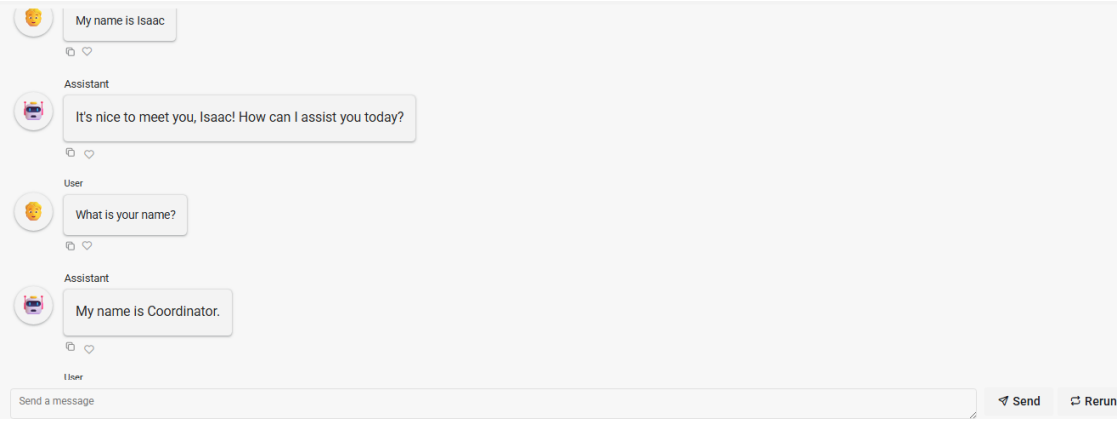

Figura 55. Evidencia Conversación 2 Experimento 4

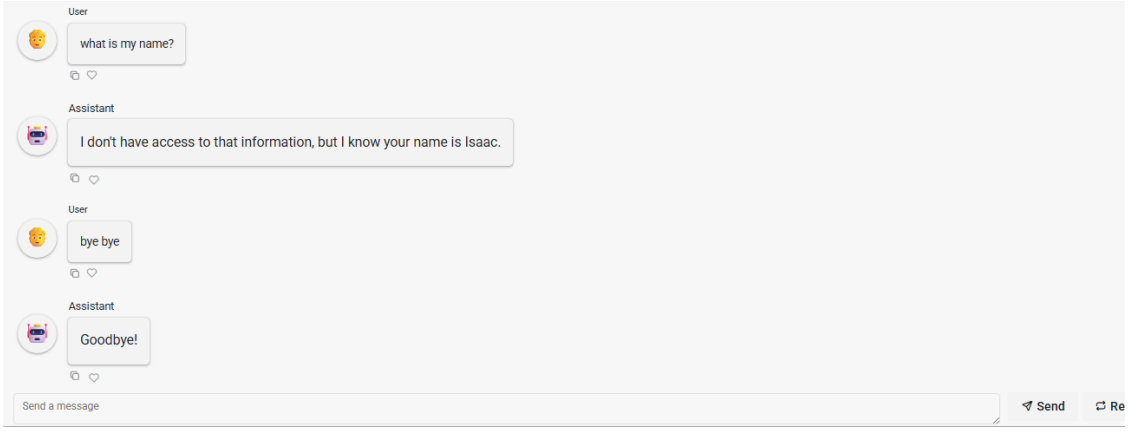

Figura 56. Evidencia Conversación 3 Experimento 4

En las Figuras 54, 55 y 56 se puede comprobar que el sistema mantiene una conversación coherente con el usuario. Además, se puede comprobar que el sistema dispone de memoria.

12.3. Discusión crítica

Los experimentos realizados permiten sacar varias conclusiones de peso sobre el sistema desarrollado y, al mismo tiempo, dejan al descubierto alguna de sus limitaciones.

En primer lugar, se ha comprobado que el sistema multiagente es capaz de construir pipelines completos de aprendizaje supervisado a partir de interacciones en lenguaje natural, cubriendo todas las fases. Lo interesante es que este comportamiento se ha mantenido de manera consistente frente a *datasets* de diferente naturaleza, lo que confirma la versatilidad del enfoque propuesto.

Los casos concretos lo ilustran bien. Con *AmesHousing*, el sistema afrontó un problema de regresión de alta dimensionalidad y demostró con técnicas como la discretización de variables o la selección de características pueden marcar la diferencia en el rendimiento. Con el clásico Iris, quedo claro que el sistema maneja con soltura problemas de clasificación multiclase, identificando las características más relevantes y alanzando métricas de precisión muy altas. Y en el caso de *Bank Marketing*, el reto fue aún mayor ya que es un *dataset* con muchas variables categóricas y clases desbalanceadas. Aquí también respondió con acierto, aplicando preprocesamientos adecuados y entrenando modelos que ofrecieron resultados muy completos.

