



Universidad Politécnica  
de Madrid

**Escuela Técnica Superior de  
Ingenieros Informáticos**



Grado en Ingeniería Informática

Trabajo Fin de Grado

**MLOps: Administrando el Diseño y  
Ciclo de Vida de los Modelos de  
Machine Learning.**

Autor: Alonso García Velasco

Tutor(a): Santiago Tapia Fernández

Madrid, Mayo - 2023

Este Trabajo Fin de Grado se ha depositado en la ETSI Informáticos de la Universidad Politécnica de Madrid para su defensa.

*Trabajo Fin de Grado*  
*Grado en Ingeniería Informática*

*Título:* MLOps: Administrando el Diseño y Ciclo de Vida de los Modelos de Machine Learning.

Mayo - 2023

*Autor:* Alonso García Velasco  
*Tutor:* Santiago Tapia Fernández  
Lenguajes y Sistemas Informáticos e Ingeniería de Software  
ETSI Informáticos  
Universidad Politécnica de Madrid

# Resumen

En este trabajo se tratará las razones porque las que es necesario el uso de MLOps (Machine Learning Operation) para realizar desarrollos de Machine Learning (ML). Para ello se justificará históricamente su aparición, se describirá como metodología, se seleccionará la herramienta que se crea más adecuada para el problema propuesto, se definirá una arquitectura para soportarla, se definirá la metodología específica que se ha diseñado y se aplicará a un caso de uso. Finalmente se estudiará su impacto y deficiencias.

La inteligencia artificial (IA) durante su historia ha tenido épocas álgidas y otras más discretas (inviernos de la IA). Por lo que su desarrollo hasta los últimos años ha sido más lento, y por lo tanto no se contaba con metodologías específicas de desarrollo. MLOps es la extensión de DevOps para ML. DevOps una metodología de desarrollo ágil, basada en la división entre el entorno de desarrollo y el de producción, la integración y entrega continua (CI/CD). MLOps añade estas ideas a los desarrollos de ML. Para lograr esto se ha de automatizar las fases de un desarrollo de ML. La limpieza y preparación de datos han de ser recogidas en un pipeline que permita la preparación de datasets, lo mismo ocurre con los entrenamientos. Se ha de añadir un sistema para dar replicabilidad y trazabilidad a todos los experimentos realizados y a los modelos generados como salida. La puesta en producción de los modelos se realizará sobre servicios web que monitoricen la calidad de los modelos en el tiempo. En el caso de que la calidad de los modelos disminuya es necesario la reactivación de los pipelines para la generación de nuevos modelos.

MLOps tiene varios niveles: 0, 1, 2. EL nivel 0 todo ha realizarse de manera manual, no hay nada automatizado, con el único elemento propio de MLOps que se cuenta es con su herramienta para la gestión de entrenamientos y de modelos. En el nivel 1 ya existe una gestión automatizada de los pipelines de datos es decir los entrenamientos y preparación de datos, aunque su activación sigue siendo manual. En el nivel 3 se automatiza la activación de los pipelines de datos, es decir se crean los pipelines de CI/CD.

Para seleccionar la herramienta que gestionará los modelos y entrenamientos se realizará un estudio sobre las herramientas disponibles. Existe una gran selección de herramientas cada una teniendo sus ventajas e inconvenientes propios. Se ha optado por elegir MLflow una herramienta de código abierto, con mucho énfasis en la flexibilidad y una comunidad de desarrollo activa. Además, es usada para realizar tareas de MLOps en otras plataformas ML más amplias como puede ser Data Bricks. Su nivel MLOps es 0 pero con otras herramientas como ETLs (Extract, Transform, Load) puede incrementar su nivel.

Para el despliegue de MLflow se ha optado por un despliegue in-premise median-

te contenedores Docker, uno por cada microservicio que sea necesario. MLflow en cuanto a su arquitectura está formado por tres partes, el servidor de MLflow, una base de datos para el almacenamiento de los parámetros y métricas de los entrenamientos y un sistema de ficheros distribuidos que almacene los artefactos, para los ficheros relacionados con los entrenamientos, incluyendo el propio modelo. MLflow y el gestor de bases de datos seleccionado, PostgreSQL son desplegados desde contenedores Docker, mientras el sistema de ficheros se almacena en un NAS y se comparte por NFS (para sistemas UNIX) y SMB (para sistemas Windows). Todos los componentes que almacenan datos son sometidos a back-ups periódicamente.

Una vez seleccionada la herramienta es necesario realizar diseñar una metodología que siga MLOps que se adapte a la herramienta. Se seguirán los mismos pasos que los definidos por MLOps. Primero se ha de describir el problema, después limpiar y preparar los datos, estas dos ultimas han de ser automatizadas. Después se ha de realizar los experimentos para generar los modelos, esto se divide en ejecuciones, de cada ejecución se ha de registrar parámetros, métricas, artefactos, modelo, entrada y salida del modelo, código de entrenamiento y datasets. Después se ha de seleccionar el modelo con respecto a las métricas, exponerlo como servicio y monitorizar su precisión. En el caso de que decaiga se ha de volver a ejecutar los pipelines que preparan los datos y realizan los entrenamientos, elegir el nuevo modelo y volver a ponerlo en producción sustituyendo el antiguo. Esto será ejemplificado con un desarrollo.

Por último, se revisará el impacto y deficiencias de la metodología producida. El impacto esperado es una reducción de los tiempos de desarrollo, la cantidad repeticiones necesarias para encontrar el modelo óptimo. Aunque en cuanto a sus deficiencias si que cabe resaltar que al no tener un nivel de MLOps 0 o 1 esas automatizaciones recaen en los usuarios o en otras herramientas.

# Abstract

This project will cover the reasons why MLOps (Machine Learning Operations) should be implemented in ML (Machine Learnings) developments. In order to achieve these goals, MLOps would be justified from an historical point of view and the methodology itself is going to be described. Followingly, the most adequate tool would be selected, and the architecture produce to sustain it would be described. Using all of the previous elements a specific methodology would set in place and test. Finally, the impact and deficiencies would be studied.

In the history of Artificial Intelligence (AI) there has been high and low moments (AI winters). Until the most recent years AI had not had a specific methodology to itself. MLOps is an extension to DevOps (Develop Operations). DevOps is an agile methodology, based on the division between development and production environments and continuous integration and deployment (CI/CD). MLOps brings these ideas into ML developments. The process of cleaning and preparing the data must be integrated into a pipeline, as well as the training processes. A system to provide traceability and replicability, for the experiments that took place and the models produced, has to be put in place. Moving into the production side the models should be expose as web services. In the case of model quality decline a trigger should restart all the processes in order to obtain new models.

MLOps has the following levels: 0, 1 and 2. In the level 0 everything has to be done manually, there is nothing automated. The only element from MLOps that is present is the tool to mange training and models. In the level 1 the automatic management of pipelines of training and data appear, although its activations keep being manual. The last level provides automatization of the activation of data pipelines, i.e., the CI/CD pipelines.

Successively, a study of the available tools to perform the management of the models and the training processes. There is a great selection of tools each one with its own set of advantages and drawbacks. MLflow, an open-source tool, has been selected for its emphasis on flexibility, its active developer community. Moreover, MLflow is used as MLOps tool in other ones much more brought such as Data Bricks. Its MLOps level is 0 although it can be improved external tools such as ETL (Extract, Transform, Load) it can be upgraded.

The deployment of MLOps is done in-premise with docker containers, one for each microservice. MLflow architecture is divided into there main parts: MLflow server, a database for storing the parameters and the metrics; and a distributed file system, for files related with trainings, including the model. MLflow and PostgreSQL, the selected data base management system, are deployed as docker

containers, whereas the file system is deployed in a NAS and shared via NFS (for UNIX systems) and SMB (for Windows systems). All the systems that store data are backed-up periodically.

## Agradecimientos

Muchas gracias a todos los que me habéis ayudado en este camino, en especialmente a mis padres y familiares. Agradezco de manera particular también a C.R.I.D.A. por permitirme realizar este proyecto, sobre todo al equipo de desarrollo por su guía y ayuda, especialmente a Faustino Tello Caballo. A los profesores que me han impartido asignaturas y conocimientos, no solo durante este grado, si no durante toda mi vida estudiantil. A mi tutor para este trabajo Santiago Tapia Fernández. A mis compañeros de clase y amigos con los que he compartido estos años de estudio. A las comunidades de desarrollo libre sobre las que se soporta parte de este trabajo. Y a todo aquel que esté obviando pero que sin su contribución no hubiera sido posible relizar este Trabajo de Fin de Grado.





# Índice general

<b>1. Introducción y objetivos</b>	<b>1</b>
1.1. Introducción	1
1.1.1. DevOps	1
1.1.2. Historia inteligencia artificial	2
1.1.3. MLOps	3
1.2. Objetivos	11
<b>2. Descripción del problema.</b>	<b>13</b>
<b>3. Análisis de las diferentes plataformas MLOps</b>	<b>15</b>
3.1. Plataformas disponibles	15
3.1.1. Amazon SageMaker	15
3.1.2. Azure Machine Learning	16
3.1.3. Mlflow	17
3.1.4. Databricks	19
3.1.5. Domino	20
3.1.6. Vertex AI (Google Cloud)	21
3.1.7. KubeFlow	23
3.1.8. H2O MLOps	24
3.2. Selección de plataforma	25
<b>4. Diseño, despliegue y configuración de la plataforma.</b>	<b>27</b>
4.1. Requisitos	27
4.2. Tecnologías usadas	27
4.2.1. Docker	27
4.2.2. HDFS	28
4.2.3. NFS	28
4.2.4. SMB	28
4.2.5. Entornos virtuales	29
4.3. Diseño arquitectura	29
4.3.1. Metodología	29
4.3.2. Mlflow	29
4.3.3. Registro de experimentos, métricas e hiperparámetros	30
4.3.4. Registro de artefactos	30
4.4. Implementación arquitectura	31
4.4.1. Mlflow	31
4.4.2. Postgres	32
4.4.3. HDFS	32
4.4.4. NFS	33
4.4.5. SBM	33

4.4.6. Despliegue conjunto . . . . .	33
4.5. Estrategia de <i>backup</i> y redundancia . . . . .	34
4.5.1. <i>Backup</i> servicios . . . . .	34
4.5.2. <i>Backup</i> de datos . . . . .	34
4.6. Esquema final de la arquitectura . . . . .	36
<b>5. Definición de la metodología</b>	<b>39</b>
5.1. Descripción del problema . . . . .	39
5.2. Análisis de los datos . . . . .	40
5.3. Limpieza de los datos . . . . .	40
5.4. Realización de experimentos . . . . .	41
5.5. Selección de mejor modelo . . . . .	43
5.6. Puesta en producción . . . . .	43
5.7. Monitorización del modelo . . . . .	44
5.8. Reentrenamiento o sustitución de modelo . . . . .	45
<b>6. Resultados</b>	<b>47</b>
6.1. Descripción de problema . . . . .	47
6.2. Análisis de los datos . . . . .	47
6.3. Limpieza de los datos . . . . .	57
6.4. Realización de experimentos . . . . .	57
6.4.1. AutoML (h2o) . . . . .	57
6.4.2. Random forest (Sklearn) . . . . .	59
6.4.3. KNN (Sklearn) . . . . .	63
6.5. Selección del modelo . . . . .	65
6.6. Puesta en producción . . . . .	69
6.7. Monitorización del modelo . . . . .	70
6.8. Reentrenamiento o sustitución del modelo . . . . .	70
<b>7. Análisis de impacto.</b>	<b>73</b>
<b>8. Conclusiones.</b>	<b>75</b>
<b>9. Líneas futuras de mejora</b>	<b>77</b>
<b>A. Apendice</b>	<b>79</b>
A.1. Archivos de configuración docker necesarios para el despliegue de la arquitectura. . . . .	79
A.1.1. Dockerfile Mlflow . . . . .	79
A.1.2. Dockerfile Postgres . . . . .	81
A.1.3. Docker compose y .env . . . . .	81
A.1.4. Manual instalación. . . . .	82
A.2. Uso de la herramienta . . . . .	85
A.2.1. Manual de usuario . . . . .	85
<b>Bibliografía</b>	<b>93</b>

# 1. Introducción y objetivos

## 1.1. Introducción

### 1.1.1. DevOps

La ingeniería de software apareció a mitad del siglo pasado y aun siendo un área del conociendo relativamente nueva debido al gran interés que despierta desde su comienzo ha generado metodologías y buenas prácticas en la creación, utilización y mantenimiento del software.

Las metodologías de desarrollo han sido variadas desde los inicios de la ingeniería del software, entre los que se han dado el desarrollo en cascada, el iterativo, incremental, en V, en espiral, entre otras. En los últimos años se está optando por las metodologías ágiles que han de seguir el manifiesto Agile [BBvB<sup>+</sup>01].

El manifiesto ágil defiende la entrega rápida y de funcional de código al cliente para poder realizar sucesivas iteraciones, y realizar los cambios sobre lo especificado, mejorando la colaboración entre desarrolladores y responsables de negocio, la motivación de los trabajadores, la comunicación rápida y eficaz, el manteniendo del ritmo de trabajo, atención a la excrecencia técnica, simplicidad, equipos autoorganizados, y las reuniones frecuentes para la reflexión y el ajuste.

Una de las variantes más populares dentro de las metodologías ágiles es DevOps (*Developing Operations*). Se base en la unión entre el desarrollo y la operación del sistema. En este modelo se generan dos entornos, preproducción o desarrollo y producción, ambos operados por distintos equipos interconectados. Los cambios frecuentes se pueden probar y testear sobre el entorno de preproducción, para ser posteriormente aplicados y desplegados sobre producción con mayor facilidad. Esto también trae con sigo otras ventajas como la capacidad de testeo en un entorno lo más parecido posible a la realidad. Esto proporciona test más seguros y la capacidad de desarrollar una sola vez, cuando un desarrollo sea probado y funcione en preproducción debería poder ser portado directamente a producción, sin ningún error.

El proceso DevOps no es solo seguir los principios del desarrolla ágil y adaptar los sistemas, si no, también la reestructuración de los equipos de desarrollo y respaldo a usuarios finales. El cambio tiene que ser soportado puesto que todo lo desarrollado está destinado a cambiar en un momento u otro. No se ha de producir el mismo resultado de otra manera, se ha de producir mediante una arquitectura que permite el cambio y se adaptarte al ritmo de producción e implantación.

Otro de los beneficios de DevOps es la reducción de los ciclos de desarrollo y el aumento de la velocidad de implementación a través de añadir a Integración y Entrega Continua.

### 1.1.2. Historia inteligencia artificial

Hay algunas ramas dentro de la programación que por sus necesidades no encajan directamente dentro de la metodología DevOps, este es el caso de los algoritmos de *Machine Learning* (ML). El *Machine Learning* es un área dentro de la Inteligencia Artificial (IA). La inteligencia artificial nació a mediados del siglo pasado, y el termino fue acuñado en la Conferencia de Dartmouth, New Hampshire, Estados Unidos. En los primeros años de investigación su desarrollo fue bastante grande con avances tanto teóricos como prácticos. Entre los teóricos se encontraban la teoría de la información de Shannon, importante para realizar análisis entre variables y sus relaciones, o la de la computación de Alan Turing, base de la computación y de las arquitecturas como la Von Neumann. En cuanto a las prácticas se encontraba el perceptrón, que simulaba una neurona humana. Esta época se caracteriza por el optimismo con respecto a la IA.

Hacia los años 70 se da el primer invierno de la IA, periodo denominado con este nombre debido a la falta de avances, puesto que no se había cumplido las expectativas debido sobre todo a la falta de potencia de cálculo. Por ejemplo, el Cray-1 de 1975 solo podía realizar operaciones a 80MHz y tenía 8MB de RAM, unas 45 veces más lento y con una memoria principal de tres órdenes de magnitud más pequeña que en la actualidad, con un precio ajustado a la actualidad de 35 millones de dolares y un consumo de 115KW. Otro factor limitante era la falta del volumen de datos necesarios para poder usar estos sistemas, que en aquel momento era imposible siquiera almacenar esa cantidad de datos. Otra de las criticas recibidas durante esta época fue la incapacidad del perceptrón de realizar clasificación no lineal. Todo esto provoco la retirada de la financiación por la falta de resultados, aunque proyectos con una aproximación algorítmica e imperativa como la programación necesaria para los módulos de programa Apollo-11 [Gar19] sí que obtuvieron financiación.

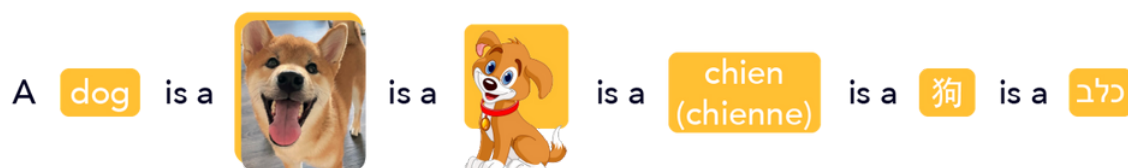
Los avances en computación tanto en rendimiento como en paradigmas, como la programación lógica, en concreto Prolog (*PROgrammation en LOGique*), que utiliza reglas y axiomas para generar resultados. Esto propicio que a finales de los 80s surgieran los sistemas expertos basados en reglas como MYCIN, el retorno de las neuronas artificiales, esta vez con más de una capa, activación no lineal y un nuevo algoritmo para entrenarlas (retro propagación del gradiente). Durante estos años se lanzaron proyectos que han sentado las bases para la creación de tecnologías en las que aún utilizamos para proyectos actuales como el coche autónomo, por ejemplo el proyecto Prometeus de Mercedes-Benz [MB19].

En 1997 se dio uno de los hitos más relevantes de la IA Deep Blue fue capaz de derrotar a Garry Kasparov en su segundo encuentro con el resultado DeepBlue 3-1-2 Kasparov. Debido a la complejidad del árbol de juego del ajedrez, es decir, los posibles movimientos que se pueden realizar, aproximadamente  $10^{120}$ , o el

número de Shannon, es imposible generara un libro de jugadas tan grandes como para realizar una búsqueda exhaustiva de la jugada. Por lo que se optó por el uso de reglas para distintas fases de la partida, guardando estados de partidas ya jugadas, y conjuntos de aperturas y cierres.

En la última década ha habido un gran desarrollo nos encontramos en una situación completamente distinta a la del invierno de la IA. El *hardware* se ha vuelto mucho más potente y asequible, tanto en el caso de una administración y uso totalmente gestionado por la empresa internamente, como utilizando *IaaS* (Infraestructura como Servicio), *PaaS* (Plataforma como Servicio) o *SaaS* (*Software* como Servicio) ofrecidos por terceras empresas. La cantidad de datos generados a diario y recogidos durante años ahora sí que permite el uso de la IA. Las empresas y entidades públicas están haciendo una inversión muy importante en este área, como D.A.R.P.A. (*Defense Advanced Research Projects Agency*) [DAR22], Open AI y su librería de IAs cada una enfocada en un propósito distinto. Por ejemplo Chat-GPT [Ope23] genera texto, DALL-E [RPG<sup>+</sup>21] imágenes, Jukebox [DJP<sup>+</sup>20]sonidos a partir siguiendo como base un artista y un estilo.

Entre los últimos avances se encuentran el *Deep Learnign* (DL), IAs multimodales, IAs generativas entre otras. El DL es un tipo de redes de neuronas artificiales con múltiples capas ocultas que han demostrado tener una mejor capacidad para la realización de tareas antes imposibles para otros tipos de IA o de red neuronal. Las IAs multimodales permiten la comprensión de múltiples fuentes de datos y usan la similitud entre la información para generar relaciones y salidas en cualquier forma. Por ejemplo, tal y como se muestra en la siguiente figura [Int18] un perro puede ser cualquier entrada de las siguientes, en varios lenguajes naturales como pueden ser el inglés y el francés, una foto o un dibujo. Las IAs generativas son capaces de generar texto, imágenes, sonidos. . . a partir de ruido o de una entrada definida por el usuario. Uno de los tipos de IA generativa con más éxitos es la red GAN (*Generative Adversary Network*) [GPAM<sup>+</sup>14] que enfrenta a una IA generadora con una clasificadora, la primera intenta general el objetivo, mientras que la segunda filtra si lo producido es real o ha sido generado.



**Figura 1.1.:** IA multimodal [Int18]

### 1.1.3. MLOps

Actualmente los desarrollos de *Machine Learning* están dirigidos por los datos y sus necesidades son distintas, en contraste con el enfoque algorítmico que tienen los desarrollos clásicos. A esto hay que añadirle que dependiendo de las

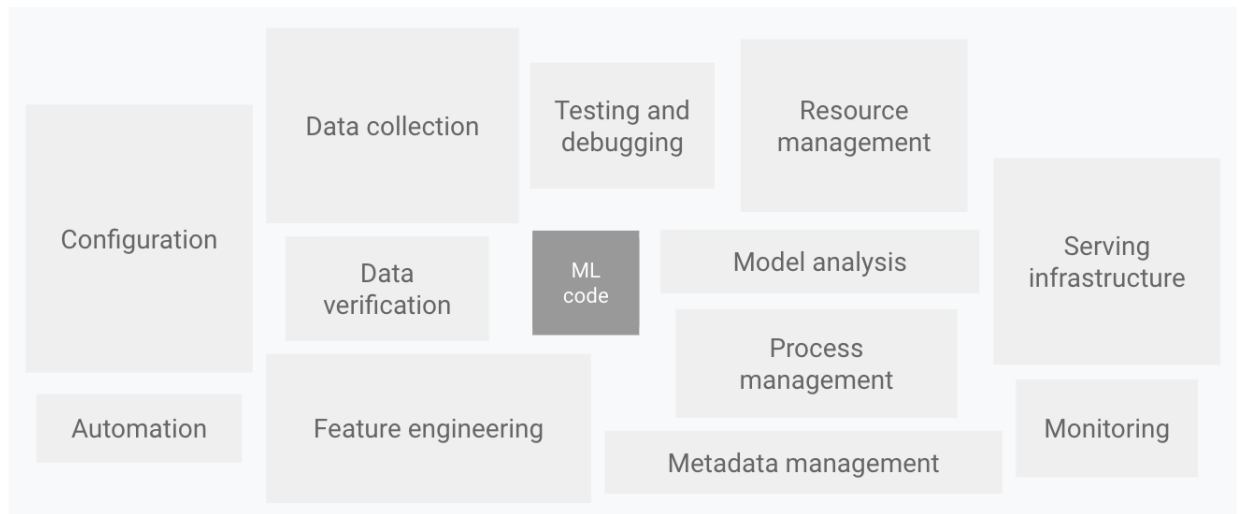
fuentes solo entre un 10 % y 13 % de desarrollos llegan a ser puestos en producción [Rob20] [Dic19]. Los motivos de esta baja llegada a producción se encuentran en el siguiente artículo [HZRS16] y la metodología recogida en [Clo21b]. En el citado artículo se discuten los motivos, apareciendo como principal la deuda técnica que se adquiere al usar *frameworks* de terceros, la desaparición de los límites de los proyectos, cliente inesperados, problemas de configuración y cambios en el entorno de los desarrollos que les afectan.