Ahora bien, no todo fueron aciertos. También aparecieron limitaciones importantes. El sistema depende bastante de que los datos estén preprocesados correctamente, ya que algunos algoritmos no admiten variables categóricas. Además, la flexibilidad que aporta el lenguaje natural también tiene su parte negativa: en ocasiones genera ambigüedad. Si se le aporta ordenes incompletas o poco precisas obligan al sistema a pedir aclaraciones, lo que puede ralentizar la conversación.

Aun con todo, los experimentos muestran que el sistema cumple con su objetivo principal que es reducir la barrera de entrada al aprendizaje supervisado y permitir que personas

sin experiencia técnica puedan configurar y evaluar pipelines mediante instrucciones tan naturales como una conversación. Y es que el verdadero valor añadido está en que el asistente no sigue un guion rígido, es decir, se adapta a distintas órdenes y contextos, ofreciendo al usuario flexibilidad.

Capítulo 13. Conclusiones y líneas futuras

El trabajo desarrollado ha permitido demostrar la viabilidad de un asistente multiagente capaz de construir pipelines de aprendizaje supervisado a partir de simples instrucciones en lenguaje natural. Tal y como se ha ido mostrando a lo largo de esta memoria, el sistema consigue abarcar todas las fases principales de un flujo de *machine learning*. Todo ello integrando librerías de uso común como *scikit-learn* y adaptando las operaciones a *datasets* de naturaleza muy distinta (ver Capítulo 12).

Si miramos atrás, en relación con los objetivos planteados en el inicio (ver Capítulo 2), puede afirmarse que el propósito general se ha cumplido. El usuario puede formular sus peticiones en lenguaje natural y recibir como respuesta pipelines completos y reproducibles. Además, los objetivos específicos definidos en el anteproyecto también se alcanzaron en gran medido ya que se diseñó una arquitectura modular con agentes especializados, se integraron herramientas de ML y se desarrolló una interfaz funcional en *Panel* que hace que la interacción sea mucho más intuitiva (ver Capítulo 9).

Entre las fortalezas del software merece la pena destacar varias. Por un lado, la posibilidad de interactuar en lenguaje natural con cierta flexibilidad lo vuelve más accesible a quienes no tienen experiencia técnica. Por otro, la reproducibilidad de resultados asegurando la persistencia de artefactos. Y, por último, su capacidad de afrontar escenarios muy variados, desde problemas de regresión hasta clasificación binaria y multiclase (ver Capítulo 12).

Ahora bien, no todo son fortalezas. También se identificaron limitaciones que condicionan el alcance del sistema en su estado actual. La más evidente es que en ocasiones el usuario debe formular instrucciones claras y explicitas ya que puede ocurrir que el asistente no sea capaz de deducir que operación es la más adecuada (ver Capítulo

12.3). Algo similar ocurre con las técnicas de balanceo de clases, que de momento se reducen a métodos básicos de muestreo. También conviene señalar que la codificación *one-hot* puede disparar el número de variables en *datasets* de alta cardinalidad.

Estas limitaciones, sin embargo, se convierten también en oportunidades de mejora. Algunas líneas prometedoras incluyen la incorporación de técnicas de preprocesamiento más avanzadas, el incremento del nivel de autonomía del agente coordinador (de forma que pueda sugerir las acciones a realizar), la visualización en forma de gráficos de los conjuntos de datos y la posibilidad de comparar el rendimiento de varios modelos entrenados de manera distinta.

En definitiva, el sistema desarrollado constituye un primer paso solido hacia la integración de modelos de lenguaje y arquitectura multiagente en la configuración de pipelines de aprendizaje supervisado. Aunque todavía presenta limitaciones, los resultados obtenidos confirman que esta aproximación es viable y con un gran potencial para la democratización del aprendizaje automático.

Cabe mencionar que el desarrollo de este TFG ha sido un reto exigente tanto en lo académico como en lo humano. Mi formación previa estaba centrada en la Ingeniería del Software y apenas tenía experiencia en técnicas de *machine learning*. Eso transformó el proyecto en una oportunidad excepcional para explorar un campo nuevo, lo que me forzó a adquirir conocimientos sobre las distintas etapas de un pipeline de aprendizaje supervisado. Si bien el proceso ha sido complicado y a veces desafiante, lo cierto es que el esfuerzo valió la pena ya que he podido extender mi formación hacia un campo de gran importancia hoy en día.