Nos encontramos en una situación en la que las IAs están consiguiendo resultados muy favorables para las empresas que las consiguen implantar, pero debido a sus grandes diferencias en sus fases de desarrollo no hay una metodología que seguir. Gracias a la experiencia acumulada en los últimos 70 años en metodologías de desarrollo software no se comienza de cero y una de las iniciativas es combinar una de las metodologías más exitosas como DevOps con las necesidades propias de *Machine Learning* (MLOps).

En MLOps aparte de cumplir como base de DevOps siguiendo CI (integración continua) y CD (entrega continua), a estos se le añade CT (*Continuos Training*, entrenamiento continuo) para desarrollos ML (*Machine Learning*). El objetivo es poder tener modelos lo más rápido posibles en producción que podrán ser modificados tanto como sea necesario para satisfacer las necesidades del cliente, al igual que sucede con la metodología DevOps.

Una de la diferencia es que los modelos han de ser periódicamente testeados para probar su valía en cada momento del tiempo y deben ser reentrenados para mejorar las prestaciones existentes, aplicar nuevas técnicas o acomodar nuevos datos que se han producido más tarde la puesta en producción del modelo. Al igual que DevOps es necesario automatizar tantas fases como sea posible para aumentar la productividad y replicabilidad de los procesos realizados.

El desarrollo ML es la pieza central de un proyecto ML, es necesario gestionar también su configuración, automatización, la recolección, verificación y extracción de características de los datos, testeo y corrección de errores, análisis del modelo, gestión de recursos, procesos y metadatos, monitorización. Como se muestra en la siguiente figura 1.2..



**Figura 1.2.:** Arquitectura MLOps [Clo21b]

### DevOps vs MLOps

- Características de los componentes de los equipos de ML: suelen estar formados por perfiles especializados en el tratamiento de los datos y no en su puesta en producción.
- Desarrollo: Los proyectos han de estar centrados en los datos y como se relacionan para construir un modelo, el código queda en un segundo plano y suele ser reutilizado (en la medida de lo posible) de un proyecto a otro.
- Testeo: Se comprueba como de bien o mal funciona el modelo generado con métricas típicamente estadísticas y no se suelen usar test unitarios, de integración o de sistema.
- Implementación: Un proyecto no es solo el modelo generado si no todos los pasos realizados para generarlo, normalmente estos quedan recogidos en un *pipeline* o un *data Flow* (flujo de datos).
- Producción: Una vez puestos en producción su degradación es mayor que la de desarrollos clásicos pues además de decaer de todas las maneras que un sistema clásico la continua evolución del perfil de los datos afecta al rendimiento del modelo. Por lo tanto, es necesario una continua revisión de las perdiciones del modelo y realizar reentrenamientos en el caso de ser necesario.

En cuanto a las características propias de DevOps también son modificadas para encajar mejor con desarrollos ML. Para mantener CI se deben validar los datos en vez de componentes y algoritmos, se validan los datos y los modelos producidos. Las entregas pasan a ser *pipelines* de datos.

### Pasos en el proceso

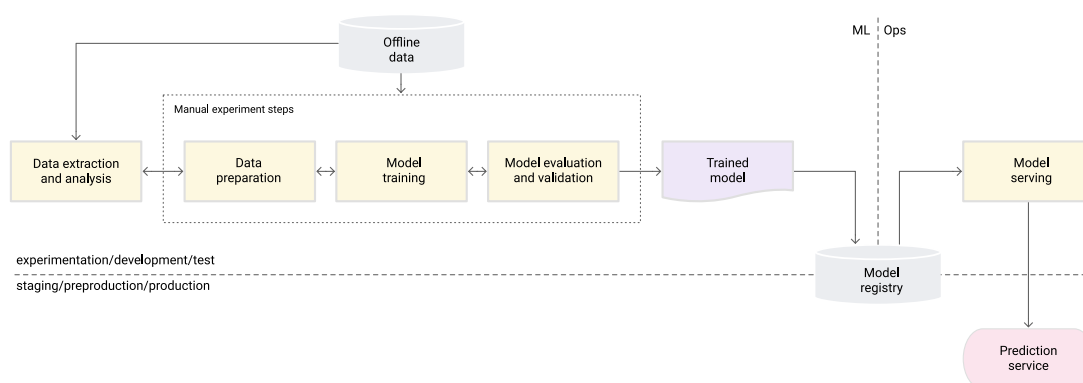
- **Extracción de datos:** Recogida de datos de las fuentes de datos para el proyecto que se va a desarrollar. Las fuentes de los datos pueden ser múltiples, por ejemplo, bases de datos propias, o ajenas, ficheros, APIs de una red social. . . Por lo que es necesario realizar una importación de estos, a poder ser a través herramientas automatizadas o ETLs (*Extract Transform Load*).
- **Análisis de los datos:** Conocimiento de los datos y sus características y planificación de la preparación de los datos. Una de las técnicas más usadas es el análisis exploratorio de los datos (EDA [IBM23], por sus siglas en inglés). El EDA se trata de un estudio de los datos y sus características, frecuentemente mediante visualización. Facilita la detección de patrones y anomalías, y la comprobación de hipótesis.
- **Preparación de los datos:** Limpieza de los datos, separación en subconjuntos para entrenamiento, testeo y validación. Es necesario el tratamiento de los datos para evitar cualquier error en ellos como tener todas las unidades en las mismas unidades, datos con variables no presentes. . . En este punto también es necesario realizar la selección de variables. Se ha de separar los conjuntos de datos en sus conjuntos de entrenamiento, prueba y validación.
- **Entrenamiento del modelo:** Se prueba con distintos algoritmos para entrenamiento y distintos hiperparámetros atrás encontrar los óptimos. Se prueba con distintos algoritmos para entrenamiento y distintos hiperparámetros atrás encontrar los óptimos. Se puede usar técnicas como el AutoML para ellos, o probar ir seleccionando todos los modelos a probar y tunear para conseguir los mejores resultados de ellos.
- **Evaluación de los modelos:** Se comprueba la eficacia de los modelos usados usando los conjuntos de prueba. Se comprueban los resultados de los modelos para la obtención de métricas para su evaluación. Estos dos pasos se realizan iterativamente para ir mejorando los resultados.
- **Validación de los modelos:** Se comprueba la validez de los modelos con un conjunto de datos que previamente se separó y reservó para esta fase. Cuando ya se tenga un modelo listo para su puesta en producción se realiza una última validación con unos datos con los que nunca ha entrado en contacto.
- **Servir el modelo:** El modelo es puesto en producción por medio normalmente de un API. Para evitar reentrenamientos de modelos todos quedan almacenados en un repositorio común y son solicitados por aquel que los necesite, también reduce la necesidad de potencia de cálculo fuera del servidor que contenga los modelos, pues este se encarga del entrenamiento y predicción.
- **Monitorización del modelo:** Comprobar con cierta regularidad el buen funcionamiento del modelo para poder cambiarlo por uno más efectivo en el caso de ser necesario. Tener esta información permite realizar decisiones



## 1.1 Introducción

soportadas en datos sobre cuando reentrenar, aporta métricas del rendimiento real de los datos. Además, aporta nuevos datos para la generación de los modelos que algún sustituirán al que este activo.

### Niveles MLOps

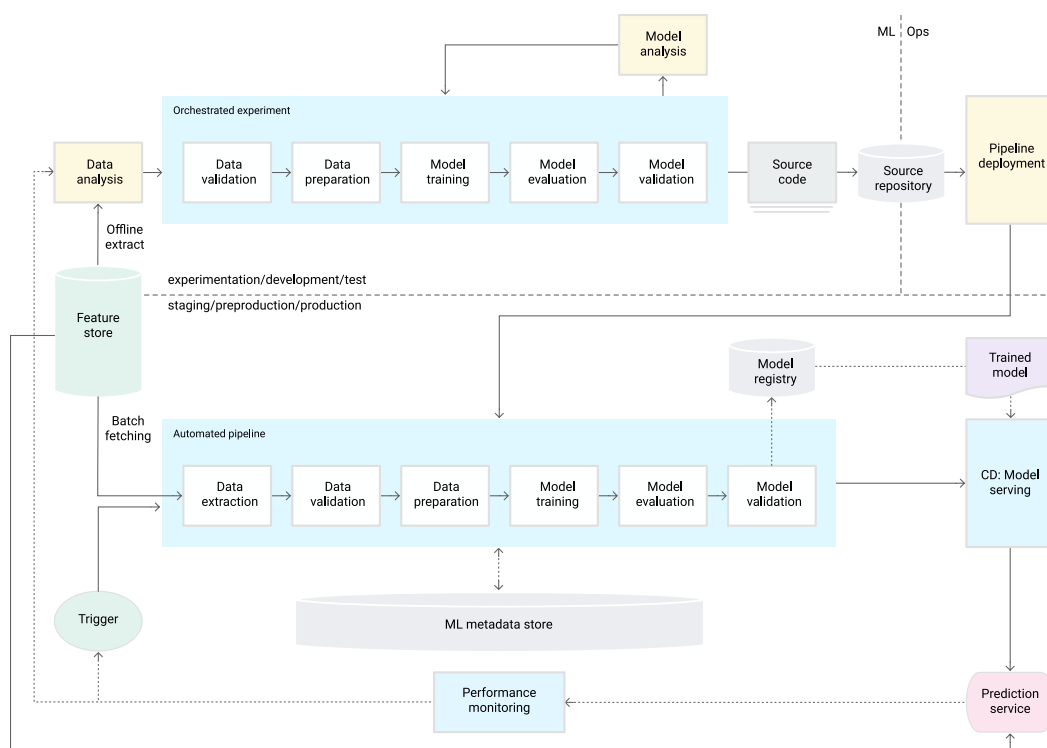


**Figura 1.3.:** MLOps nivel 0[Clo21b]

**MLOps Nivel Cero, Proceso manual** En la figura 1.3. se muestra un posible *pipeline* MLOps nivel 0.

Los procesos para el tratamiento de los datos son ejecutados de manera manual, incluyendo las transiciones entre pasos. Existe una desconexión entre los procesos producidos en el *pipeline* con la operación de los modelos, tienen que ser entregados del equipo de desarrollo al de producción. Se almacena todo en sistemas compartidos que sean capaces de guardar los parámetros necesarios para la ejecución de las fases y los artefactos producidos y necesarios por y para la ejecución del *pipeline*. Desde ahí pueden ser servidos a producción. Es válido si los cambios en los modelos son pocos y no se espera un incremento en la cantidad de modelos. No hay integración continua, ni entrega ni comprobación de estado de los modelos.

No cumple con todos los requisitos de MLOps para un desarrollo, pero es la base puesto que si existe una manera de guardar los parámetros y artefactos necesarios para reentrenos, trazabilidad o para su puesta en producción.



**Figura 1.4.:** MLOps nivel 1 [Clo21b]

**MLOps Nivel Uno, ML *pipeline* automatizado** En la figura 1.4. se muestra un posible *pipeline* MLOps nivel 1.

Al tener el *pipeline* para el tratamiento de los datos automatizados la generación de experimentos es mucho más rápida. El modelo puede ser alterado una vez puesto en producción con datos nuevos para mantenerlo siempre al día. Los *pipelines* usados en preproducción y producción son los mismos por lo que se siguen los preceptos DevOps. Se entrega continuamente nuevos modelos entrenados con datos nuevos, ya testeados y validados. También se aumenta la capacidad de reutilizar el código al tener que generar *pipelines* repetibles.

Para obtener todas las anteriores ventajas se ha de incluir nuevos componentes.

- Validador de datos que pueda reentrenar el modelo o para su ejecución de manera automática. Debe de tener en cuenta las diferencias las anomalías en los esquemas de datos recibidos y en los propios valores de los datos.
- Validador de modelos que comprueba las características del nuevo modelo generado antes de que llegue a producción. Ha de realizar los siguientes pasos:
  - Generar las nuevas métricas de evaluación.
  - Compararlas con el modelo antiguo para asegurarse un mejor comportamiento.

## 1.1 Introducción

---

- Comparar las métricas para distintos segmentos de datos. Un modelo puede ser más adecuado para cierta parte de los datos usados, pero tener un peor desempeño general.
- Probar que es compatible con la infraestructura en la que se va a servir el modelo.

Se ha de gestionar los siguientes metadatos para aumentar la reproducibilidad.

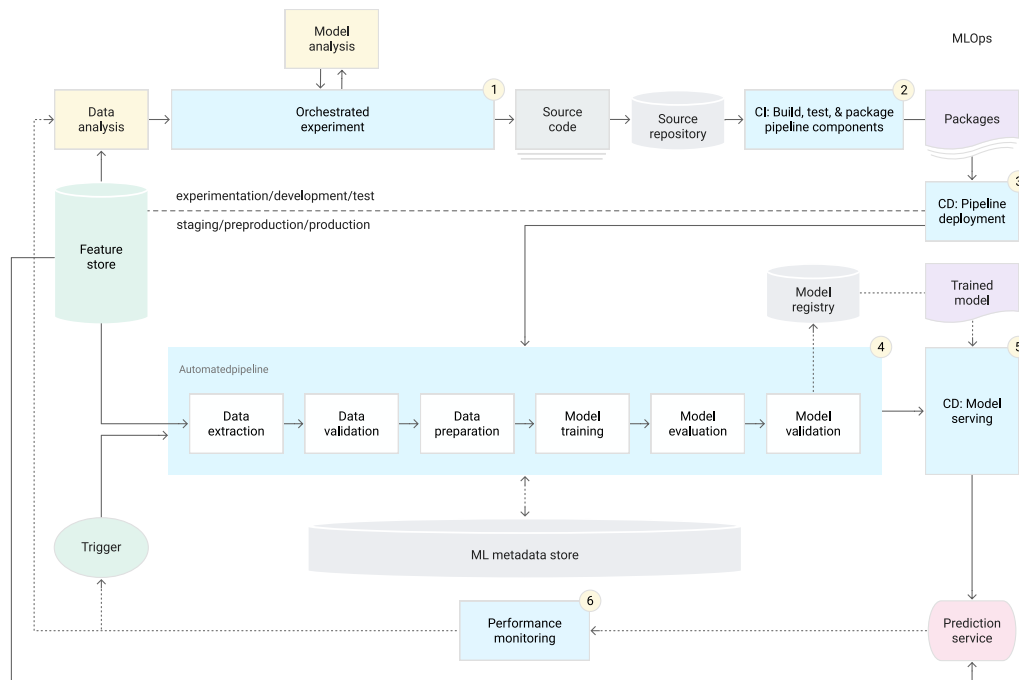
- Versiones de los componentes y del *pipeline*.
- Cuando empezó y termino la ejecución y el tiempo que llevo.
- Quien realizo la ejecución. - Parámetros usados en el *pipeline*.
- Punteros a los pasos del *pipeline*, localización de los datos preparados usados, anomalías en la validación, estadísticas y vocabulario extraído. Con el objetivo de poder reanudar una ejecución
- Un puntero al último modelo puesto en producción para poder volver a el si fuera necesario.
- Las métricas de evaluación producidas durante la ejecución de la evaluación del modelo. Para poder almacenarlas. Disparadores del *pipeline*.
- Bajo demanda, manualmente.
- De manera programática, teniendo en cuenta cada cuanto cambian los datos y cual es el coste de volver a entrenar los modelos.
- Cuando haya nuevos datos disponibles.
- Cuando el rendimiento del modelo sufra una degradación.
- Cuando los datos cambien de manera significativa

Opcionalmente se puede llegar a usar un almacén de características de los modelos. Tiene las siguientes ventajas.

- Evitar la creación de nuevos conjuntos de entidades que ya existan o que sean muy similares.
- Servir características actualizadas.
- Mantener unas características iguales entre la preproducción y la producción.

### Retos

Es necesario la gestión de todos los *pipelines* generados, lo que incluye los test que se realizan para comprobar su validez, su puesta en producción. Cuando son pocos puede ser gestionado manualmente, sin embargo, según crece su número es necesario que esto sea gestionado automáticamente.



**Figura 1.5.:** MLOps nivel 2

**MLOps nivel 2, *pipeline* CI/CD automatizado** En la figura anterior se muestra un posible *pipeline* MLOps nivel 2 [Clo21b].

En un entorno de nivel dos la entrega y la integración están automatizadas, lo que permite al equipo rápidamente explorar nuevas soluciones, modelos, arquitecturas e hiperparámetros.

Ha de estar conformado por los siguientes componentes:

- Control de fuente.
- Servicios para construcción, testeo e implementación.
- Registro de modelos. - Almacenamiento de características y de metadatos de *Machine Learning*
- Director de los *pipelines* de ML.

Pasos a seguir en el *pipeline*:

- Desarrollo e implementación de *pipelines* de ML, donde se prueban iterativamente nuevos algoritmos y modelos de ML.
- Integración continua del *pipeline*, que generara artefactos, paquetes y ejecutables para usar con posterioridad.
- Entrega continua del *pipeline*, implementación de los resultados de la anterior fase en el entorno real.

## 1.2 Objetivos

---

- Activación automática del *pipeline* producido, a través de una planificación o de manera reactiva.
- Entrega continua del modelo, los modelos producidos por el *pipeline* son expuestos a través de un servicio.
- Monitorización del servicio expuesto para que pueda ser usado como disparador del proceso.

Integración continua:

Al tener que automatizar los *pipelines* es necesario que el código fuente sea almacenado en un repositorio, el proceso de CI puede incluir lo siguiente:

- *Unit testing*, incluso distintas maneras de realizar un mismo paso.
- Testear las capacidades del modelo y las salidas erróneas del modelo.
- Testear que cada componente del *pipeline* genere los artefactos esperados.
- Testear la integración de los componentes del *pipeline*.

Entrega continua:

Para conseguir una entrega continua es necesario además en un repositorio los nuevos *pipelines*, sus paquetes y el nuevo código lo siguiente:

- Verificar la compatibilidad del modelo con la infraestructura sobre la que se desplegara (memoria, procesador, GPU...).
- Probar el servicio que sirve el modelo con todos los parámetros posibles para evitar peticiones que no podrán ser cumplidas.
- Testear el rendimiento del servicio, por ejemplo peticiones por segundo.
- Entorno de test automatizado o semiautomatizado, dependiendo de la metodología para incluir código en el repositorio, usar la gestión de pruebas automáticas.
- Inclusión manual en producción una vez se haya demostrado su buen funcionamiento en preproducción.

## 1.2. Objetivos

Este trabajo ha sido realizado para la empresa C.R.I.D.A. A.I.E., una agrupación de interés económico sin ánimo de lucro establecida por ENAIRE, la Universidad Politécnica de Madrid (UPM) e Ingeniería y Economía del Transporte, S.A. (INECO). Se trata de una empresa que realiza investigación sobre el tráfico aéreo y para ello en ocasiones recurren al machine learning y necesitaban de una metodología, que les permitiera agilizar dichos proyectos y conseguir llevar más de ellos a producción.

Los objetivos de este trabajo son los siguientes:

1. Análisis del problema. Es necesario estudiar el caso de aplicación de las metodologías MLOps para para C.R.I.D.A.
2. Análisis de las diferentes plataformas MLOps actualmente utilizadas. Debido a haber muchas herramientas actualmente para la gestión de proyectos MLOps es necesario realizar un estudio sobre ellas para elegir la mejor para la situación presentada.
3. Diseño de arquitectura, despliegue y configuración de la plataforma acorde a la metodología definida. Estrategia de *backup* y redundancia. Una vez seleccionada la herramienta es necesario construir una arquitectura para su correcto funcionamiento.
4. Definición de la metodología a seguir en un desarrollo de ML. Se definirá teniendo en cuenta la metodología MLOps y la plataforma escogida y desplegada.
5. Resultados: Aplicación de metodología a proyecto de ML acorde a su ciclo de vida. Una vez que se disponga de plataforma y metodología es necesario su prueba en un entorno real para comprobar su funcionamiento.
6. Conclusiones: Análisis de impacto sobre el desarrollo sostenible. Como ha funcionado la herramienta y la metodología diseñadas, su impacto en el mundo real.
7. Líneas futuras de mejora: Análisis de deficiencias y futuras mejoras. Teniendo en cuenta lo expuesto anteriormente se realizará un análisis de que podría ser mejorado.

## 2. Descripción del problema.

En el desarrollo de proyectos ML en la actualidad solo alrededor de un 10% [Rob20][Dic19] llegan a poner se en producción y es debido en parte a la gran cantidad de deuda técnica oculta que tienen este tipo de proyectos al usar librerías y *frameworks* de terceros. Una de las soluciones para poner remedio a este problema es la aplicación de metodologías de desarrollo software específicas para ML.

Para la utilización de metodologías MLOps nos debemos centrar en dos puntos principales. El primero es el registro de todos los procesos de entrenamiento de modelos, junto con todos los parámetros e información necesaria para dicho entrenamiento. El objetivo es favorecer el seguimiento y replicabilidad de los distintos entrenamientos realizados y escoger el modelo que mejores resultados obtenga. Por otro lado, el segundo punto se centra en establecer una metodología específica de desarrollo de modelos mediante la utilización de *pipelines* de CI/CD junto con una monitorización y gestión del ciclo de vida de los modelos. La aplicación de MLOps puede aumentar en un 30 % el valor del proyecto. [Com19].

Adaptar estas tecnologías tiene como retos tanto en su implementación como en su curva de aprendizaje [Ser22b].

- Gestión de proyectos:

No solo hay que gestionar un equipo de desarrollo, si no un equipo multidisciplinar que contiene perfiles tan dispares como desarrolladores, científicos de datos, expertos en el campo a tratar, personal no técnico (gestión). . .

- Comunicación y colaboración:

Adoptar un modelo de comunicación como el propuesto en la metodología DevOps.

- Todo es código:

- Se han de usar datos reales, los mismos que se usaran posteriormente en producción, para desarrollar los experimentos, las dependencias, reentrenamiento de modelos y métricas.
- Los modelos tienen ciclos de vida distintos a los servicios en los que se implementan.
- El sistema entero ha de ser reproducible mediante el código usado y sus artefactos. Todo queda definido como código, la infraestructura (IaC), configuración (CaC), los *pipeline* (PaC) para asegurar las estructuras CI/CD. El desarrollo se realiza en *pipelines* de ML y se puede

incluir *pipelines* de CI/CD para proveer de una mayor automatización. Hay que generar políticas que permitan al *pipeline* reducir la producción de resultados basados en datos poco fiables.

- Monitorización:
  - Las fases de *feature engineering* y entrenamiento de modelos necesitan recoger todas las métricas y los experimentos. El *tuning*, ajuste preciso de los algoritmos, requiere la manipulación de los datos, hiperparámetros y capturarlo sistemáticamente. Tener un registro de todos estos datos permite tener una reproducibilidad mayor.
  - Es necesario que los modelos puedan ser monitorizados a través de un interfaz para poder analizar su rendimiento.

### Beneficios de la MLOps

Conseguir desarrollar un sistema que cumpla lo anterior aporta lo siguiente a la organización:

- Productividad.
- Repetibilidad.
- Fiabilidad.
- Auditabilidad.
- Calidad de los datos y de los modelos.

En este caso se está desarrollando para la empresa C.R.I.D.A. A.I.E., una agrupación de interés económico sin ánimo de lucro establecida por ENAIRE, la Universidad Politécnica de Madrid (UPM) e Ingeniería y Economía del Transporte, S.A. (INECO) [CRI23].

Se trata de una empresa de investigación en el ámbito de la gestión de tráfico aéreo. Ya tiene experiencia con la utilización de modelos de ML en diferentes proyectos. Actualmente ha tenido dificultades para realizar una monitorización y un mantenimiento de los modelos. También a la hora de agilizar desarrollos de nuevos proyecto ML y en la reutilización de información de otros proyectos ML. Estos problemas se han visto acrecentados por el comienzo de numerosos proyectos que requieren la colaboración de equipos multidisciplinares en proyectos ML para los cuales se necesita una formación muy específica.



## 3. Análisis de las diferentes plataformas MLOps

Existe una gran variedad de herramientas para gestionar el ciclo de vida de proyectos ML siguiendo la metodología MLOps. Cada una de ellas tiene un proveedor distinto, una arquitectura distinta y sigue una arquitectura distinta. Cada una ofrece distintas ventajas e inconvenientes, que han de ser tenidos en cuenta.

### 3.1. Plataformas disponibles

#### 3.1.1. Amazon SageMaker

Es una plataforma gestionada en su totalidad por Amazon [Ser22a] como un servicio SaaS. Con un nivel básico gratuito y a partir de ese punto pago bajo demanda. Sus principales características están repartidas en módulos ejecutados en distintas máquinas, estando más limitados cuando más al principio del proyecto se encuentre el módulo. Cuenta con un almacén de características y monitorización de modelos que permite comparar el nuevo modelo con el ya existente. El desarrollo puede realizarse desde cuadernos Jupyter, RStudio conectados al servicio o desde la herramienta propietaria, y ser gestionados por la herramienta para diseñar *pipelines*. Cuenta con capacidad para realizar entrenamiento distribuido y facilidad para despliegue en dispositivos IoT (*Internet of Things*) o *Edge computing*.

#### Características que afectan directamente a MLOps:

- Ajuste de modelos automático.
- Piloto automático: Realiza un estudio automatizado de un *dataset*, y genera el mejor modelo posible.
- Wrangler: Se trata de una interfaz gráfica que permite la selección de datos e ingeniería de características. En la metodología MLOps corresponde con el *pipeline* de preparación de datos. Está conectado con Autopilot para la gestión poder lanzar entrenamientos directamente una vez se complete la selección de los datos.
- Depurator: Es un gestor de alertas y *debugger* que revisa el buen funcionamiento de los entrenamientos y de los modelos en producción. Permite la creación y gestión de alarmas y soluciones a medida.

**Características que no afectan directamente a MLOps:**

- Canvas: GUI para mostrar los datos al cliente final los resultados sin necesidad de grandes conocimientos de tecnología.
- Clarify: Aumenta la comprensión de los datos y de los modelos. Detecta sesgos en los modelos generados. Hace esto mediante la generación de informes sobre los modelos producidos.
- Ground Truth: Identifica datos sin procesar como imágenes, texto y videos. Además, permite generara datos sintéticos.
- JumpStart: Accede a repositorios de modelos para acelerar el desarrollo y evitar realizar modelos ya existentes.

**Pros:**

- Ecosistema completo de desarrollo con una aproximación *zero-code*.
- Diseñada agilizar el principio de los desarrollos.
- Acceso global gestionado por AWS
- Nivel 2 MLOps.

**Contras:**

- Es un servicio de pago, y al ser bajo demanda es difícil realizar un presupuesto, puesto que depende del uso que puede fluctuar.
- Es recomendable usar el sistema con todos sus componentes.
- Curva aprendizaje alta debido a la complejidad del entorno.

**3.1.2. Azure Machine Learning**

Ofrece un servicio [Mic22] desde el inicio hasta el final del ciclo de vida de los modelos y todas sus herramientas estas enfocadas en una parte de este proceso. Está diseñado para el uso con exclusivo de Python con librerías y *frameworks* de ML. Permite el despliegue de modelos dentro de contenedores, despliegue en nubes híbridas, parte en la nube de Azure u otra *in-premise* (dentro de los servidores de la propia empresa). Guarda los modelos en un registro de modelos. Optimiza modelos con ONNX (*Open Neural Network Exchange*) un proyecto de código abierto con el que colabora Microsoft. Ofrece también monetización de modelos ya en producción, deriva de los datos y análisis de errores.

**Características que afectan directamente a MLOps:**

- AutoML. Entrenamiento automático de múltiples modelos, con múltiples configuraciones para obtener de manera rápida un conjunto de modelos sobre los que poder partir.

### 3.1 Plataformas disponibles

---

- Canalizaciones y CI/CD. Herramienta para el diseño de los *pipelines* de CI/CD, están basados en las herramientas ofrecidas en los repositorios GitHub para realizar las mismas funciones.
- Repositorio de modelos. Almacén centralizado de los modelos generados, permite compartir modelos y realizar un seguimiento de ellos.
- Etiquetado y preparación de datos. Permite la automatización de tareas repetitivas de etiquetado de datos. Para la preparación de datos esta disponible el uso de Apache Spark y de Azure Synapse Analytics, la ETL de Azure.
- Supervisión y análisis de datos, modelos y recursos..
- Se controla la cantidad de desfase de datos que sufren los datos.
- Análisis de errores ocurridos en los modelos una vez en producción, permite su depuración.
- Auditoría de los artefactos generados durante el aprendizaje automático.

#### Características que no afectan directamente a MLOps:

- Imágenes precompiladas en contenedores para la el uso de macros y librerías.
- Posibilidad de desarrollo con *notebooks*, CLI (*Comand Line Interface*), *scripts* o herramientas graficas.
- Optimizar modelos a través de ONNX.

#### Pros:

- MLOps nivel 2.
- Aproximación *zero-code*.
- Acceso global mediante Azure y soluciones *in-premise*.

#### Contras:

- Es un servicio de pago, y al ser bajo demanda es difícil realizar un presupuesto, puesto que depende del uso que puede fluctuar.
- Es recomendable usar el sistema con todos sus componentes.
- Curva aprendizaje alta debido a la complejidad del entorno.

#### 3.1.3. Mlflow

Mlflow [mdcD22][dtD22c] es una plataforma de código abierto enfocada a la resolución de la problemática MLOps. Se instala como librería de Python, por lo

que se puede usar en cualquier máquina con una versión de Python superior a la 3.8, cuenta con APIs para Python, java y R, y una API REST para lenguajes no soportados de manera nativa. Dentro de su soporte a Python tiene soporte para librerías de ML bastante usadas como Scikit-learn, Keras, Gluon, XGBoost, LightGBM, Statsmodels, Spark, Fastai y Pytorch. Tiene cuatro módulos: *Tracking*, *projects*, *Models* y *Model Registry*.

- *Tracking*[dtD22e]: Permite el almacenamiento de versiones de código (identificador del *commit* en Git), tiempos de ejecución, fuente (código fuente), parámetros, métricas y artefactos (cualquier tipo de fichero producido por el entrenamiento o ejecución de los modelos). Todos estos datos, a excepción de los artefactos, pueden ser almacenados en bases de datos o sistemas de ficheros, los artefactos solo en sistemas de ficheros. Para las librerías de Python mostradas con anterioridad es posible realizar una captura automática. Mlflow organiza se organiza en runs (ejecuciones), cada ejecución individual que quede registrada y experimentos, conjuntos de experimentos bajo el mismo nombre de grupos definidos por el usuario.
- *Projects*[dtD22d]: Código empaquetado de una manera reusable y reproducible, también se incluye una API y un CLI para hacer posible introducir estos proyectos en el flujo de trabajo. Cada proyecto es un directorio o repositorio Git que contenga el código que se quiera ejecutar. Son llamados mediante *entrypoints*. Las dependencias pueden ser capturadas en entornos virtuales, Conda o Docker. Son definidos en un archivo YAML con el nombre MLproject. (<https://mlflow.org/docs/latest/projects.html#mlproject-file>).
- *Models*[dtD22b]: Formato estándar de empaquetamiento de modelos. Cada modelo es un directorio. Se permite distintos tipos de modelo. Cada modelo queda descrito en un archivo YAML con el nombre MLmodel. Mlflow es capaz de servir los modelos que tiene registrados. En el caso de que los modelos necesiten alguna dependencia más, automáticamente se añade al modelo sus dependencias. Para el uso de los modelos es necesario definir mediante metadatos los esquemas de entrada y salida de estos.
- *Model registry*[dtD22a]: Es el componente que centraliza los modelos guardados, las APIs y UIs para poder colaborar en la gestión del ciclo de vida de los modelos. Los modelos son creados al ejecutar un run dentro de un experimento. Los modelos se pueden registrar, esto hace que tenga un nombre único, versiones, estados de transición asociados, trazabilidad entre otros metadatos. Un modelo puede encontrarse en varios estados como *Staging*, producción y archivado. Es la herramienta para la gestión de ciclo de vida del modelo, se puede gestionar desde la API o desde una interfaz de usuario.

#### Pros:

- Flexibilidad en el desarrollo, se puede desarrollar con cualquier lenguaje y no es necesario que el lenguaje este soportado. Dentro de los soportados

### 3.1 Plataformas disponibles

---

hay funcionalidades bastante útiles como el Autolog para un desarrollo más cómodo.

- Incluye funcionalidades para poder realizar *pipelines* de ML y CI/CD.
- Código libre, gratuito, soportado por una comunidad activa.
- Incluido como solución MLOps por algunas soluciones para desarrollo ML de código cerrado.

#### Contras:

- La herramienta esta centrada en ser el servidor para alcanzar el nivel 0 de MLOps, pero ofrece herramientas para poder incrementar el nivel, alternativamente se pueden usar herramientas externas para lograr esto.
- No provee ninguna manera de saber la precisión de un algoritmo ya en producción.
- Anterior a la versión 2.3.0 no hay autenticación de usuarios [CD23].

#### 3.1.4. Databricks

Databricks [Dat23] es una plataforma para gestión de *data warehouse* y *data lakes* que se ofrece como PaaS, que puede ser desplegado en Azure, Google Cloud o AWS. Se basa en el uso de proyectos de código libre con los cuales contribuye. Mlflow es su herramienta para la gestión de MLOps, del que ademas es su principal contribuyente.

Añade una capa de automatización por encima de Mlflows, para su gestión automática, guarda todos los experimentos, artefactos y parámetros. Gestiona el ciclo de vida de los modelos. Añade las siguientes funcionalidades a Mlflow:

#### Características que afectan directamente a MLOps:

- Almacenamiento de características para su correcta reutilización en entrenamientos y servicios de modelos.
- AutoML, realiza un estudio para generar el mejor modelo posible y el código para conseguirlo para poder se refinado posteriormente.
- Mlflow autogestionado, para poder pasar modelos entre fases del ciclo de vida con facilidad.
- Servicio de modelos.
- Monitorización de modelos, proporciona métricas en tiempo real del funcionamiento de los modelos para poder reentrenar de nuevo los modelos y saber cuándo retirar modelos.
- Repositorios para permitir CI/CD automatizada.

**Características que afectan directamente a MLOps:**

- *Notebooks* colaborativos.
- Acceso a *cluster* para el entrenamiento ya preconfigurados y optimizados para la utilización de *frameworks* de ML.

**Pros:**

- MLOps nivel 2.
- PaaS gestionado por una nube empresarial.
- Permite despliegue en varias nubes.

**Contras:**

- Es un servicio de pago, y al ser bajo demanda es difícil realizar un presupuesto, puesto que depende del uso que puede fluctuar.
- Está diseñado para usar el ecosistema completo.

**3.1.5. Domino**

Domino [Lab22] es una plataforma para el desarrollo de ML. Es un PaaS/SaaS de pago con precio personalizado. Es independiente de la nube usada y permite gestión *in-premise* (en los servidores de la propia empresa), además, permite intercomunicación entre nubes. Tiene tres módulos principales: registro del sistema, factoría de modelos integrada y portal de autoservicio de infraestructuras. Utiliza Mlflow como base para la recogida de experimentos y artefactos.

- Registro del sistema: Mantiene un control de versiones del código, los datos las herramientas y los paquetes. Incorpora objetivos, integrado con Git y Jira, a la vista de los modelos para poder realizar un seguimiento efectivo.
- Factoría de modelos integrada: Manejo sencillo del ciclo de vida de los servicios. Pueden ser desplegados a distintas arquitecturas como APIs escalables, apps, AWS Sagemaker, Docker images para generar *pipelines* de CI/CD. Revisa la salud de los modelos y la deriva de estos con el tiempo.
- Portal de autoservicio de infraestructuras: Gestión de las herramientas necesarias para la creación y despliegue de modelos. Permite la selección de distintos *cluster* de ordenadores para el entrenamiento y de distintos entornos de desarrollo. El acceso a los datos se realiza desde un entorno seguro.

**Pros:**

- MLOps nivel 2.

### 3.1 Plataformas disponibles

---

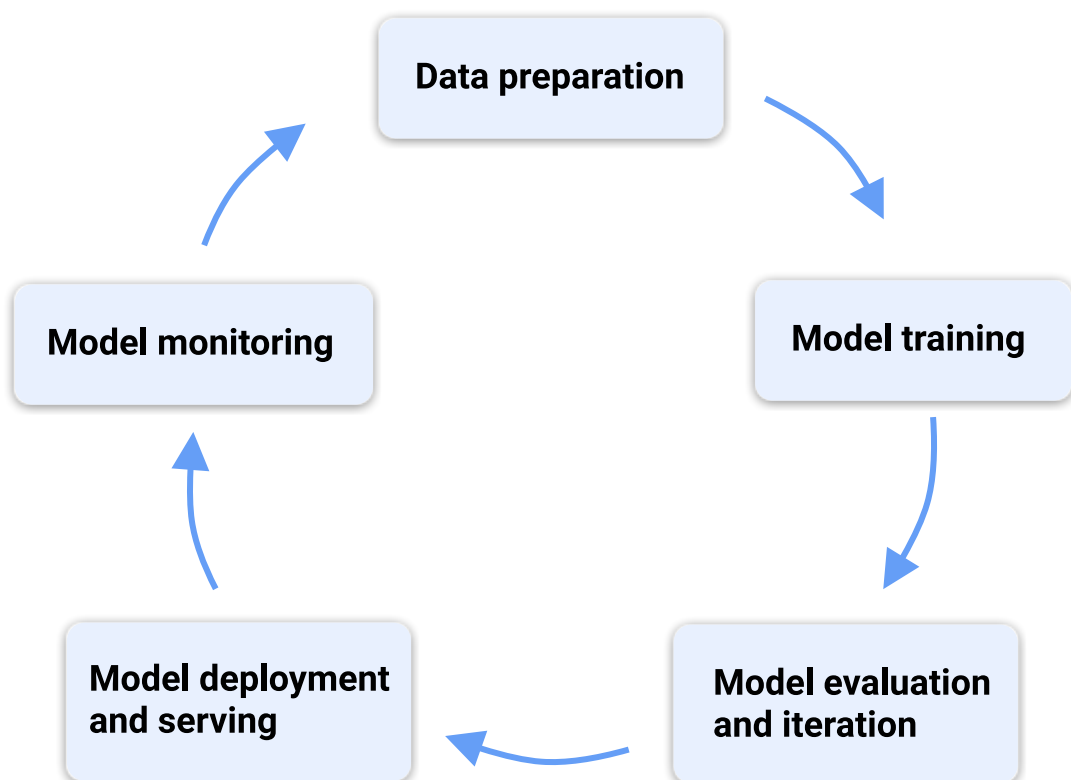
- Posibilidad de uso de varias nubes incluyendo los servidores de la propia empresa, lo que reduce la dependencia de un solo proveedor de este tipo de servicios. Se puede desplegar en cualquier nube.

#### Contras:

- Servicio de pago.
- Sus ventajas son bastante similares a las ofrecidas por Mlflow.

#### 3.1.6. Vertex AI (Google Cloud)

## Machine learning workflow



**Figura 3.1.:** Ciclo de vida Vertex AI [Clo21a]

En la figura 3.1. se muestra el ciclo de vida propuesto por la herramienta Vertex AI.

Vertex es la herramienta MLOps dentro del ecosistema de Google [Clo21a]. Está diseñada para su uso en combinación con el resto del ecosistema de Google para ML. Utiliza Auto-ML, para generar de manera automático modelos para encontrar el mejor y ajustarlo con posterioridad. Gestiona el ciclo de vida con *pipelines* automatizados. Contiene modelos preentrenados para ciertas ramas. Su aproximación es a través de un flujo circular de trabajo como se muestra en la anterior figura, preparación de los datos, entrenamiento del modelo, evaluación del modelo, servicio del modelo y monitorización del modelo. Se organiza las a realizar automáticamente desde Vertex AI Pipelines, las métricas, parámetros y artefactos son almacenados en metadatos y tratados por Vertex ML Metadata. La gestión de modelos versiones se realiza desde Vertex AI Model Registry, de los atributos desde Vertex AI Feature Store y puede ser monitorizado desde Vertex AI Model Monitoring. Tiene las siguientes características:

#### Características que afectan directamente a MLOps:

- Vertex AI Workbench. Herramienta para el entrenamiento de los modelos, se permite el uso de *notebooks*, *scripts* o CLI (*Command Line Interface*).
- Vertex AI Feature Store. Permite el almacenamiento de *features* de manera reusable entre proyectos para evitar trabajo duplicado y previene la aparición de deriva de los datos.
- Vertex AI Model Monitoring. Alerta automáticamente de la deriva de los datos u otros fallos en los modelos ya puestos en producción.
- Vertex AI Pipelines. Construcción de *pipelines* de ML y CI/CD necesarios para los niveles 1 y 2 de MLOps. Usa TensorFlow extended o KubeFlow.
- Vertex ML Metadata. Facilita las tareas de auditabilidad y gestión de proyectos a través de los metadatos que definen todos los componentes usados.
- AutoML. Entrenamiento automático de múltiples modelos, con múltiples configuraciones para obtener de manera rápida un conjunto de modelos sobre los que poder partir.

#### Características que afectan directamente a MLOps:

- Vertex Explainable AI.
- Vertex AI Training. Herramienta para el entrenamiento de modelos.
- Vertex AI Vizier. Optimizador de modelos
- Vertex AI Data Labeling

#### Pros:

- MLOps nivel 2.



### 3.1 Plataformas disponibles

---

- Muy enfocado en el ciclo de vida, influenciado por su artículo sobre el motivo de los fallos en ML.
- Disponible en cualquier lugar, totalmente gestionado desde la nube.

#### Contras:

- Servicios de pago por uso, cada uno por separado
- Es necesario el conjunto de herramientas.

#### 3.1.7. KubeFlow

Es una herramienta [GC22] de código abierto para la gestión de proyectos de ML a través de Kubernetes [Com23], una plataforma para la gestión de contenedores distribuidos de manera centralizada. Kubeflow y Kubernetes han sido desarrolladas por Google. Permite tener total control sobre como gestionar cada paso del ciclo de vida: preparación de datos, entrenamiento de modelos, servicio de modelos y gestión del servicio. Se organiza en los siguientes componentes:

#### Características que afectan directamente a MLOps:

- Central Dashboard: gestiona la herramienta, desde aquí se puede gestionar las modelos, experimentos (tanto AutoML, como KubeFlow pipelines), *pipelines*, runs, artefactos, ejecuciones y manejo de contribuciones. Se pueden añadir aplicaciones personalizadas a través de Kubernetes.
- Kubeflow pipelines: gestiona el diseño de los flujos de trabajo, los flujos son *scripts* de Python o componentes en contenedores. Son definidos mediante archivos YAML.
- Katib: herramienta AutoML

#### Características que no afectan directamente a MLOps:

- Training Operators: componente encargado de los entrenamientos tiene soporte nativo para las siguientes librerías: TensorFlow, PaddlePaddle, PyTorch, MXNET, XGBoost y MPI.
- Kubeflow notebooks: es la herramienta para el desarrollo dentro de Kubeflow, permite la inclusión de recursos propios a través de Kubernetes.

#### Pros:

- Código libre.
- Centrado en la automatización de los *pipelines* de ML y CI/CD.
- Basado en Kubernetes.

**Contras:**

- Necesita de un equipo de desarrollo que conozca el funcionamiento de contenedores y Kubernetes.
- Provee de las herramientas necesarias para usarlo en cualquier nivel MLOps pero no lo ofrece en su base.

**3.1.8. H2O MLOps**

Es la plataforma para el tratamiento de la problemática MLOps de H2O.ai [H2O22]. Dentro de las soluciones que provee H2O.ai esta es de pago, se puede desplegar en la nube o en una nube híbrida. Consiste en cuatro módulos.

- **Model Management:** Muestra los resultados de todos los experimentos realizados y almacenados en la herramienta. Pueden ser comparados en una tabla clasificatoria, mediante sus métricas y distintos metadatos recogidos.
- **Model Deployment:** Permite el despliegue de modelos a través de un API REST. Pueden ser desplegados de las siguientes maneras: Modelo único; A/B test, parte de los datos son desviados para testear el modelo B; o campeón contra contendiente, se comprueba si el modelo activo es el mejor en un momento dado.
- **Model Governance:** Maneja los datos, artefactos, experimentos, modelos, despliegues y aporta trazabilidad. Permite gestionara el ciclo de vida de todos los modelos, otorga los permisos a cada usuario o grupo determinado que pueden hacer.
- **Model Monitoring:** Monitoriza los modelos ya puestos en producción para comprobar su deriva, precisión y anomalías en tiempo real. Permite configurara alertas con las métricas de los modelos.

**Pros:**

- Aunque la herramienta es de pago gran parte de su ecosistema es código abierto.
- Permite despliegue en varias nubes o incluso la nube híbrida.
- Permite definir los roles de cada usuario.

**Contras**

- Es un servicio de pago.

### 3.2. Selección de plataforma

Plataforma	Código abierto/ cerrado	Nivel MLOps	Localización de despligue	Curva aprendizaje
Amazon SageMaker	Cerrado	2	Nube	Pronunciada
Azure Machine Learning	Cerrado	2	Nube/ <i>In-premise</i>	Pronunciada
Mlflow	Abierto	0 (Compatible con cualquier herramienta para subir el nivel)	<i>In-premise</i> /Nube	Baja
Databricks	Cerrado, automatización de Mlflow (abierto)	2	Nube	Media
Domino	Cerrado, automatización de Mlflow (abierto)	2	Nube/ <i>In-premise</i>	Media
Vertex AI	Cerrado	2	Nube	Pronunciada
KubeFlow	Abierto	0 (Compatible con cualquier herramienta para subir el nivel)	<i>In-premise</i> /Nube	Pronunciada
H2O MLOps	Cerrado	2	Nube/nube híbrida	Media

**Cuadro 3.2.:** Selección herramienta

Este proyecto no cuenta con presupuesto ninguno, ni para comprar licencias software ni para usar sistemas de pago por demanda. Las dos plataformas que quedan permiten la gestión de MLOps nivel 0, Mlflow y Kubeflow. Ambos proveen un software de código abierto altamente configurable, que puede ser desplegado en cualquier lugar, nivel MLOps 0, es decir solo el almacén centralizado de modelos, artefactos, métricas y parámetros.