Por otro lado, este TFG ha permitido demostrar las competencias adquiridas en el grado de Ingeniería Informática, incluyendo la capacidad de análisis y diseño de sistemas, programación avanzada, uso de técnicas de IA, trabajo autónomo, gestión de bibliografía científica y comunicación escrita.

Bibliografía

[1]. Murphy, K. P. (2012). *Machine learning: A Probabilistic Perspective*. MIT Press.

[2]. NetApp. (2025, March 21). Qué es el aprendizaje automático (ML), importancia del aprendizaje automático. *NetApp*. https://www.netapp.com/es/artificial-intelligence/what-is-machine-learning/

[3]. Scikit-learn. (n.d.). *Normalizer*. Scikit-learn. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.Normalizer.html

[4]. GeeksforGeeks. (2025, July 23). *What is Data Cleaning?* GeeksforGeeks. https://www.geeksforgeeks.org/data-analysis/what-is-data-cleaning/?utm_source=chatgpt.com

[5]. Google for Developers. (n.d.). *Imbalanced datasets* https://developers.google.com/machine-learning/crash-course/overfitting/imbalanced-datasets?utm_source=chatgpt.com&hl=es-419

[6]. IBM. (2025, 27 febrero). *Random Forest. ¿Qué es el Random Forest?* https://www.ibm.com/mx-es/topics/random-forest

[7]. Ibm. (2025a, enero 28). *Grandes modelos de lenguaje. ¿Que son los grandes modelos de lenguaje (LLM)?* https://www.ibm.com/es-es/topics/large-language-models?utm_source=chatgpt.com

[8]. OpenAI. (n.d.). *ChatGPT*. https://chatgpt.com/

[9]. Google. (n.d.). *Gemini*. https://deepmind.google/gemini

[10]. Microsoft. (2023). *Copilot: AI assitant for coding* [Software]. https://copilot.microsoft.com/

[11]. Stryker, C., & Kavlakoglu, E. (2025, 12 febrero). *Inteligencia artificial. ¿Qué es la IA?* IBM. https://www.ibm.com/es-es/think/topics/artificial-intelligence

[12]. Redacción UNIR. (2024, julio 11). *Ingeniería de software: qué es, objetivos y funciones del ingeniero*. Universidad Internacional de La Rioja. https://colombia.unir.net/actualidad-unir/ingenieria-de-software-que-es-objetivos/

[13]. HoloViz. (n.d.). *Panel v1.7.5*https://panel.holoviz.org/

[14]. GeeksforGeeks. (2025, 11 julio). *Classification vs Regression in Machine Learning*. GeeksforGeeks. https://www.geeksforgeeks.org/machine-learning/ml-classification-vs-regression/?utm_source=chatgpt.com

[15]. Scikit-learn. (n.d.). *scikit-learn: Machine learning in Python.* https://scikit-learn.org/stable/

[16]. pandas development team. (n.d.). *pandas: Python data analysis library.*https://pandas.pydata.org/

[17]. CrewAI. (n.d.). *Introduction to CrewAI.* https://docs.crewai.com/en/introduction

[18]. AutoML.org. (n.d.). *Auto-Sklearn: Automated machine learning for tabular data.* https://www.automl.org/automl-for-x/tabular-data/auto-sklearn/

[19]. TPOT. (n.d.). *Tree-based Pipeline Optimization Tool.* https://epistasislab.github.io/tpot/latest/

[20]. Amazon Web Services. (n.d.). *Machine learning (ML) e inteligencia artificial (IA): Descripción general. AWS Whitepapers.*https://docs.aws.amazon.com/es_es/whitepapers/latest/aws-overview/machine-learning.html?utm_source=chatgpt.com#amazon-sagemaker

[21]. Zhou, Z., Jin, J., Phadnis, V., Yuan, X., Jiang, J., Qian, X., Wright, K., Sherwood, M., Mayes, J., Zhou, J., Huang, Y., Xu, Z., Zhang, Y., Lee, J., Olwal, A., Kim, D., Iyengar, R., Li, N., & Du, R. (2025). *InstructPipe: Generating Visual Blocks Pipelines with Human Instructions and LLMs. Building Visual Programming Pipelines With Human Instructions Using LLMs, 1-22.* https://doi.org/10.1145/3706598.3713905

[22]. Khattab, O., Singhvi, A., Maheshwari, P., Zhang, Z., Santhanam, K., Vardhamanan, S., Haq, S., Sharma, A., Joshi, T. T., Moazam, H., Miller, H., Zaharia, M., & Potts, C. (2023, October 5). *DSPy: Compiling Declarative Language Model Calls into Self-Improving Pipelines. arXiv.org.* https://arxiv.org/abs/2310.03714?utm_source=chatgpt.com