Mlflow provee un sistema flexible, se puede usar con cualquier lenguaje o libre-

ría, provee la base la gestión en distintos de métricas, parámetros y artefactos; es necesario realizar la arquitectura necesaria para guardar todos los datos generados. Su curva de aprendizaje es baja, solo es necesario aprender el uso de una librería Python, solo se ha de añadir un conjunto pequeño de líneas al código original.

Por el contrario, Kubeflow también es flexible, todo se hace a través de contenedores, pero es necesario tener conocimientos sobre contenedores y Kubernetes.

Se ha seleccionado Mlflow debido a su flexibilidad, facilidad para aprender a usarlo y de uso general. Provee una documentación extensa sobre su uso, soporte a las librerías de ML más usadas, dando soporte en nativo a Python, java y R, y por medio de API al resto de lenguajes.

## 4. Diseño, despliegue y configuración de la plataforma.

### 4.1. Requisitos

Para conseguir implementar la herramienta Mlflow seleccionada es necesario diseñar una arquitectura que cumpla lo siguiente:

- Servidor Mlflow.
- Repositorio para guardar los artefactos necesarios para el entrenamiento y los generados por los modelos.
- Base de datos para almacenar métricas e hiperparámetros y la gestión de la localización de los artefactos.

### 4.2. Tecnologías usadas

#### 4.2.1. Docker

Docker es una tecnología para la creación de contenedores. En comparación con las máquinas virtuales (VM) tradicionales la virtualización se produce a nivel de sistema operativo en vez a través de un hipervisor, esto quiere decir que son más ligeros y eficientes que las VMs. Elimina las dependencias con las librerías del sistema operativo. Su uso está muy extendido en la aplicación de DevOps porque unifica el entorno de trabajo para poder mantener preproducción y producción. Docker es el gestor de contenedores más utilizado y extendiendo, además es código abierto y cuenta con una comunidad bastante activa. Cuando se usa para desplegar múltiples servicios, como es el caso se usa una arquitectura de microservicios, cada módulo de un proyecto se ejecuta desde un contenedor distinto.

**Herramientas Docker usadas:**

- Docker networking - Published ports [Doc22b]: Permite la gestión y vinculación de los puertos virtuales de cada contenedor con los de la máquina real.
- Docker storage - Volumes/Bind mounts [Doc22c] [Doc23]: Permiten almacenamiento persistente en los contenedores. Montan parte del sistema de archivos real, manejado por Docker o por el sistema operativo respectivamente, en parte de sistema de archivos virtual del contenedor.

- Docker compose [Doc22a]: Docker compose es la funcionalidad que se proporciona de manera nativa para poder gestionar el despliegue de múltiples contenedores simultáneamente. Permite recoger todas las opciones que se le darían al comando *docker run* para una mayor replicabilidad y facilidad en su uso. Aquí se pueden definir los volúmenes a usar, puertos, opciones de relanzado, variables de entorno pasadas para configurar el sistema, dependencias en el orden de arranque. . .

#### 4.2.2. HDFS

Hadoop Distributed File System [The21] es un sistema de ficheros totalmente distribuido de código abierto, basado en los artículo de Google sobre su sistema de ficheros (GFS) [GGL03] y MapReduce [DG04].

HDFS es un sistema de ficheros diseñado para funcionar en sistemas con pocos recursos, de manera tolerante a fallos, optimizado para el acceso a ficheros grandes, siguiendo el estándar POSIX. Sus objetivos son los siguientes:

- Recuperación rápida de errores *hardware*.
- Sistema enfocado al flujo de datos. Se prefiere una gran cantidad de datos transferidos a una baja latencia.
- Preparación para grandes ficheros.
- Modelo simple de coherencia, *write-once-read-multiple*. Solo se permite la escritura una sola vez para evitar bloqueos, un fichero solo se puede ampliar o borrar.
- “Mover la computación es más barato que mover datos”. Es preferible hacer que el código se ejecute desde otra máquina que mover cantidades ingentes de data para realizar cálculos.
- Portabilidad.

#### 4.2.3. NFS

NFS [Cos22] (*Network FileSystem*) es un protocolo para la distribución de sistemas de ficheros desarrollado por SUN. Esta diseñado para su uso en entornos heterogéneos. El concepto que sigue es montar una rama del sistema de ficheros de otra maquina en su propio sistema ficheros local. Para conseguir esto el i-nodo local apunta al i-nodo remoto (r-nodo) que se quiera compartir. Los ficheros se almacenan en el servidor NFS.

#### 4.2.4. SMB

SMB (*Server Message Block*) es un protocolo para compartir archivos en red. Para sistemas Windows es el protocolo estándar que proporciona este servicio. Se usa como sustituto de NFS en ordenadores Windows. El sistema de ficheros importado se muestra como un volumen.

### 4.2.5. Entornos virtuales

En Python por defecto los paquetes se instalan en un repositorio común para todo el sistema. Esto puede causar problemas con la compatibilidad de librerías, pues distintos proyectos pueden necesitar de distintas versiones. La función de los entornos virtuales es encapsular las dependencias de manera local al proyecto. Otra de sus ventajas la capacidad de replicación de los entornos de desarrollo, proveen de una manera de encapsular las dependencias de las librerías usadas. Las dos implementaciones más comunes son la librería venv [Fou22] y los entornos de Conda [Ana21].

Se suelen usar junto con contenedores para gestionar las dependencias tanto del sistema operativo (contenedores), como de las librerías usadas (entornos virtuales).

## 4.3. Diseño arquitectura

### 4.3.1. Metodología

La arquitectura se ha diseñado y probado con una aproximación *bottom-up*. Se ha ido añadiendo incrementalmente distintos elementos al sistema empezando por la base y construyendo desde ahí.

### 4.3.2. Mlflow

Mlflow es una librería de Python con bastantes dependencias de versiones específicas de paquetes comunes en Python, por lo tanto, para su gestión es necesario el uso de entornos virtuales para evitar conflictos entre paquetes. Mlflow no cuenta con dos paquetes separados para realizar la función de cliente y de servidor por la gran cantidad de formas de las que puede ser desplegado. Esto significa que el cliente tiene toda la funcionalidad del servidor y por lo tanto en algunos casos realiza acciones normalmente realizadas por el servidor.

Para garantizar independencia del sistema operativo y realizar una arquitectura replicable se usa Docker. Para poder usar los entornos virtuales que se usaran para la gestión de dependencias es necesario que se encuentren activas, es decir que se use la instancia de Python con las librerías adecuadas y diseñada para el proyecto. Los contenedores Docker utilizan por defecto el superusuario, por lo tanto, es recomendable usar un usuario no con los permisos mínimos y necesarios para gestionar el contenedor.

Mlflow se despliega como servicio web, por lo tanto, es necesario indicarle cual va a ser su puerto y a que maquinas puede servir. Se quiere servir a todas las maquinas que tengan acceso a este servidor (0.0.0.0), a través del puerto 80 (Estándar para servicios http). Los contenedores despliegan sus servicios sobre un puerto solo visible desde el interior del contenedor, Docker proporciona una manera de vincular puertos reales y virtuales. Por lo tanto, el comando básico para el despliegue es el siguiente: `mlflow server -h 0.0.0.0 -p 80`.

### 4.3.3. Registro de experimentos, métricas e hiperparámetros

Mlflow usa puede usar un sistema de ficheros o una base de datos para almacenar experimentos, métricas e hiperparámetros. Una base de datos proporciona cualidades para el mantenimiento de datos en un entorno concurrente mejor que un sistema de ficheros datos que un sistema de ficheros.

Dentro de las bases de datos Mlflow, a través de la librería SQLAlchemy, permite realizar conexiones con las siguientes: MySQL, MariaDB, Oracle, MSSQL (Microsoft SQL), SQLite y PostgreSQL. De estas se pueden descartar las siguientes por ser de pago Oracle y MSSQL. SQLite es una base de datos que almacena toda su información en un único fichero y su principal objetivo es tener un código que ocupe muy poco en el almacenamiento y muy eficiente en su ejecución; a costa de funcionalidades más avanzadas. Como se trata de un sistema que haya de ejecutar con pocos recursos de almacenamiento o de procesamiento no es recomendable su uso. De los gestores de bases de datos restantes MariaDB es el proyecto de código libre de que se creo tras la compra de Oracle a MySQL, antiguo proyecto de código libre y PostgreSQL, o Postgres, es un proyecto iniciado en la Universidad de Berkeley. Las tres tienen propiedades ACID (Atomic, Consistency, Isolation, Durability), están entre las más usadas según DB-Engines [DE23]. Por lo tanto, cualquiera de las tres es una buena opción para usarla como base de datos, y se ha escogido Postgres.

Los contenedores no tienen almacenamiento persistente de datos, por lo que es necesario añadir un volumen para poder recuperar la información ya guardada en el caso que se pare o caiga el contenedor. Para las bases de datos dockerizadas es el componente más importante añadir. La conexión a una base de datos se realiza sobre un puerto que también hay que asignar para poder realizar conexiones desde el exterior del contenedor y vincularlo con un puerto real de la máquina para permitir el acceso desde el exterior de la máquina.

### 4.3.4. Registro de artefactos

En Mlflow todo aquello que se genere durante el entrenamiento y construcción del modelo y no pueda ser almacenado en una base de datos, se almacena en un sistema de ficheros, que puede o no ser distribuidos. Mlflow puede almacenar información en los siguientes sistemas o a través de los siguientes protocolos: Amazon S3, Azure Blob Storage, Google Cloud Storage, FTP/SFTP, NFS, HDFS y sistema de ficheros local. Amazon S3, Azure Blob y Google Cloud Storage son sistemas de ficheros almacenados en las nubes de cada una de las empresas y son servicios de pago. Mlflow configurado de esta manera permite al cliente transmitir los archivos de manera directa a la ruta especificada por el servidor, pero lo realiza usando el propio usuario del cliente y con un grupo desconocido por lo que el servidor es incapaz de acceder a estos. Usar el sistema local de ficheros no es posible para una arquitectura distribuida. NFS permite al cliente guardar los artefactos como si el repositorio de artefactos fuera local a su máquina cuando está en otra localización, por el contra es necesario que cliente y servidor tengan la misma localización para el repositorio, solo se define una



localización de este al lanzar el servidor a través del comando. Para sistemas Window, que no permiten el uso del protocolo NFS es necesario el uso de SMB, que monta los sistemas de ficheros como volumen y que posteriormente hay que sincronizar con la carpeta que se quiere usar. HDFS es la mejor por su resistencia a fallos y su diseño totalmente distribuido para este tipo de arquitecturas, además de su facilidad para el tratamiento de datos de manera distribuida con MapReduce.

Por todo lo anterior se optó inicialmente por HDFS por tener mejores características para un sistema estable y a prueba de fallos. Debido a problemas al realizar el montaje del servidor HDFS y su uso junto a Mlflow ha sido necesario utilizar NFS/SMB.

### 4.4. Implementación arquitectura

Para realizar la arquitectura se ha usado un conjunto de Dockerfiles orquestados por un Docker compose, disponibles en el anexo de la memoria. Además, se ha realizado un manual de instalación, en el que queda documentado el proceso a seguir, que también se encuentra en el anexo.

El primer paso necesario para el despliegue de la arquitectura es la instalación de Docker, Docker compose y Docker engine. Para comodidad posteriormente se puede añadir el usuario que ejecute esto a Docker.

#### 4.4.1. Mlflow

Se parte del contenedor base de Ubuntu su versión 22.04, la versión estable publicada el año pasado. Los contenedores solo ofrecen las funciones mínimas e indispensables para su funcionamiento, por lo que es necesario instalar toda librería que se vaya a necesitar. En este caso se va a necesitar python, wget y libpq-dev (dependencias para usar Postgres).

Como se ha comentado con anterioridad, para poder usar los entornos virtuales es necesario que estén activos. Para conseguir esto con contenedores es necesario realizar una configuración previa del terminal a usar, puesto que cada ejecución realizada dentro del Dockerfile, la sentencia RUN, además deben ser activados automáticamente para realizar cualquier ejecución posterior. Por comodidad se usará un terminal bash, en vez del sh estándar para los contenedores. Como argumentos se usará `--login` para forzar la ejecución de los archivos `.bashrc` y `.profile`, en los que se encuentra la información necesaria para poder activar los entornos. Por seguridad se debe usar un usuario sin los privilegios de super-usuario. Este usuario solo tendrá los permisos mínimos y necesarios para las ejecuciones que debe realizar, Este usuario no tendrá una contraseña definida.

Los ficheros `entrypoint.sh` y `.bashrc` han sido definidos previamente en la máquina real. Son copiados en el interior del contenedor y el fichero `entrypoint.sh` convertido en ejecutable.

A partir de este momento se activa el usuario para realizar el resto del montaje. Aunque se pueda usar Conda o venv para la gestión de paquetes, es más sencillo el uso de Conda en combinación con los contenedores. Se descarga Miniconda. Para poder usar Conda sin tener que la ruta completa se añade a la variable PATH. Se modifica el archivo `.profile` para poder activar entorno Conda, es necesario inicializarlo para Conda.

Se actualizan los repositorios Conda y crea el entorno necesario para la ejecución. Como todos los RUN se ejecutan distintas instancias del terminal, es necesario que las librerías se instalen en una misma ejecución. Se actualiza pip a su última versión, se instala mlflow, psycopg2-binary (dependencias de Postgres en Python) y la librería para la gestión de HDFS desde Python.

Para la gestión los accesos al contenedor se usa el fichero `entrypoint.sh`. En este fichero hay dos instrucciones, la primera hace que la ejecución termine en el caso de encontrarse un error; la segunda activa el entorno de conda y ejecuta todo lo que se pase como argumentos a este *script*.

#### 4.4.2. Postgres

Para desplegar una base de datos Postgres en Docker solo es necesario importar la imagen base oficial.

#### 4.4.3. HDFS

Para el despliegue de un servidor HDFS se han probado dos imágenes: la primera pravega/hdfs [Pra19a] bajo licencia MIT y su código está almacenado en el repositorio [Pra19b]; y la segunda la oficial apache/hadoop [Had21] con su código almacenado en el repositorio [Had23] bajo licencia Apache.

Se comenzó probando la primera de las dos imágenes, esta despliega un contenedor con HDFS disponible, pero el sistema no tiene una raíz del sistema de archivos HDFS ni una ruta donde se almacena definidas. Para definir estos dos atributos es necesario modificar el archivo que describe el servidor: `hdfs-site.xml`. Se ha de añadir una propiedad `dfs.datanode.data.dir` para definir la localización del nodo de datos dentro del sistema local de archivos, para que pueda ser almacenado en una ruta conocida, para posteriormente poder añadir un almacenamiento persistente.

Además, según la documentación de Mlflow es posible la conexión mediante usuarios definidos en el sistema Unix que soporta HDFS, o mediante un Kerberos. Para añadir la identificación a través de usuarios es necesario activarla en el fichero `core-site.xml`. Se cambian las propiedades `hadoop.proxyuser.hadoop.hosts` y `hadoop.proxyuser.hadoop.groups` a `*` para aceptar a todos los usuarios. Además de esto se creó un directorio en HDFS con dueño y grupo mlflow y permisos `drwxrws_`. Esto provoca que tengan acceso el dueño de los ficheros, y cualquiera que comparta el grupo Mlflow, además los archivos se guardaran con el grupo Mlflow.

## 4.4 Implementación arquitectura

---

Con esta configuración se realizaron varias pruebas de conexión no satisfactorias, se intentó probar otra configuración del sistema con la imagen oficial. Con esta todo el sistema de ficheros era hdfs lo que no era muy conveniente por tener que gestionar el acceso a los ficheros de la misma manera, es necesario crear los ficheros necesarios con un super-usuario y después cambiar el dueño y grupo a mlflow con los permisos `drwxrws__`.

Teniendo las dos imágenes disponibles en cuanto a conectividad sin tener en cuenta Mlflow la imagen pravega/hdfs funciona mejor. Sin embargo el problema para la conexión no estaba en el contenedor de HDFS, si no en la forma de en la que se gestiona la conexión desde Mlflow. Aunque en el manual se afirme que la conexión a través de usuarios UNIX es posible, si se revisa el código fuente para la conexión solo se contempla la conexión mediante un Kerberos intermedio.

### 4.4.4. NFS

Debido a los problemas anteriores se ha decidido no usar HDFS en favor de NFS. Mlflow no tiene una manera para el uso de NFS nativo, se usa la conexión que se usa para ficheros locales. Como se ha comentado antes Mlflow no tiene dos paquetes distintos para realizar acciones de cliente y servidor, por lo que es el propio cliente el que guarda los artefactos, por lo que solo se define una única ruta para la gestión de artefactos. Se a de usar la misma ruta global para ambos sistemas para que se puedan ver el repositorio.

### 4.4.5. SBM

NFS no esta disponible en dispositivos Windows, fuera de las versiones Pro. Por lo que es necesario encontrar un sustituto, SMB tiene soporte nativo para Windows. SMB solo permite montar el sistema remoto como un volumen del sistema (D:, F:, Z:...). Por lo tanto, para poder definir la ruta completa es necesario sincronizar las carpetas se usa `mklink`, que sería el equivalente al comando `ln`, pero que permite realizar un enlace entre directorios.

### 4.4.6. Despliegue conjunto

Para realizar el despliegue de todos los elementos se usará la herramienta `compose` dentro de Docker.

#### Mlflow

Para el despliegue de este contenedor es necesario que la base de datos Postgres ya esté levantada, se define que el puerto 80 de la red interna del contenedor corresponderá con el 80 en la red externa. El contenedor será levantado automáticamente después de cualquier fallo o parada no voluntaria, además de cuando se inicie la maquina real a través del demonio de Docker. Se monta dentro del contenedor y en la misma ruta que en la maquina real un volumen para permitir

la comunicación entre el servidor de Mlflow dentro del contenedor y el servidor NFS donde se almacenan los artefactos.

En cuanto al comando para lanzar el sistema se utilizará el siguiente: *mlflow server -h 0.0.0.0 -p 80 -backend-store-uri \$BACKEND -default-artifact-root \$ARTIFACTS/artifacts*.

Se puede ver que el servidor se despliega para todos los usuarios dentro de la misma red, a través del puerto 80. Se están usando un para definir *backend-store-uri* y *default-artifact-root* unas variables de entorno, esto es por seguridad, la primera define la conexión a la base de datos y la segunda al repositorio de artefactos. Estas variables son almacenadas en el archivo *.env*. En este archivo la contraseña de la base de datos se obtiene de la variable de entorno en la máquina real que la define. Para desplegar el servidor se crea la variable de entorno, una vez levantado se puede retirar la contraseña de las variables de entorno de la máquina real.

### Postgres

Para el correcto funcionamiento del gestor de bases de datos es necesario definir mediante el entorno la base de datos que se va a usar, el usuario que se va a usar, la contraseña siguiendo la misma técnica que antes para que no quede registrada. Además, para poder usar un volumen para el almacenamiento persistente de los datos es necesario modificar el almacenamiento interno dentro del contenedor a un subdirectorío para evitar el siguiente problema [sta21]. El volumen se sitúa sobre la antigua ruta de los datos.

### Repositorio de artefactos

Se creará una carpeta compartida en un NAS preexistente para exportar

## 4.5. Estrategía de *backup* y redundancia

### 4.5.1. *Backup* servicios

En el caso de que uno de los contenedores caiga esta configurado para que vuelva a iniciarse de manera automática, en el caso que sea la máquina real la que caiga, el demonio de Docker vuelve a activar el servicio Docker en el arranque y levantar los contenedores que sea necesarios.

### 4.5.2. *Backup* de datos

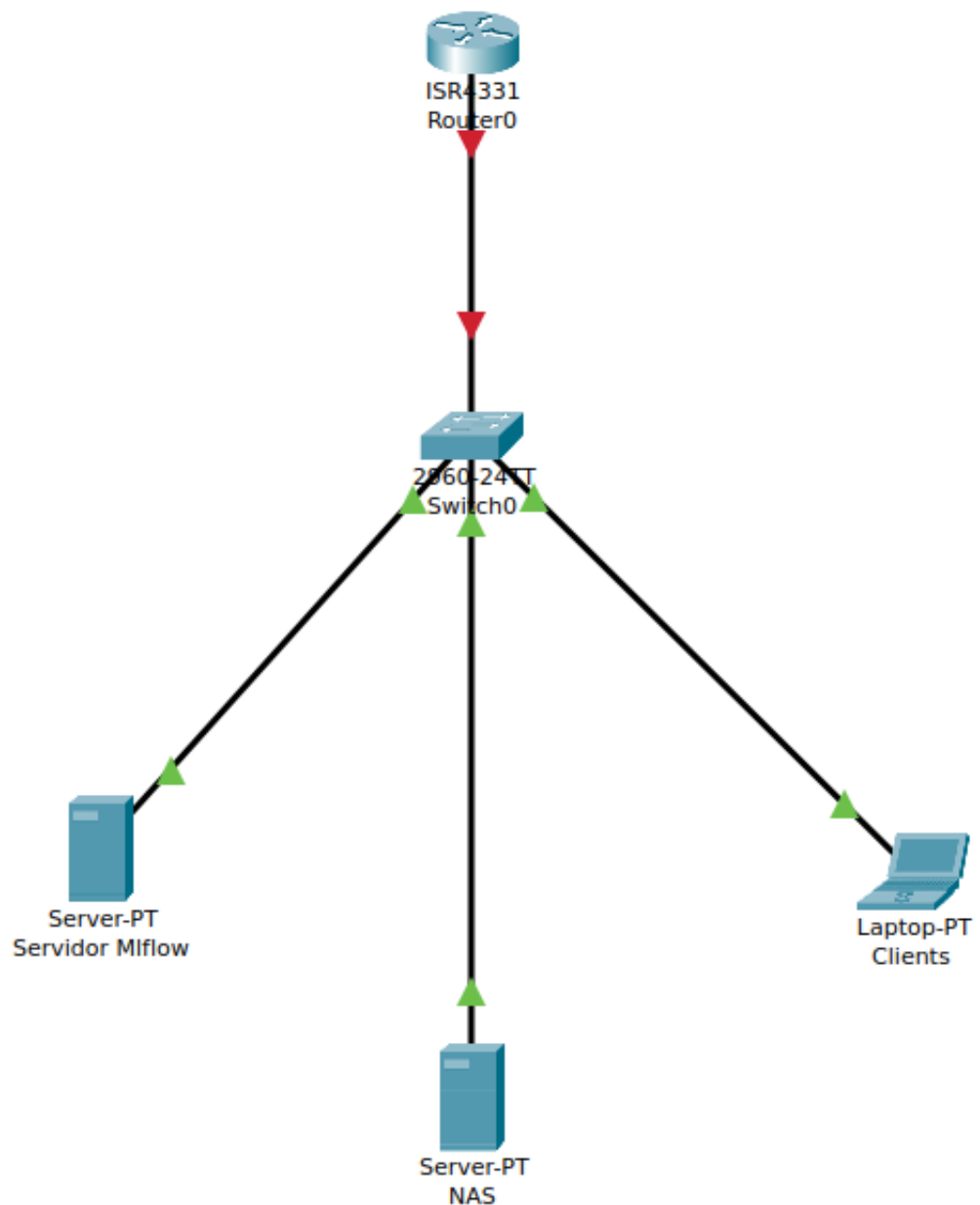
Por facilidad cuando se diseñó propuso el uso de NFS y de SMB, se usó un NAS (NAS Synology DSM) con capacidad para compartir archivos en estos dos formatos, dicho sistema NAS está preparado para tener backups incrementales cada semana. La base de datos Postgres se optó por montar el volumen de manera

#### 4.5 Estrategía de *backup* y redundancia

---

que estuviera gestionado por el propio sistema de ficheros y se ha planeado un backup incremental semanal de los datos ahí presentes.

#### 4.6. Esquema final de la arquitectura

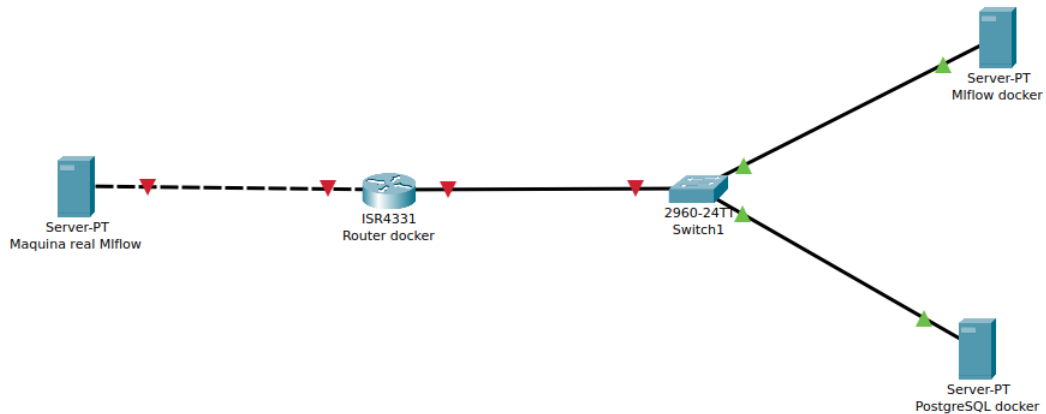


**Figura 4.1.:** Arquitectura externa

## 4.6 Esquema final de la arquitectura

---

En la imagen anterior se puede observar un esquema de red simplificado de los sistemas interconectados para el despliegue y uso de Mlflow. Se cuenta con la máquina en la que se despliega Mlflow y Postgres, el NAS con el repositorio de artefactos y los clientes que se pueden conectar a Mlflow para realizar los desarrollos que ML.



**Figura 4.2.:** Arquitectura interna

En la anterior figura se puede apreciar la red interna Docker desplegada. Estando los servidores de Mlflow y Postgres.





## 5. Definición de la metodología

Para un adecuado uso de la metodología MLOps es necesario el seguimiento de una metodología que siga los principios de MLOps. Teniendo en cuenta que la plataforma seleccionada soporta MLOps nivel 0, es necesario añadir por distintos métodos los componentes necesarios para lograr otros niveles. El nivel adaptarse a las necesidades de cada proyecto de manera independiente a otros proyectos.

La metodología a seguir en un desarrollo ML según MLOps se compone de las siguientes fases:

1. Descripción de problema.
2. Análisis de los datos.
3. Limpieza de los datos.
4. Realización y almacenamiento de experimentos.
5. Selección del modelo.
6. Puesta en producción del modelo.
7. Monitorización del rendimiento del modelo.
8. Reentrenamiento o sustitución del modelo.

Mlflow provee un repositorio para modelos, métricas, parámetros y artefactos. Para la automatización de las tareas de procesamiento y tratamiento de datos se cuenta con herramientas ETLs (*Extract, Transform, Load*). Además, las que se encuentren fuera de *frameworks* pueden ser configuradas para ser ejecutadas repetidamente de manera manual.

### 5.1. Descripción del problema

En esta fase se debe describir el problema, para esto hay que describir los objetivos que se quieren lograr, cuáles son los datos disponibles y si es factible lograr los objetivos con ellos. Es necesario realizar un estudio sobre los datos que estarán disponibles durante el proceso de entrenamiento y sus fuentes. Saber si las fuentes van a ser internas o externas y las conocer las técnicas necesarias para su extracción. Se han de definir los umbrales de error aceptables para el modelo. Por ejemplo, los modelos usados para realizar predicciones médicas siempre han de mantener una calidad muy alta por su carácter crítico.

Es necesario realizar un análisis del tiempo que se espera que el modelo este activo, como puede ser afectado por el *data drifting*, si se tiene conocimiento previo sobre variaciones temporales en la distribución de los datos. Cual va a ser su uso, el cual puede tener implicaciones sobre determinados aspectos del modelo; cuales van a ser sus beneficiarios, que implicaciones tanto positivas como negativas van a obtener de este.

Como se va a ser explotado, a través de un servicio, una librería, localmente; quien tendrá acceso. Ha de ser ejecutado de manera automática y repetitiva. Quien va a ser el responsable de su mantenimiento.

## 5.2. Análisis de los datos

Se deben entender todas las variables predictoras que se vayan a usar, junto con su correlación entre ellas y con la variable objetivo (en caso de ser un problema de aprendizaje supervisado), además de saber que rango de valores puede tomar en condiciones normales.

Para realizar esto se aplicará un análisis exploratorio de los datos (EDA). Que se trata de la combinación de técnicas estadísticas y de visualización que permitirán la extracción de información de la relación entre los datos. Además, el análisis estadístico es importante para entender la distribución de los datos, su calidad, detectar valores ausentes y poder determinar qué datos son atípicos dentro de un *dataset*.

Una de las principales herramientas que se usa es la matriz de correlación. Existen multitud de métodos para medir la correlación entre variables y dependiendo del tipo de variables se utilizarán unos u otros. En esta matriz de tamaño  $N \times N$ , siendo  $N$  el número de variables, se puede observar el coeficiente de correlación de todas las variables con todas las variables. El coeficiente de correlación es un número que puede tomar valores entre -1 y 1. El valor absoluto es la cantidad de correlación entre dos variables, siendo 0 el mínimo y uno el máximo. El signo indica si la relación es directa o inversa. Con esta técnica se puede apreciar de una manera sencilla que variables son las más adecuadas para realizar predicción. También es común incluir la una gráfica entre las variables correladas para poder apreciar visualmente la correlación entre ellas.

## 5.3. Limpieza de los datos

Para un correcto entrenamiento de los modelos es necesario que los datos cumplan con los estándares de calidad necesarios, es decir, sean precisos, no contengan datos ni repetidos, ni falsos o incorrectos, ni falten datos. En este punto se puede aplicar de manera automática el uso de herramientas ETLs, que permiten tratar datos de manera automatizada en el caso de ser necesario para elevar el nivel MLOps del sistema.

Es necesario eliminar los datos replicados por que pueden modificar el peso de

las métricas que usan los distintos algoritmos para realizar las predicciones. Como por ejemplo el KNN, en el cual podría decantar la decisión por el contrario dos veces el mismo punto.

Es necesario eliminar o corregir los datos que sean incorrectos para evitar que se aprenda de estos. Para ello es necesario haber realizado un correcto análisis previo de los datos. Hay dos posibilidades, recuperar los datos o descartar su uso. Un error que puede darse de manera común es que los datos estén en distintas unidades o escalas, en este caso es posible recuperar datos. Esto también ha de ser realizado para los datos de los que falten variables. Se ha de eliminar toda variable que sea única, o identifique de manera única a un dato porque no proveen de información para clasificar que sea extrapolable, para esto también se utilizan métodos que analizan la varianza de las variables y descartan las variables con una varianza muy baja.

Para realizar unos experimentos adecuados se ha de validar los datos que vayan a ser usados y comprobar que todo sea correcto antes de la realización de cualquier experimento.

En esta fase también se puede preparar *datasets* sobre los que se vaya a realizar selección de variables, mediante distintos métodos como: análisis univariante o multivariante y almacenarlos para su uso en siguientes pasos.

### 5.4. Realización de experimentos

La creación de experimentos y sus distintas ejecuciones (runs) es lo que permite entrenar distintos modelos realizando modificaciones sobre los elementos que consideremos necesarios para obtener los mejores resultados. En este proceso podemos guardar, asociado al propio experimento, todo tipo de datos, como las métricas de evaluación, los parámetros de entrenamiento, los artefactos, o los modelos generados.

Recapitulando, para el uso de la herramienta Mlflow es necesario la instalación del paquete de Python, “mlflow”, que contiene el cliente y el servidor de la herramienta. Debido a la gran cantidad de librerías que son necesarias para la instalación se recomienda el uso de una herramienta entornos virtuales para evitar problemas por incompatibilidad en las librerías.

Como se explicó en la selección de la herramienta, los puntos fuertes de la herramienta son su flexibilidad y su facilidad de uso. Para la creación de un experimento y de sus distintas runs solo es necesario la modificación de los *scripts* o *notebooks* que se usaban con anterioridad. Por lo general solo es necesario importar la librería de Mlflow, teniendo la opción de activar el Autolog para registrar automáticamente la información necesaria, en caso que se quiera guardar más información se puede guardar explícitamente los parámetros, métricas y artefactos que no se hayan gestionado por el modelo.

Una forma de realizar el entrenamiento de los modelos es el uso de herramientas de AutoML. AutoML es una técnica para generar modelos de manera automática,

normalmente para obtener una aceleración al inicio de un proyecto de ML. La cual proporciona un conjunto de modelos que dan buenos resultados y que a través de un tuneo de los modelos obtenidos se pueden conseguir modelos muy precisos. Aunque Mlflow no provee de este servicio, H2O sí que lo hace, esta vez de manera gratuita. H2O AutoML genera tantos modelos como sean requeridos durante el tiempo asignado, ordenándolos por las distintas métricas que se hayan indicado con el objetivo que se pueda obtener el que se considere más adecuado en función de las métricas seleccionadas. Una vez terminada la ejecución se puede almacenar en Mlflow para su comparación con otros modelos durante el proceso de selección.

Como se ha mencionado, posteriormente al primer entrenamiento de modelos, se puede ajustar de manera más precisa los parámetros de entrenamiento para los modelos obtenidos de la ejecución de la herramienta AutoML. Los modelos generados por este tipo de herramientas generan modelos de manera muy rápida y con buenos resultados, no obstante, si se requiere un mejor resultado es necesario un proceso de tuneado con la intención de optimizarlos. Cada uno de estos modelos puede ser almacenado para poder realizar comparaciones más adelante.

Además del uso de AutoML para la aceleración del desarrollo, también se puede usar otras clases de modelo no obtenidas de manera automática. Estos modelos han de ser elegidos por el científico de datos entendiendo cuales son las características de problema en cuestión, conocimiento adquirido durante las dos primeras fases de proyecto. Para esto probará distintos algoritmos con distintas configuraciones y distintos conjuntos de variables, con el objetivo encontrar el mejor modelo posible. Se pueden usar técnicas como la selección de variables o el *wrapper*, para evitar duplicar el *dataset* al almacenar un modelo, se almacenará la ruta donde se encuentra.

La opción de Autolog de Mlflow permite almacenar los parámetros utilizados por un algoritmo, con los datos de entrenamiento realiza y almacena un estudio del rendimiento y guarda el modelo y su descripción en un archivo *.yaml*. En dicho archivo *.yaml* se encuentra la *signature* del modelo que es el conjunto de variables que ha de recibir como entrada y la que producirá como salida (interfaz de entrada/salida). Todo aquel parámetro, métrica o artefacto adicional que se quiera guardar ha de ser almacenado de manera manual. Para poder tener unas métricas realistas y honestas es necesario almacenar realizar un estudio sobre la evaluación de los modelos de manera activa. La librería *mlflow-extend* permite el almacenamiento de artefactos que requieren un preprocesado para obtenerse, como puede ser una matriz de confusión. Otros parámetros que pueden no haber sido tenidos en cuenta como la semilla para la generación de números semialeatorios. También es posible guardar artefactos como el resumen del modelo.

### 5.5. Selección de mejor modelo

En esta fase se ha de seleccionar el mejor modelo entre los producidos en la anterior fase. Para ello se tomará en cuenta todas las métricas recogidas durante la fase de entrenamiento, y los artefactos que ayuden a su entendimiento, como las matrices de confusión, curvas *ROC* o los informes de clasificación.

Mlflow provee con una herramienta para la visualización de las métricas en gráficos y la capacidad de ordenar los experimentos con respecto a distintas métricas. Gracias a esta visualización se puede decidir cuál es el modelo con mejores cualidades, o si es necesario seguir produciendo modelos para obtener nuevos modelos con mejores rendimientos.

En el caso de tener que eliminar un modelo hay dos estados en los que puede encontrarse como eliminado, la primera con la herramienta web de Mlflow se establece su estado del ciclo de vida a *deleted*, lo que lo excluye de las comparaciones; en la segunda si fuera necesario se puede eliminar su directorio de artefactos y sus apariciones en la base de datos. Para ello se debe eliminará todas las referencias a la ejecución con anterioridad. La finalidad de la primera forma de eliminación es permitir una comparación más sencilla entre los modelos, la segunda es ser eliminar modelos en los que haya errores o se hayan incluido por accidente.

Una vez seleccionado el modelo se ha de registrar para que pase a la siguiente fase. Se comprueba en la pestaña de modelos que ha sido registrado, su versión y su *signature* (entradas y salidas). Se puede además puede asignar una etiqueta para su búsqueda con mayor facilidad. Y cambiar el estadio en el ciclo de vida también puede ser definido desde esta pantalla, se puede elegir entre ninguno, *staging*, *production* y *archived*. El primer estadio se usará para los modelos aceptados, pero sobre los que todavía se requiere seguir realizando pruebas. En producción se encuentran los modelos que se están usando de manera activa y que están siendo expuestos como servicios. *Archived* son los modelos que ya han pasado por producción y han sido retirados del servicio. Se pueden generar distintas versiones de un modelo registrado y que las versiones estén en distintas fases del ciclo de vida.

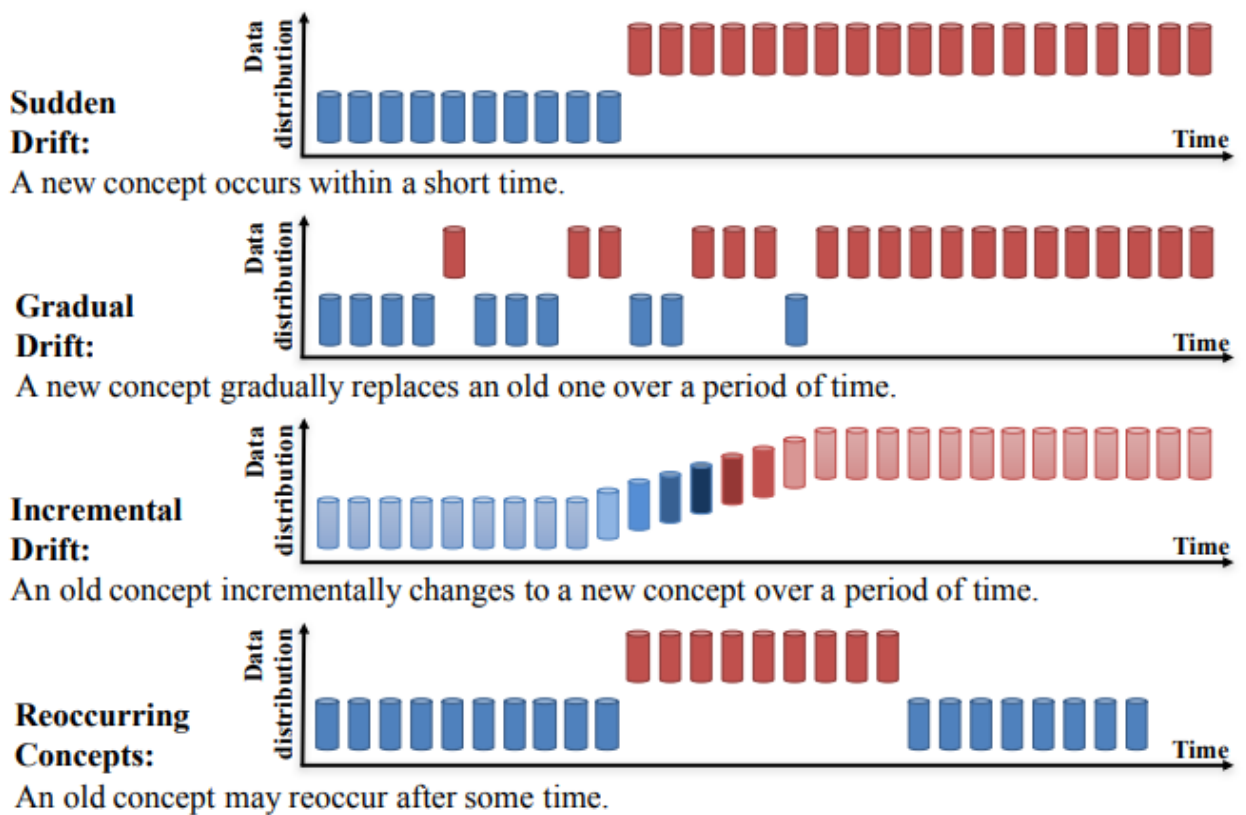
### 5.6. Puesta en producción

Para el correcto aprovechamiento del repositorio de modelos y evitar tener el mismo modelo duplicado en distintos lugares se exponen los modelos como servicios web, o se puede recoger el modelo del propio repositorio de artefactos. Además de esta manera se facilita la tarea de realizar predicciones desde cualquier plataforma, a través de los accesos con soporte oficial o de la API REST. Esto permite, además, que en el caso de que el servicio de predicción sea computacionalmente exigente y requiera de aceleración de *hardware* o software específico pueda ser más fácilmente configurable en un servidor centralizado. Su despliegue puede ser realizado en una arquitectura de microservicios.

Es necesario guardar todos los datos que están llegando a los modelos para poder estudiar la degradación que sufren los modelos con el paso del tiempo. Además, es posible que los nuevos datos también se deben usar para entrenar a los nuevos modelos, o para reentrenar el modelo que ya está en producción.

## 5.7. Monitorización del modelo

Se debe mantener los modelos bajo constante monitorización, para reconocer cuando cruza los umbrales de error definidos, tras los cuales no es aceptable su rendimiento y es necesario su reentrenamiento o sustitución. Los modelos son sensibles a los cambios de tendencias en los datos, no pueden predecir algo que no conocen. Para detectar la degradación de la capacidad de predicción de los modelos se realiza un estudio sobre los nuevos datos recibidos. Esto es debido a que la causa es que el modelo no conoce los patrones seguidos por los datos a predecir después de haber sido entrenado. No se conoce tampoco cual será la distribución futura de los datos, cuanto tardarán en cambiar ni cuando lo harán [LLD<sup>+</sup> 18].



**Figura 5.1.:** Tipos de *data drift* [LLD<sup>+</sup> 18]

En la figura 5.1. se muestra distintos tipos de deriva de los datos en distribuciones teóricas. Cada uno de los tipos de cambio requerida una respuesta distinta para poder realizar un tratamiento adecuado. También es necesario mantener antiguos modelos, porque en el caso de volver a una distribución pasada tenerlos a disposición reducirá el tiempo de respuesta, como en los casos 2 y 4.

### 5.8. Reentrenamiento o sustitución de modelo

Una vez superada la tolerancia de error definida es necesario realizar un nuevo estudio para generar un nuevo modelo o reentrenar el modelo actual. Para esto es necesario establecer una condición que se active cuando se llegue a este umbral. Una vez activada la condición se vuelve a realizar los pasos anteriores. Es posible volver a tener que realizar un estudio del problema si los datos han cambiado mucho. Siempre es necesario realizar una limpieza de los datos, al haber sido obtenidos a través de un servicio es necesario, se ha podido intentar predecir varias veces la misma información, predecir sin conocer todos los campos o sin conocer. Se ha de usar los scripts anteriormente generados para limpiar el 9 del modelo original.

Una vez preparados los nuevos datos se ha reentrenar el antiguo modelo con los nuevos datos, y volver a realizar un estudio con AutoML o con algoritmos seleccionados. Todos han de ser registrados como nuevas runs dentro del experimento original. A continuación, se compararán los resultados obtenidos mediante las métricas y artefactos para decidir cuál será el modelo que será puesto en producción como sustituto.

El modelo que se retira del servicio se quedara registrado como *archived* para poder volver a él si en algún momento la deriva volviera hacia el punto en el que fue entrenado. Además, se incluirán en las comparaciones para las selecciones de nuevos modelos, antes y después de la creación de nuevos modelos.





## 6. Resultados

El objetivo en este punto es comprobar el uso de la metodología y de la herramienta seleccionada. Para ello se realizará un proyecto de ML usando ambas, siguiendo la metodología y utilizando la mayoría de las funcionalidades disponibles. En este caso, debido a que solo se va a aplicar una sola vez sin realizar actualizaciones de los modelos, se aplicará el nivel 0 de MLOps.

### 6.1. Descripción de problema

El *dataset* seleccionado para este problema proviene de Kaggle [kag]. Kaggle es una página que pone a disposición de los usuarios retos de *machine learning* y los *datasets* asociados a dicho reto. El *dataset* escogido [Meh18], publicado bajo licencia “CC0: Public Domain” contiene información sobre los cuerpos cercanos a la Tierra (NEOs) y esta extraído de una API proporcionada por la NASA [nas].

El problema que se desea resolver es la obtención de un clasificador que sea capaz de reconocer que asteroides son peligrosos y cuáles no. Esta información puede ser usadas de varias maneras, como predictor al uso o como herramienta para detectar riesgos y determinar la prioridad con el objetivo de realizar más cálculos sobre su trayectoria (el problema de los 3 cuerpos es caótico). Independientemente del uso que se le dé es un sistema crítico y por lo tanto la tolerancia a errores es muy baja. Su uso más probable es formar parte de la toma de decisiones en las misiones DART [nas22] el programa de la NASA para desviar asteroides en ruta a colisionar con la Tierra.

En este caso, debido a que es un *dataset* cerrado y no se va a poder registrar nuevos datos, no se va a sufrir *data drifting*.

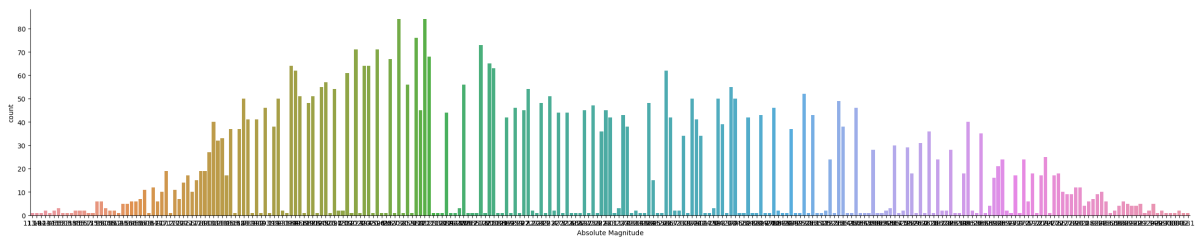
Debido a que el sistema es crítico y es igual de negativo los falsos positivos que negativos, se usarán el *f1-score* y área bajo la curva *ROC* como métricas para comprobar la eficacia de los modelos.