[23]. Cryptoniche (n.d.). *Generative AI architecture a comprensive guide to it´s achitecture and components*. Medium. https://medium.com/cryptoniche/generative-ai-architecture-a-comprehensive-guide-to-its-architecture-and-components-in-2024-5912fb6a6023

[24]. LM Studio. (n.d.). *LM Studio documentation.* https://lmstudio.ai/docs/app

[25]. Google. (n.d.). *Flutter documentation.*https://docs.flutter.dev/?_gl=1*k0rwfo*_ga*MTAxODQ2NTQ1NC4xNzI5NTM3OTU0*_ga_04YGWK0175*czE3NTY0NzA5NTAkbzMkZzAkdDE3NTY0NzA5NTAkajYwJGwwJGgw

[26]. Bhavik Jikadara (2024, July 30). *Data Science Automation: A Step-by-Step Guide Using CrewAI. Medium.* https://medium.com/ai-agent-insider/data-science-automation-a-step-by-step-guide-using-crewai-e1468823e0f8

[27]. Luque, M. (s.f.). *Modelo de requisitos*. En *Diseño y construcción del software*.

[28]. Simões, C. (2020, July 14). *MoSCoW. ¿Qué es y cómo priorizar en el desarrollo de tu aplicación?* Blog ITDO - Agencia De Desarrollo Web, APPs Y Marketing En Barcelona. https://www.itdo.com/blog/moscow-que-es-y-como-priorizar-en-el-desarrollo-de-tu-aplicacion/

[29]. Luque, M. (s.f.). *Modelado de casos de uso*. En *Diseño y construcción del software*.

[30]. *Rumbaugh, J., Jacobson, I., & Booch, G. (2004). UML reference manual. Addison-Wesley.* https://repository.unikom.ac.id/37898/1/Addison%20Wesley%20-%20UML%20Reference%20Manual.pdf

[31]. IBM. (2024a, febrero 21). *¿Qué es el clustering?* https://www.ibm.com/es-es/think/topics/clustering

[32]. Fernández, Y. (2025, July 9). *Qué es un prompt y por qué son tan importantes para usar la Inteligencia Artificial*. Xataka. https://www.xataka.com/basics/que-prompt-que-importantes-para-usar-inteligencia-artificial

[33]. Streamlit. (2025, mayo 24). *Streamlit documentation.*https://docs.streamlit.io/

[34]. Kaggle. (2018, septiembre 10). *Ames housing dataset*https://www.kaggle.com/datasets/prevek18/ames-housing-dataset?resource=download

[35]. Fisher, R. (1936). Iris [Dataset]. *UCI Machine Learning Repository*https://doi.org/10.24432/C56C76.

[36]. Moro, S., Rita, P., & Cortez, P. (2014). Bank marketing [Dataset]. *UCI Machine Learning Repository*. https://doi.org/10.24432/C5K306.

Anexos

Los anexos tienen como finalidad proporcionar documentación complementaria que facilite tanto el uso del sistema desarrollado como su posible mantenimiento y ampliación por parte de futuros desarrolladores. En este caso se incluyen los siguientes documentos:

- **Anexo I - Manual de Usuario**: Es el documento que permite guiar al usuario dentro del sistema.

- **Anexo II- Manual de Código**: Repositorio documentado donde se incluye todo el código fuente del proyecto.

Agradecimientos

Este trabajo no habría sido posible sin el apoyo de muchas personas que, de una forma u otra, han estado presentes en este largo camino.

En primer lugar, quiero dar las gracias a mi familia que ha estado ahí en todo momento. Gracias por animarme en los días más largos, por recordarme que el esfuerzo siempre merece la pena y darme la energía necesaria para seguir cuando parecía que no quedaba.

Pero en especial, quiero agradecer a mi pareja. Me ha estado apoyando cada día, me ha aguantado cuando estaba muy agobiado, cuando no tenía mucha energía para seguir y cuando necesitaba desahogarme. La verdad es que sin ella jamás hubiera podido lograr los resultados que he conseguido.

Por otro lado, quiero agradecer a la comunidad de desarrolladores y autores de recursos abiertos. Su generosidad al compartir conocimiento fue una fuente constante de aprendizaje e inspiración.

Y, por último, me quiero agradecer a mí mismo la perseverancia y la capacidad de aprender a contrarreloj. Este TFG ha sido un reto exigente, sí, pero también una oportunidad única para crecer, no solo como estudiante, sino también como persona.