### 6.2. Análisis de los datos

Para realizar el análisis de los datos, lo primero es exponer los distintos campos que componen el *dataset*:

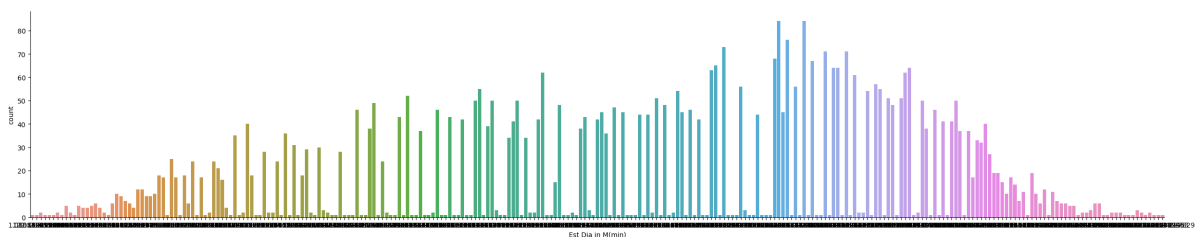
- NEO Reference ID: Identificador único de cada asteroide. Es descartado porque no provee de información relevante para clasificar los asteroides.

- **Name:** Nombre del asteroide. Será descartado porque no provee de información relevante para clasificar los asteroides. Los identificadores o nombres únicos no aportan información generalizable para el entrenamiento los modelos.
- **Absolute Magnitude:** Describe el brillo de un objeto a 32,616 años luz (10 Pársecs), en un espacio vacío. Tiene relación con el tamaño y luminosidad de los cuerpos celestes. En la figura 6.1. vemos la distribución de la variable en la muestra.



**Figura 6.1.:** Magnitud absoluta

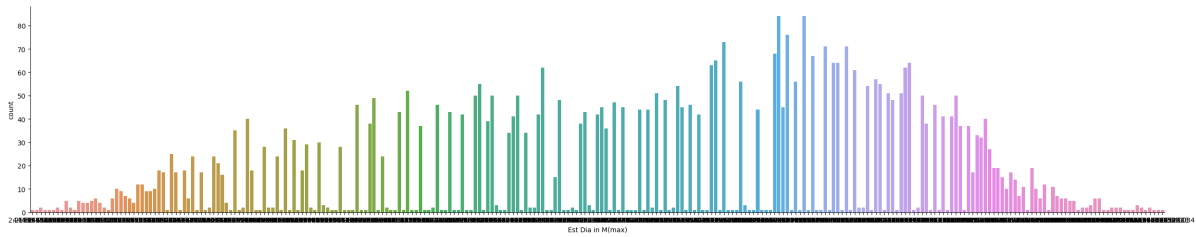
- **Est Dia in <unidad>(min):** Estimación mínima del diámetro del cuerpo estelar. Se encuentra en las siguientes unidades: kilómetros, metros, millas y pies. Se selecciona metros por dar más precisión que los kilómetros y ser una unidad del Sistema Métrico Internacional. En la figura 6.2 vemos la distribución de la variable en la muestra.



**Figura 6.2.:** Estimacion diámetro mínimo

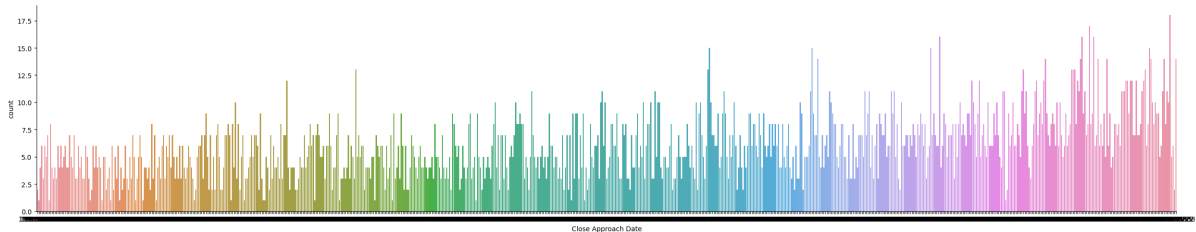
- **Est Dia in <unidad>(max):** Estimación máxima del diámetro del cuerpo estelar. Se encuentra en las siguientes unidades: kilómetros, metros, millas y pies. Se selecciona metros por dar más precisión que los kilómetros y ser una unidad del Sistema Métrico Internacional. La relación entre la los diámetros es muy alto, además de con la magnitud absoluta, pues todas describen el tamaño del asteroide. En la figura 6.3. vemos la distribución de la variable en la muestra.

## 6.2 Análisis de los datos



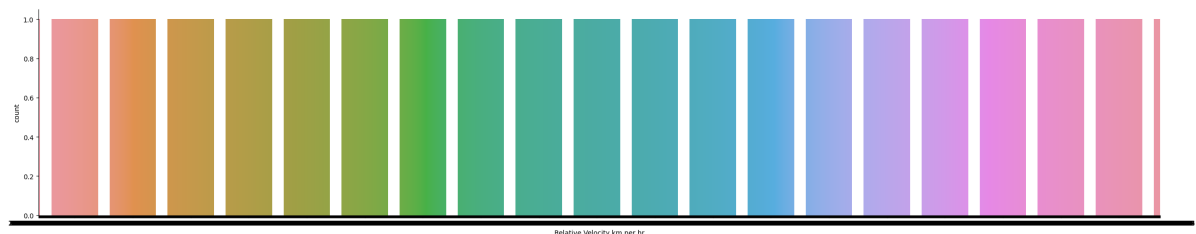
**Figura 6.3.:** Estimacion diámetro máximo

- Close approach date: Día en que el asteroide relocalizó su última pasada mas cercana. Con el fin de simplificar las variables a utilizar y facilitar su análisis se utilizará la diferencia de tiempo desde que ocurrió el evento. Siendo la última pasada registrada en julio de 2016, se utilizará como fecha de referencia para el cálculo de diferencias enero de 2017. En la figura 6.4 vemos la distribución de la variable en la muestra.



**Figura 6.4.:** Dia de acercamiento

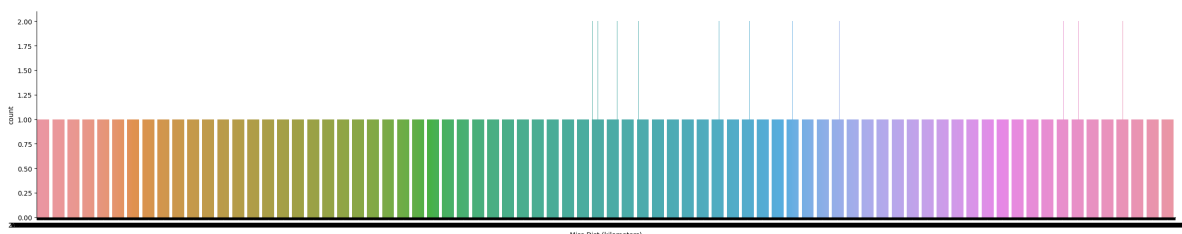
- Epoch date close approach: Muestra la misma información que close approach date, pero en tiempo astronómico. Ya que son dos variables mostrando la misma información se descartará una, y se usará close approach date.
- Relative velocity in <unit>: Velocidad relativa entre el NEO y la Tierra, ofrecida en km/h, km/s, mph (millas por hora). Se escoge la unidad de medida en km/h por los mismos motivos anteriores. En la figura 6.5 vemos la distribución de la variable en la muestra. La velocidad sigue una distribución completamente uniforme, sin velocidades repetidas.



**Figura 6.5.:** Velocidad relativa

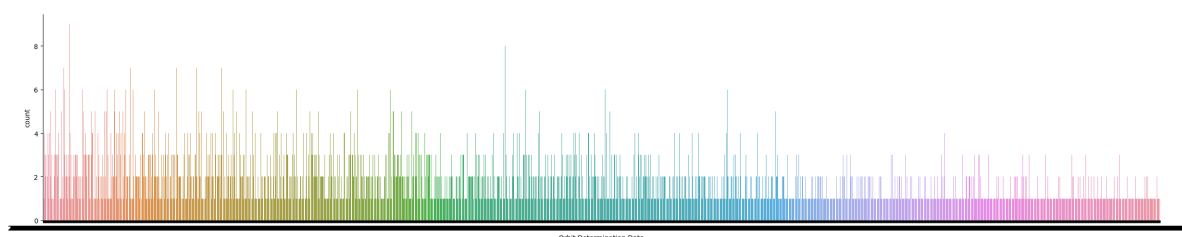
- Miss Dist. (<unidad>): Distancia a la que ocurrió la pasada se encuentra

en UA (Unidades Astronómicas o la distancia entre la tierra y el sol), lunar, km y millas. Debido a que es recomendable tener las distintas variables en las mismas unidades se selecciona km. Tiene la misma distribución que la velocidad relativa aunque si que hay valores repetidos en este caso. En la figura 6.6 vemos la distribución de la variable en la muestra.



**Figura 6.6.:** Distancia de fallo

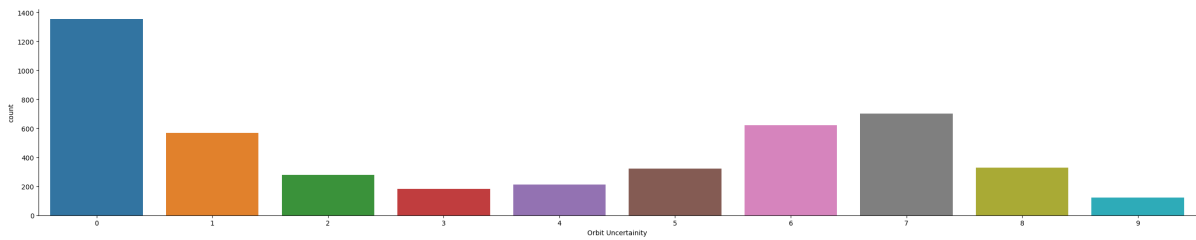
- Orbiting body: Cuerpo orbitado, al ser todo NEOs siempre es la tierra por lo que un cambio con varianza igual a cero no genera información relevante.
- Orbit ID: Identificador de la órbita. Se ha de descartar por no aportar información relevante para el clasificador.
- Orbit determination date: Día en que la órbita fue determinada. Con el fin de simplificar las variables a utilizar y facilitar su análisis se utilizará la diferencia de tiempo desde que ocurrió el evento con el final del año de la última instancia recogida en el *dataset*. La última órbita determinada es de 2017 por lo que se usará 2018 como fecha para calcular la diferencia. Se puede ver un descenso de las determinaciones de posiciones en el tiempo. En la figura 6.7 vemos la distribución de la variable en la muestra.



**Figura 6.7.:** Día de determinación de la órbita

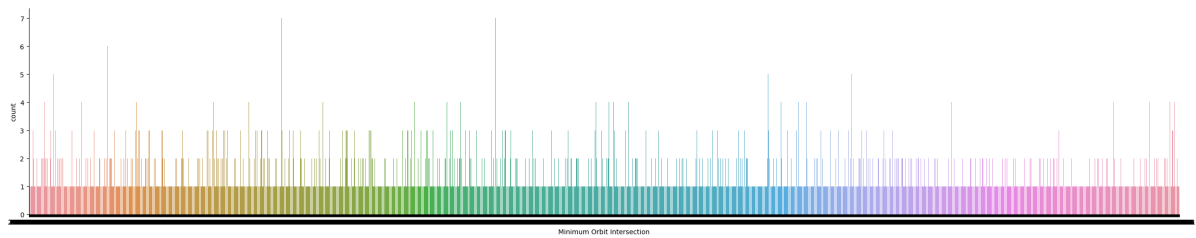
- Orbit uncertainty: Falta de certeza sobre la órbita calculada de 0 a 9. Es una variable discreta ordinal, la distribución indica que o se conoce con muy alta certeza las órbitas que van a seguir los cuerpos o se tiene una incertidumbre moderada/alta. En la figura 6.8 vemos la distribución de la variable en la muestra.

## 6.2 Análisis de los datos



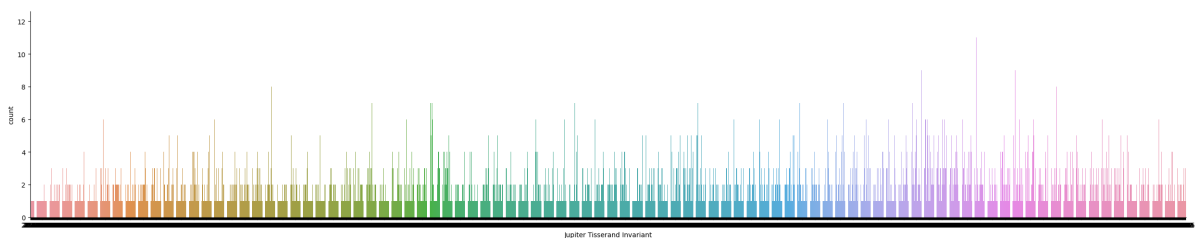
**Figura 6.8.:** Incertidumbre sobre la orbita

- **Minimun orbit intersection:** Distancia mínima entre los dos puntos en los que intersecan la órbita terrestre con la del asteroide. Un valor muy bajo quiere decir que comparte gran parte de su trayectoria con La Tierra. Se puede observar que hay asteroides con trayectorias parecidas que asteroides con más tiempo entre acercamientos. En la figura 6.9 vemos la distribución de la variable en la muestra.



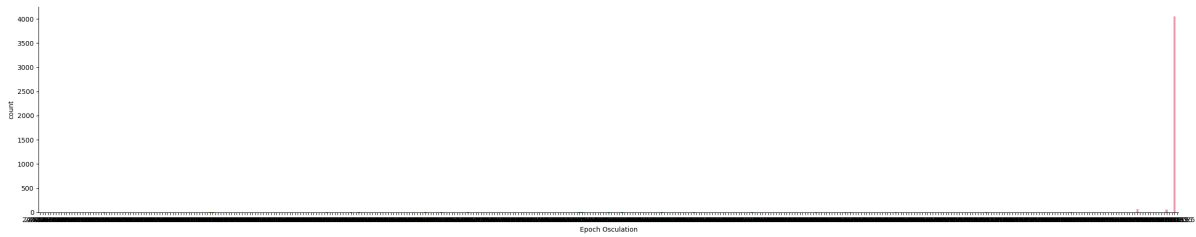
**Figura 6.9.:** Intersección mínima de la orbita

- **Jupiter Tisserand Invariant:** Perturbación ejercida en el asteroide por Júpiter. En la figura 6.10 vemos la distribución de la variable en la muestra.



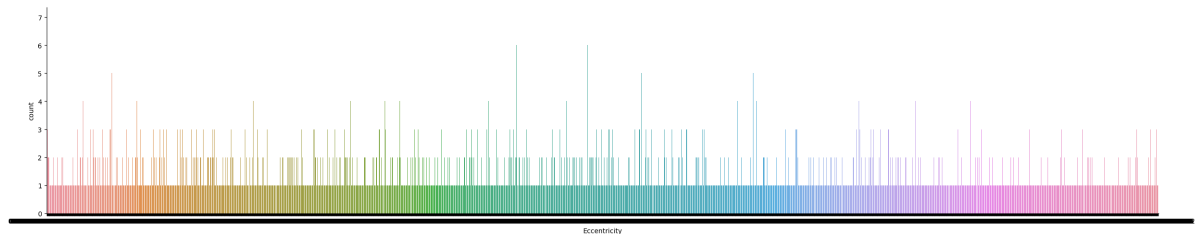
**Figura 6.10.:** Tisserand de Jupiter

- **Epoch Osculation:** Orbita que hubiera seguido el cuerpo celeste sin tener en cuenta la perturbación que ejercen terceras masas. Casi todos los cuerpos toman valores cercanos a los 6 millones, existe poca variación de la información. En la figura 6.11 vemos la distribución de la variable en la muestra.



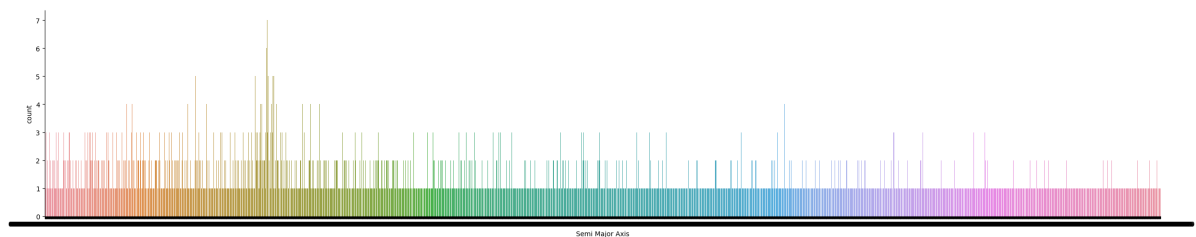
**Figura 6.11.: Osuclación**

- **Eccentricity:** Relación entre los semiejes mayor y menor de la elipse que describe la órbita. La excentricidad toma valores entre 0 y 1 siendo 0 un círculo y 1 una parábola, es decir, indica como de achatada es la elipse. En la figura 6.12 vemos la distribución de la variable en la muestra.



**Figura 6.12.: Excentricidad**

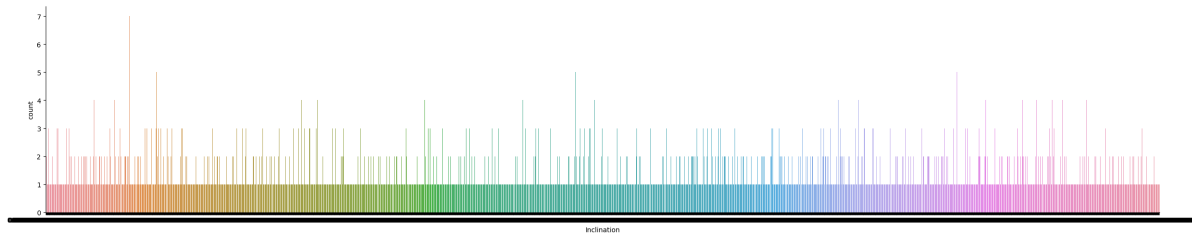
- **Semi Major Axis:** Distancia entre el centro de la elipse y su perímetro pasando por uno de los focos. El semeje mayor toma valores en torno a 1. En la figura 6.13 vemos la distribución de la variable en la muestra.



**Figura 6.13.: Semieje mayor**

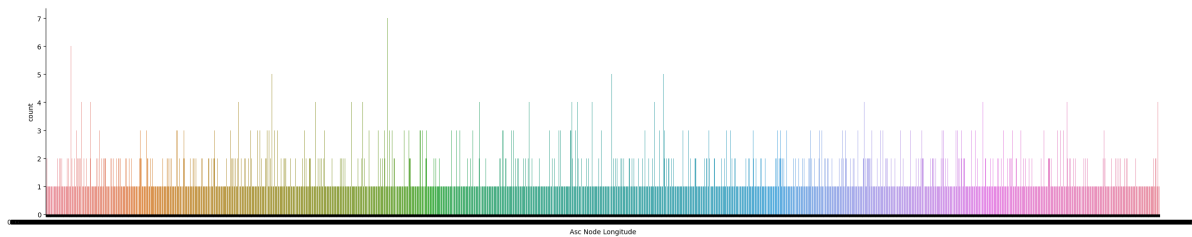
- **Inclination:** Ángulo formado entre los planos en los que orbitan la Tierra y el asteroide. Los valores se agrupan sobre todo en ángulos bajos y pocos en ángulos casi perpendiculares. En la figura 6.14 vemos la distribución de la variable en la muestra.

## 6.2 Análisis de los datos



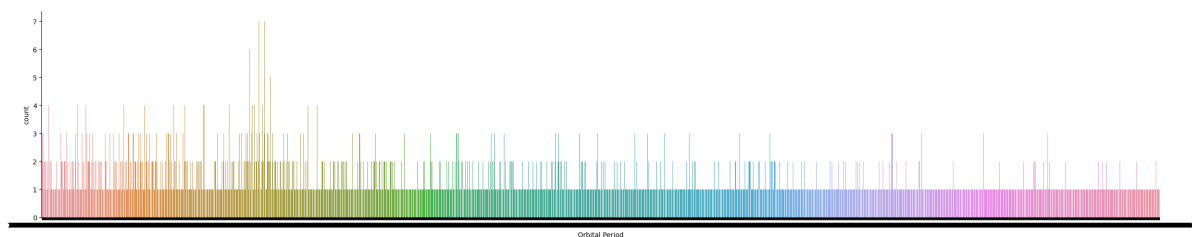
**Figura 6.14.:** Inclinación

- Asc Node longitude: El ángulo, medido en sentido antihorario, formado entre la dirección de referencia y la recta que interseca entre los planos de las orbitas. Todas las orbitas se distribuyen uniformemente en todos los ángulos posibles. En la figura 6.15 vemos la distribución de la variable en la muestra.



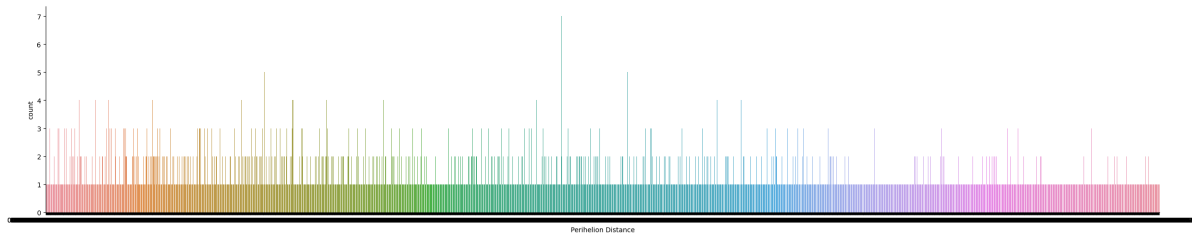
**Figura 6.15.:** Logitud del nodo ascendente

- Orbital period: Tiempo que tarda el cuerpo en completar una vuelta alrededor del cuerpo al que orbita. Debido a su cercanía a La Tierra sus periodos son cercanos al de La Tierra, 365 días. En la figura 6.16 vemos la distribución de la variable en la muestra.



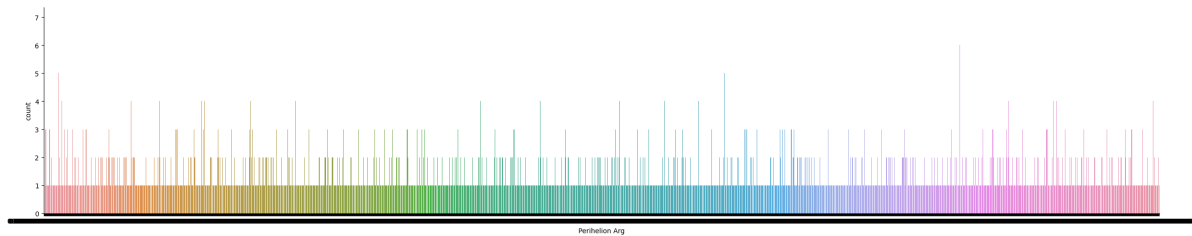
**Figura 6.16.:** Periodo orbital

- Perihelion distance: Distancia a la que se encuentran dos cuerpos orbitando en punto más cercano en la órbita. En la figura 6.17 vemos la distribución de la variable en la muestra.



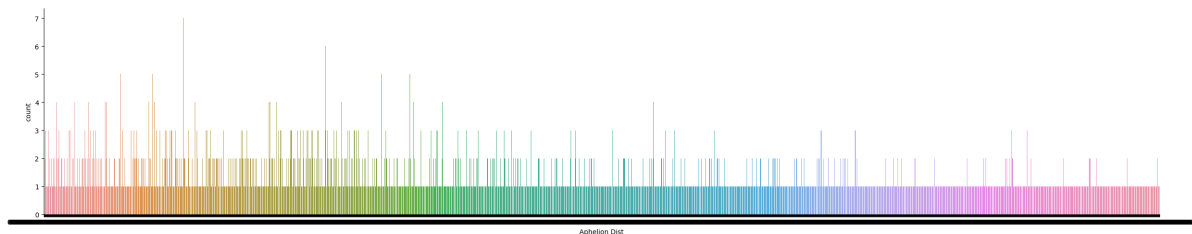
**Figura 6.17.:** Distancia en el perihelio

- Perihelion arg: Ángulo formado entre la intersección de los planos y la intersección entre plano del asteroide el plano perpendicular al terrestre pasando por la dirección de referencia. En la figura 6.18 vemos la distribución de la variable en la muestra.



**Figura 6.18.:** Argumento del perihelio

- Aphelion distance: Distancia a la que se encuentran dos cuerpos orbitando en punto más lejano en la órbita. Tiene una distribución parecida al perihelio, pero desplazada a la izquierda por ser datos más pequeños que el perihelio. En la figura 6.19 vemos la distribución de la variable en la muestra.

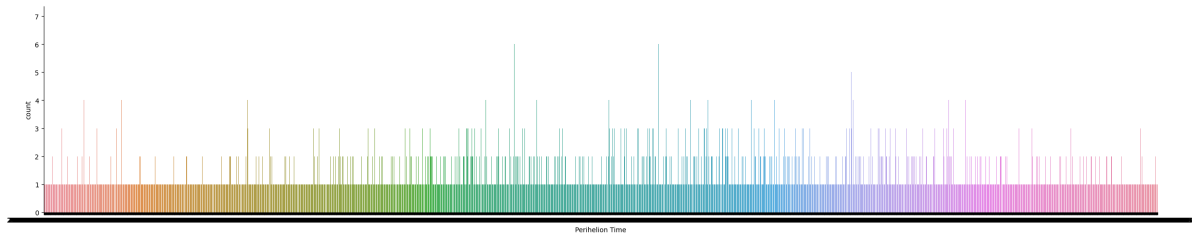


**Figura 6.19.:** Distancia en el afelio

- Perihelion time: Momento en que se alcanza el perihelio. En la figura 6.20 vemos la distribución de la variable en la muestra.

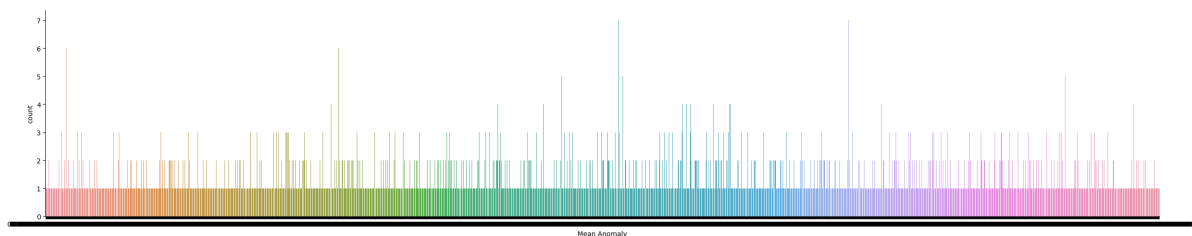


## 6.2 Análisis de los datos



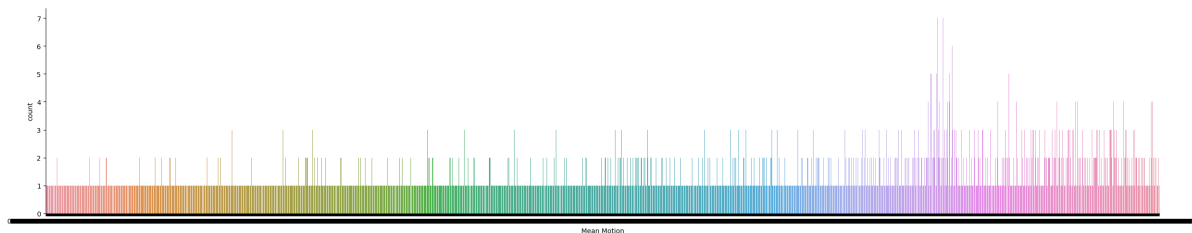
**Figura 6.20.:** Tiempo en el perihelio

- Mean anomaly: Ángulo recorrido por el asteroide con respecto a su perihelio si su órbita fuera totalmente circular. Se encuentra entre 0 y 360 grados. En la figura 6.21 vemos la distribución de la variable en la muestra.



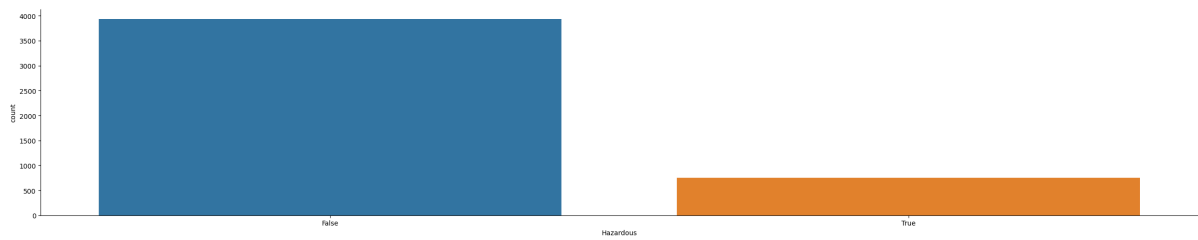
**Figura 6.21.:** Anomalía media

- Mean motion: Velocidad angular media del asteroide. Los asteroides tienden a ser cuerpos rápidos. En la figura 6.22 vemos la distribución de la variable en la muestra.

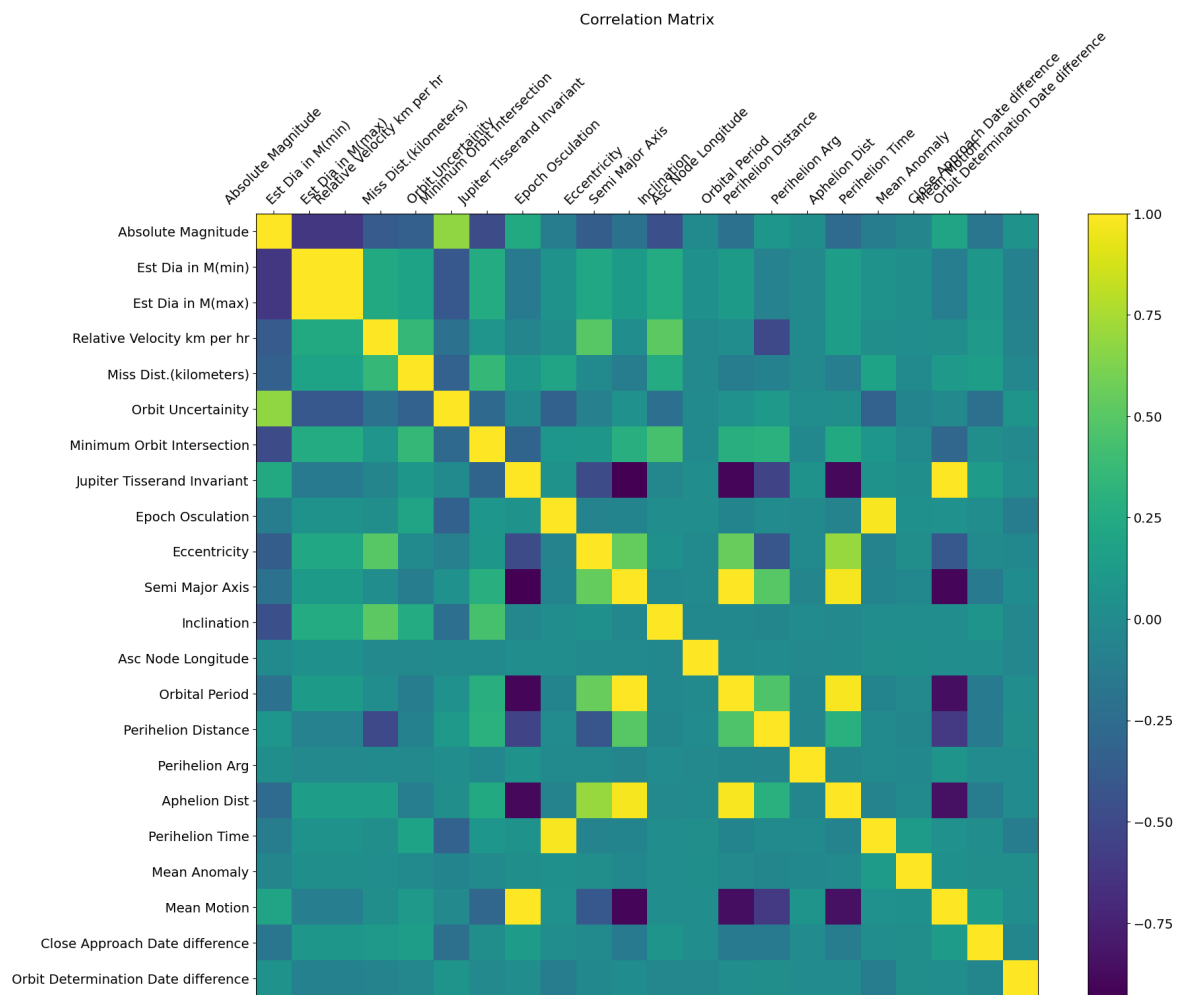


**Figura 6.22.:** Velocidad angular media

- Equinox: Solo contiene un único valor por lo que no es útil para la clasificación.
- Hazardous: Variable a clasificar, peligrosidad del asteroide, se presenta como una variable booleana. Se puede apreciar la gran diferencia que hay entre ambos casos a clasificar y posiblemente sea necesario equilibrar el *dataset*. En la figura 6.23 vemos la distribución de la variable en la muestra.



**Figura 6.23.:** Peligrosidad



**Figura 6.24.:** Matriz de correlación

En la anterior matriz de correlación, figura 6.24 se aprecia la relación directa e indirecta entre distintas variables. La relación entre la los diámetros es 1, por lo que están totalmente correlados, además de con la magnitud absoluta, pues todas describen el tamaño del asteroide. La velocidad relativa es directamente proporcional a la energía cinética del asteroide ( $E = \frac{1}{2}mv^2$ ). Tienen una correlación significativa de manera directa, lo incertidumbre de la orbita y la magnitud

absoluta, la velocidad relativa con excentricidad e inclinación, el movimiento medio y la Tisserand de Júpiter, el tiempo en el perihelio y la osculación, la excentricidad con los semiejes mayor y menor ( $\sqrt{1 - \frac{b^2}{a^2}}$ , siendo  $a$  el semieje mayor y  $b$  el menor), con el periodo orbital y con la distancia en el afelio, el semieje mayor con el periodo orbital y las distancias en el perihelio y afelio y el periodo orbital y la distancia en el afelio. En cuanto a las correlaciones inversas relevantes, la distancia en el afelio y la velocidad relativa, la Tisserand de Júpiter con las distancias en el afelio y perihelio y el periodo orbital, el movimiento medio con el semieje mayor, inclinación y el periodo orbital. La gran mayoría de estas dependencias vienen dadas por alguna forma matemática.

### 6.3. Limpieza de los datos

El *dataset* usado ya tenía un proceso de limpieza realizado, por lo que se puede trabajar directamente con él, aparte de la eliminación de los datos que no sean de utilidad para extraer información sobre la variable a predecir. Además, en esta fase también se transformará las variables que contienen fechas, para estas variables utilizaremos la diferencia con las fechas determinadas durante la anterior fase. En el caso de ser necesario se ha preparado *datasets* sobre los que se ha aplicado selección de variables univariante y multivariante.

Para la obtención de las métricas se ha dividido el *dataset* en un 90% para entrenamiento y un 10% para ejecución.

### 6.4. Realización de experimentos

Para facilitar la exposición de los experimentos realizados se ha decidido dividir esta sección por cada uno de los métodos y librerías utilizados.

#### 6.4.1. AutoML (h2o)

Para comenzar se iniciará con una ejecución con AutoML con el objetivo de conseguir un modelo del cual se pueda partir. AutoML calculará los  $N$  mejores modelos en el tiempo proporcionado, en este caso se le pide los 10 mejores. En alrededor de un 1 minuto se generan modelos y todos son almacenados para poder realizar pruebas en próximas fases con ellos.

AutoML progress:  (done) 100%

	model_id	auc	logloss	aucpr	mean_per_class_error	rmse	mse
	GBM_1_AutoML_1_20230513_234506	0.999912	0.00928563	0.999561	0.00761965	0.0504775	0.00254798
	XGBoost_2_AutoML_1_20230513_234506	0.999735	0.0145927	0.998333	0.00539864	0.0550575	0.00303133
	StackedEnsemble_BestOfFamily_1_AutoML_1_20230513_234506	0.999704	0.0106091	0.998978	0.00587443	0.0518207	0.00268538
	GBM_3_AutoML_1_20230513_234506	0.999678	0.0111461	0.998843	0.00635022	0.0525779	0.00276443
	GBM_4_AutoML_1_20230513_234506	0.99967	0.0110764	0.998795	0.00444606	0.0537324	0.00288717
	XGBoost_3_AutoML_1_20230513_234506	0.999663	0.0118541	0.998101	0.00460499	0.0532341	0.00283387
	GBM_2_AutoML_1_20230513_234506	0.999657	0.0103876	0.998806	0.0041282	0.0504423	0.00254443
	StackedEnsemble_AllModels_1_AutoML_1_20230513_234506	0.999601	0.0106465	0.99865	0.00650915	0.0513825	0.00264016
	XRT_1_AutoML_1_20230513_234506	0.99955	0.0381072	0.998457	0.00460499	0.0770762	0.00594075
	XGBoost_1_AutoML_1_20230513_234506	0.99938	0.0193486	0.998055	0.00857224	0.0577314	0.00333291
	DRF_1_AutoML_1_20230513_234506	0.998928	0.0388534	0.998228	0.00508179	0.072053	0.00519163
	GLM_1_AutoML_1_20230513_234506	0.98838	0.103139	0.953084	0.0746234	0.177203	0.0314009

Figura 6.25.: Entrenamiento AutoML

En la figura 6.25 se muestra los modelos producidos por AutoML, junto con las métricas correspondientes.

A continuación, se almacena todos los modelos generados por esta técnica en Mlflow, almacenado como parámetros el número de modelos generado, el resto de los parámetros para la generación del modelo son almacenados en un artefacto por facilidad. AutoML genera un informe en texto en el que expone cuales han sido los parámetros usados para su entrenamiento. Además, se almacena las métricas que se han tomado: *f1* y *auc*, por otro lado, también se ha almacenado su matriz de confusión como un artefacto.

Hazardous\_NEOS

Experiment ID: 1    Artifact Location: file://home/mlflow/artifacts/1

Description

Table view   Chart view   metrics.mse < 1 and params.model = "tree"

Sort: F1   Columns

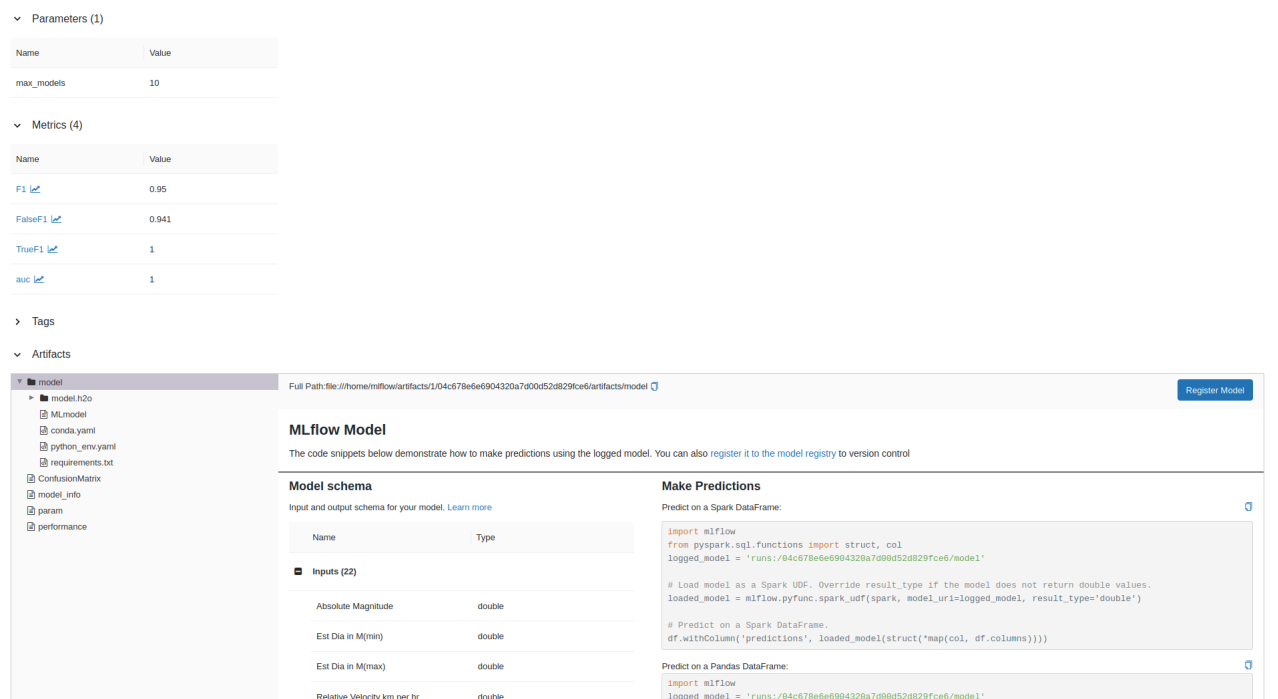
Time created: All time   State: Active

	Run Name	Created	Duration	Source	Models	Metrics			
						F1	FalseF1	TrueF1	auc
	fun-flea-241	7 minutes ago	3.4s	ipykern...	h2o	0.95	0.941	1	1
	amazing-shoat-502	7 minutes ago	3.0s	ipykern...	h2o	0.944	0.933	1	1
	casual-newt-395	7 minutes ago	3.0s	ipykern...	h2o	0.942	0.93	1	1
	likeable-mole-988	7 minutes ago	3.1s	ipykern...	h2o	0.93	0.917	1	1
	auspicious-bat-513	7 minutes ago	3.2s	ipykern...	h2o	0.885	0.863	1	1
	adaptable-sloth-603	7 minutes ago	3.2s	ipykern...	h2o	0.851	0.822	1	1
	popular-ape-804	7 minutes ago	3.0s	ipykern...	h2o	0.731	0.681	0.992	1
	bald-sheep-881	7 minutes ago	2.9s	ipykern...	h2o	0.625	0.556	0.986	0.999
	bustling-lamb-124	7 minutes ago	3.0s	ipykern...	h2o	0.561	0.476	0.998	1
	righteous-lark-765	7 minutes ago	3.0s	ipykern...	h2o	0.506	0.412	0.994	1
	luminous-sheep-699	7 minutes ago	3.0s	ipykern...	h2o	0.469	0.368	0.991	0.999

Figura 6.26.: Modelos generados

En la figura 6.26 se puede ver todos los modelos almacenados, junto con las métricas usadas para ordenarlas.

## 6.4 Realización de experimentos



**Figura 6.27.:** Modelo H2O

En la figura 6.27 se observa el modelo resultante observado con sus parámetros, métricas y artefactos.

### 6.4.2. Random forest (Sklearn)

Como la mayoría de las runs anteriores han producido los mejores resultados son sobre todo con modelos basados en *random forest*, se va a generar otro conjunto de runs con este mismo método partiendo de la librería de sklearn. Esta librería dispone de Autolog, una facilidad que nos ofrece Mlflow y por la que solo es necesario almacenar las métricas y parámetros que no se disponga de manera automática, además de generar la *signature* del modelo. También se guardará algunos artefactos como las matrices de confusión. Se ha decidido entrenar arboles de máxima profundidad y 10, 100, 1.000 y 10.000 árboles.

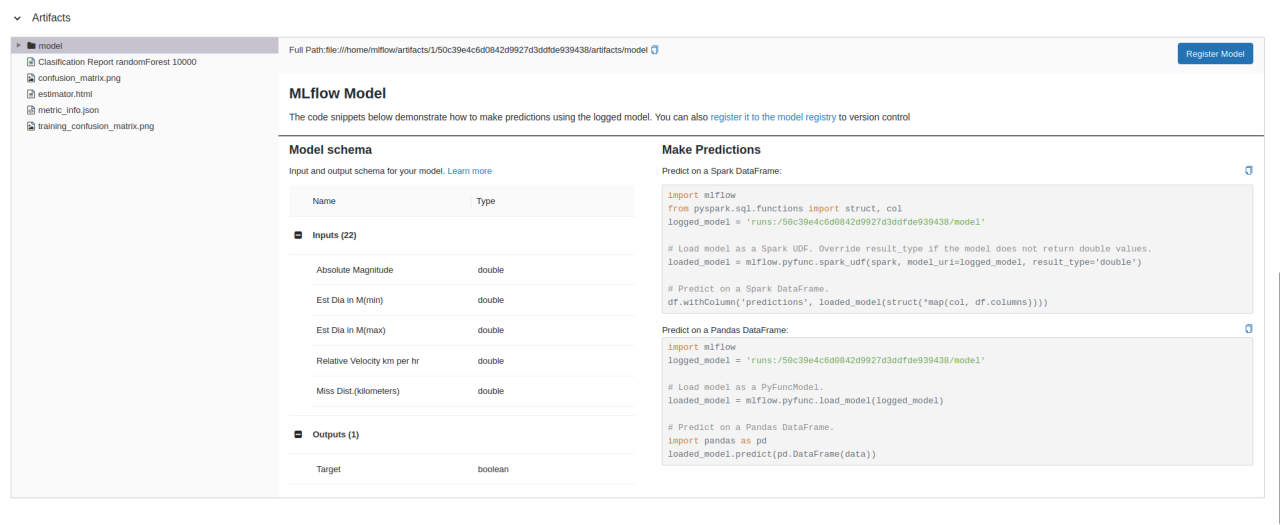


Figura 6.28.: Artefactos

En la figura 6.28 se puede observar los parámetros de configuración, las métricas obtenidas y todos los artefactos, incluyendo el modelo. En el modelo se presenta su *signature*, es decir sus variables de entrada y salida. Se han guardado además como parámetros la semilla que se ha usado y el tamaño del conjunto de entrenamiento.

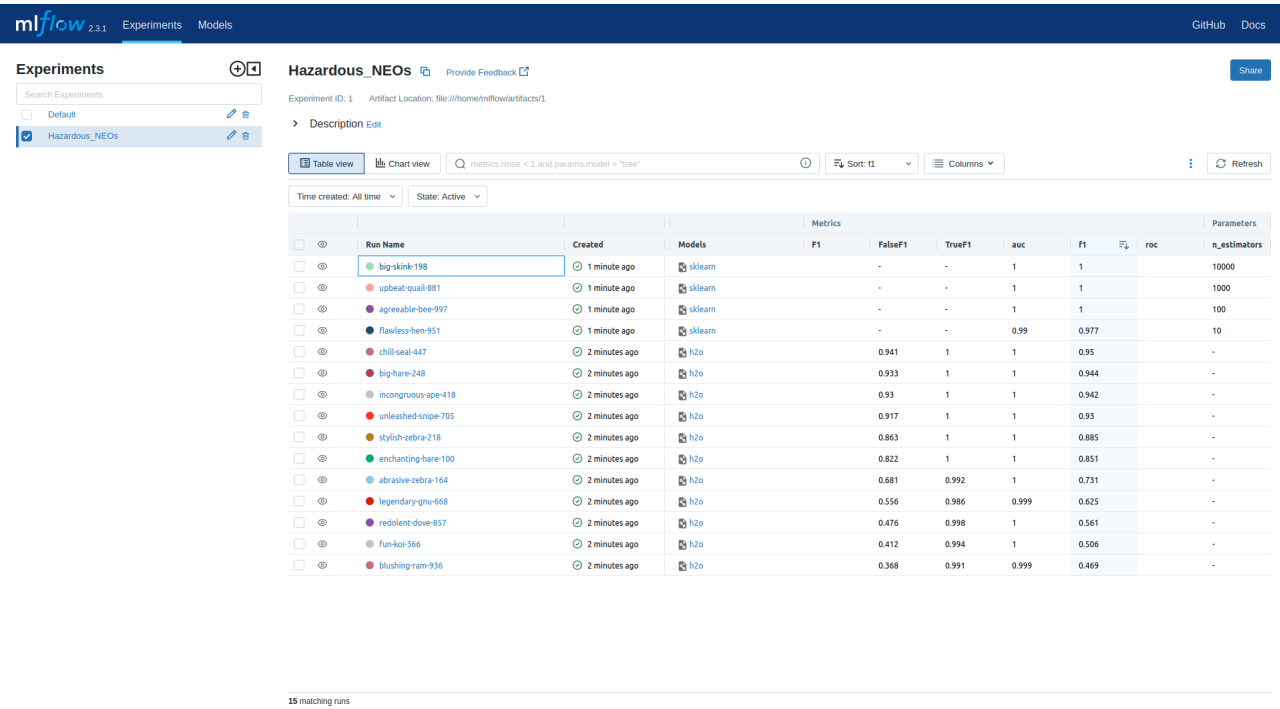


Figura 6.29.: Nuevos modelos

En esta figura 6.29 se muestra los modelos actuales registrados en el sistema,

## 6.4 Realización de experimentos

---

en esta se puede apreciar como hay modelos provenientes de distintas librerías. Además de los valores de las métricas usadas para comparar los modelos.

▼ Parameters (21)

Name	Value
bootstrap	True
ccp_alpha	0.0
class_weight	None
criterion	gini
maxDepth	None
max_depth	None
max_features	sqrt
max_leaf_nodes	None
max_samples	None
min_impurity_decrease	0.0
min_samples_leaf	1
min_samples_split	2
min_weight_fraction_leaf	0.0
n_estimators	10000
n_jobs	-1
oob_score	False
randomState	1606323526
random_state	None
testSize	0.1
verbose	0
warm_start	False

**Figura 6.30.:** Parámetros

En la figura 6.30 se aprecia los parámetros usados para entrenar el modelo, muchos capturados gracias a Autolog.

## ▼ Metrics (13)

Name	Value
RandomForestClassifier_score_xTest 	1
auc 	1
f1 	1
f1_score_xTest 	1
precision 	1
roc_auc_score_xTest 	1
training_accuracy_score 	1
training_f1_score 	1
training_log_loss 	0.008
training_precision_score 	1
training_recall_score 	1
training_roc_auc 	1
training_score 	1

**Figura 6.31.:** Métricas

En la figura 6.31 se observa las métricas recogidas por el Autolog son sobre el conjunto de entrenamiento, por lo que es necesario calcular y almacenar de manera manual las métricas sobre el conjunto de test.



## 6.4 Realización de experimentos

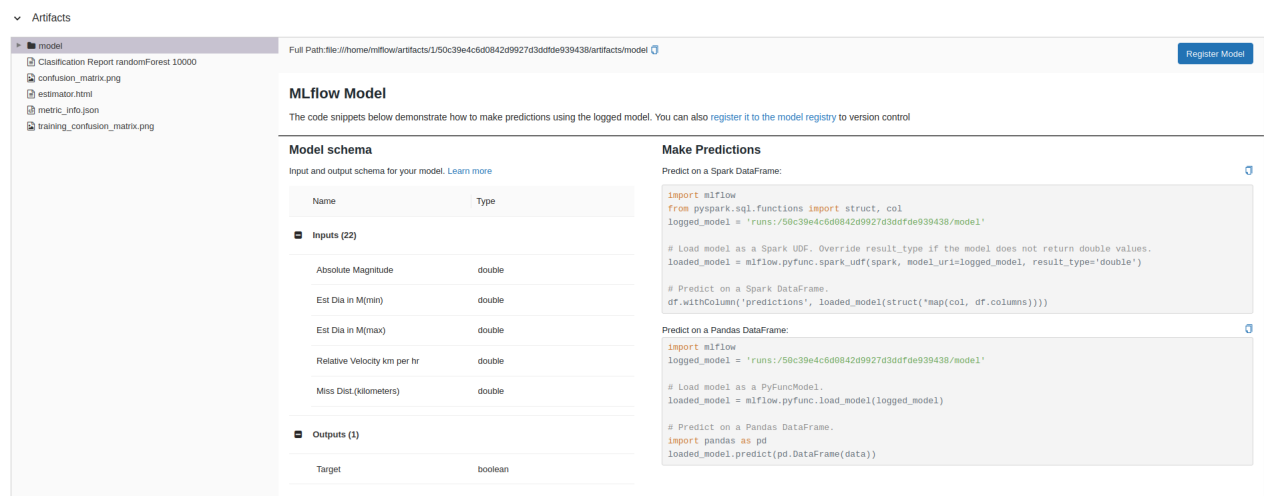


Figura 6.32.: Artefactos

En la figura 6.32 se encuentran los artefactos como el modelo, la matriz de confusión (de test y de entrenamiento) y un informe del modelo.

### 6.4.3. KNN (Sklearn)

Por último, se decidió evaluar el rendimiento del algoritmo *KNN*, realizando un ajuste del parámetro *k* entre 1 y 19 vecinos.

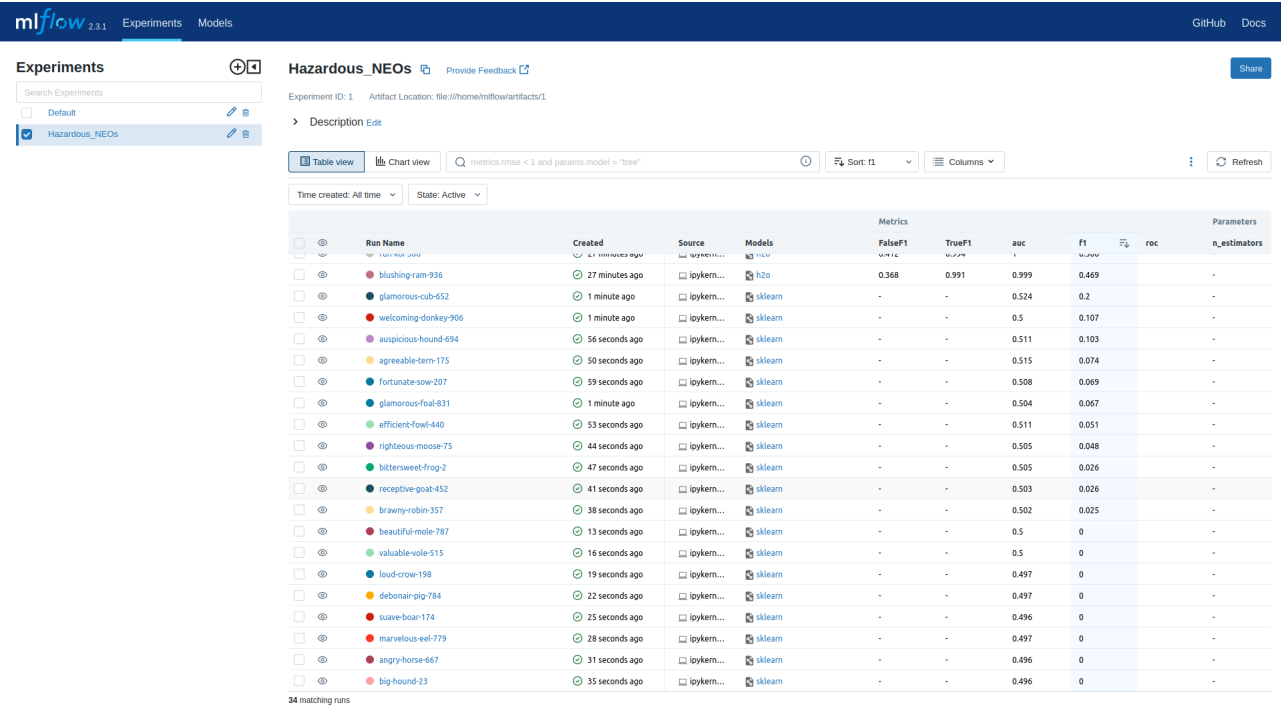
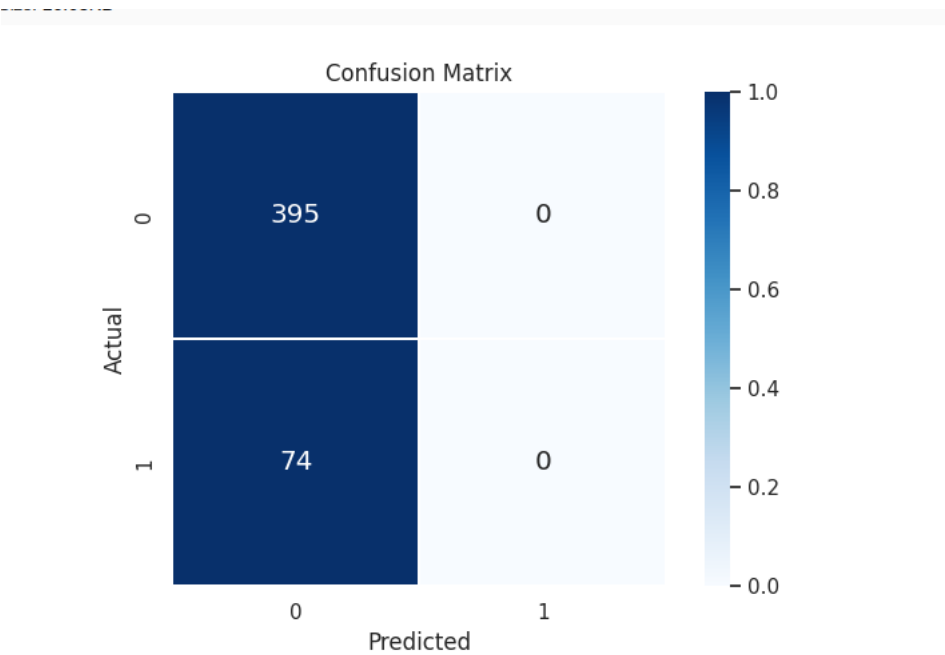


Figura 6.33.: Nuevos modelos KNN

Como se puede apreciar, en la figura 6.33, las métricas obtenidas son bastante peores con el algoritmo KNN, llegando a obtener el valor 0 en la métrica *f1 score*.

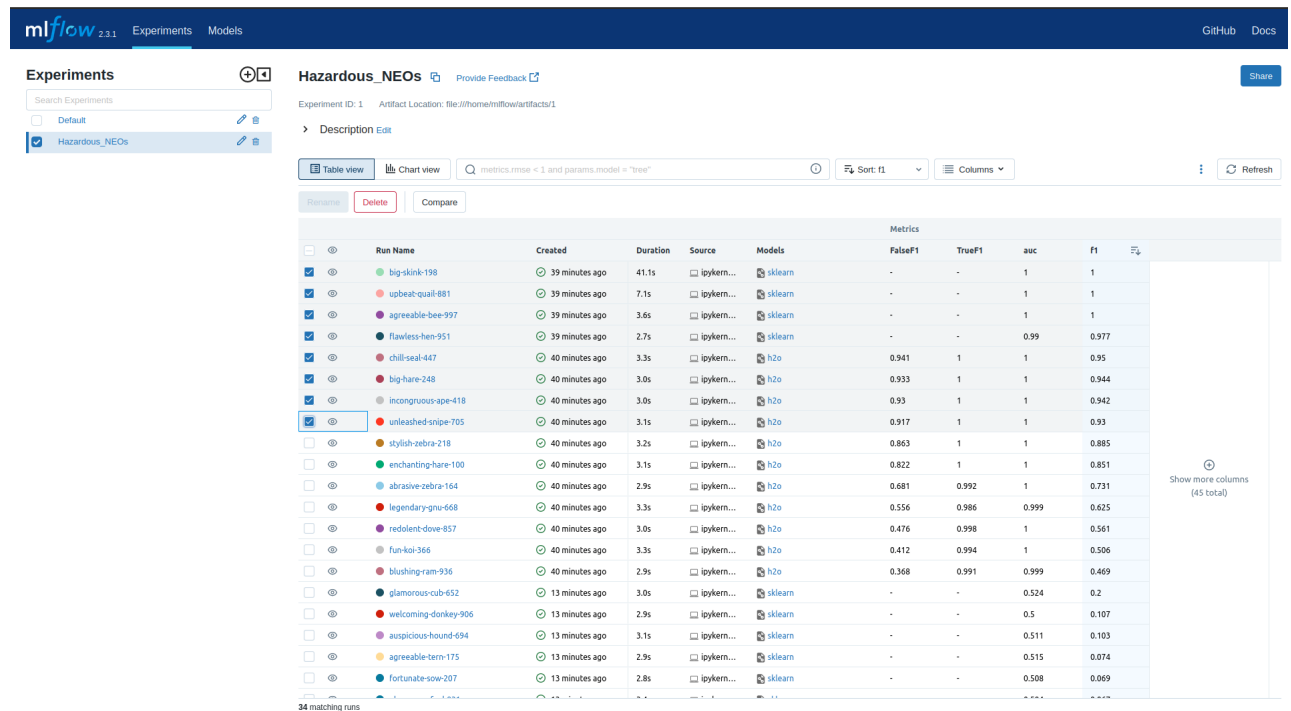


**Figura 6.34.:** Matriz confusión

Esto puede ser observado, en la figura 6.34, en el ejemplo destacado, puede ser observado en las matrices de confusión generadas, siendo todas las instancias clasificadas como falso.

## 6.5. Selección del modelo

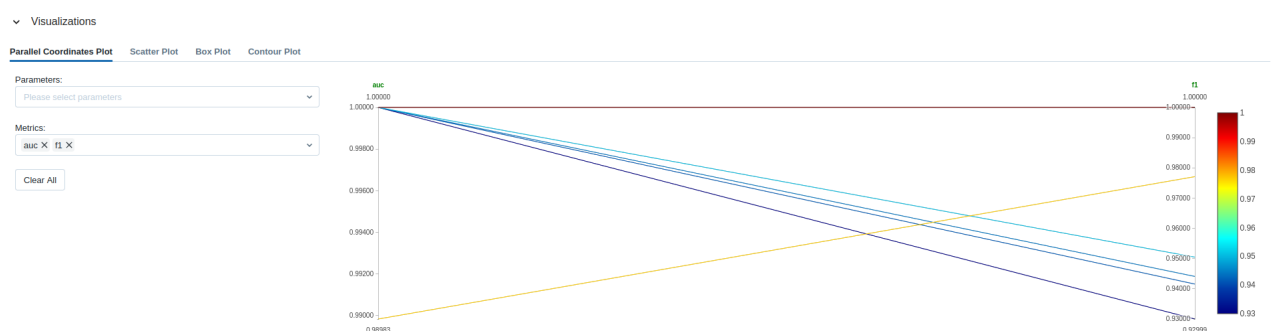
Gracias a todo el registro de modelos, parámetros y métricas podremos realizar la selección del modelo de forma sencilla y clara. Para la selección del modelo se usa la interfaz que permite comparar modelos en MLFlow. Para esto se ordenan los modelos por la métrica que se considere de mayor importancia y se selecciona el número que se considere adecuado.



**Figura 6.35.:** Mejores modelos

Como se puede ver en la figura 6.35, en este caso, se realiza una primera selección de los 8 primeros que tienen un f1 mayor que 0.9.

A continuación, se configura en una gráfica la comparación entre los modelos obtenidos. Se puede observar la combinación de ambas métricas para los modelos seleccionados. Además, si se tratará de modelos pertenecientes al mismo algoritmo, se podría añadir los parámetros que les diferencien.



**Figura 6.36.:** Gráfico para haer compraciones

En la figura 6.36, se puede realizar otra comparativa a nivel de tabla, en las cuales encontramos descritas los runs en los que han sido entrenados los modelos, sus parámetros, métricas y etiquetas asignadas a los modelos.

## 6.5 Selección del modelo

Run details								
Run ID:	148dd65de7a9423ea1356c3...	518d3d62d7744c58fe7ff75...	9825dcb607b7486a83fe4ab...	26523debcca445309358e07...	d3932f9808734ffc8d47ece5...	ee89fb5ed0994dada6404b8...	bef79ebe32204728911fa23...	50c39e4c6d0842d9927d3dd...
Run Name:	unleashed-snipe-705	incongruous-ape-418	big-hare-248	chill-seal-447	flawless-hen-951	agreeable-bee-997	upbeat-quail-881	big-skink-198
Start Time:	2023-05-14 01:34:18	2023-05-14 01:34:24	2023-05-14 01:34:05	2023-05-14 01:34:15	2023-05-14 01:34:47	2023-05-14 01:34:50	2023-05-14 01:34:54	2023-05-14 01:35:01
End Time:	2023-05-14 01:34:21	2023-05-14 01:34:27	2023-05-14 01:34:08	2023-05-14 01:34:18	2023-05-14 01:34:50	2023-05-14 01:34:54	2023-05-14 01:35:01	2023-05-14 01:35:42
Duration:	3.1s	3.0s	3.0s	3.3s	2.7s	3.6s	7.1s	41.1s

**Figura 6.37.:** Conjunto de runs de entrenamiento

Parameters				
Show diff only				
min_samples_leaf	1	1	1	1
min_samples_split	2	2	2	2
min_weight_fraction_leaf	0.0	0.0	0.0	0.0
n_estimators	10	100	1000	10000
n_jobs	-1	-1	-1	-1
oob_score	False	False	False	False
randomState	1606323526	1606323526	1606323526	1606323526
random_state	None	None	None	None
testSize	0.1	0.1	0.1	0.1
verbose	0	0	0	0
warm_start	False	False	False	False

**Figura 6.38.:** Parámetros de los modelos

Metrics				
Show diff only				
FalseF1	0.917	0.93	0.933	0.941
RandomForestClassifier_score_xTest				0.994
TrueF1	1	1	1	1
auc	1	1	1	0.99
f1	0.93	0.942	0.944	0.95
f1_score_xTest				0.977

**Figura 6.39.:** Métricas de los modelos

Tags				
Show diff only				
estimator_class	sklearn.ensemble._forest.Ra...			
estimator_name	RandomForestClassifier	RandomForestClassifier	RandomForestClassifier	RandomForestClassifier

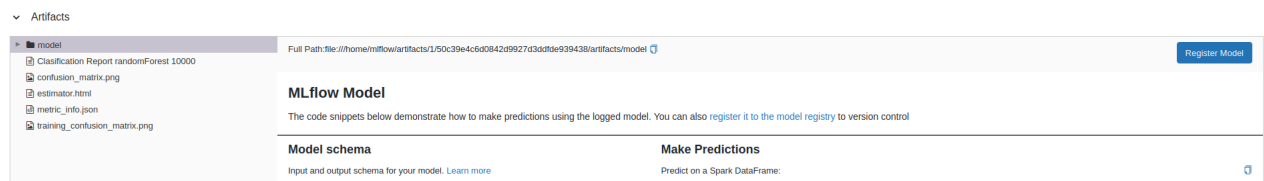
**Figura 6.40.:** Etiquetas

En las figuras 6.37 a 6.40 se observan distintas comparaciones entre los modelos.

Etiquetas para permitir una búsqueda rápida dentro de los modelos generados.

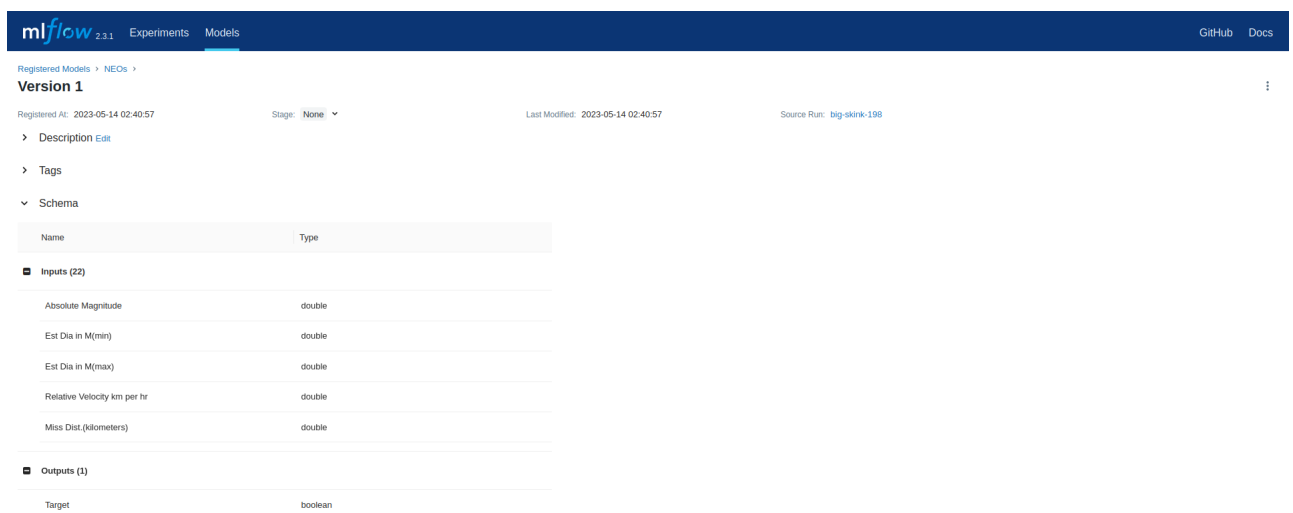
Con toda la información recogida se selecciona el *random forest* de 10000 árboles por tener las mejores métricas entre los modelos registrados. En la selección del modelo, además de las métricas, se pueden tener en cuenta otros factores como el tiempo de predicción o el peso del modelo.

Una vez el modelo ha sido seleccionado debemos registrarlo como modelo. Es recomendable realizar esta acción desde la interfaz para evitar errores a la hora de realizar sobre un código de entrenamiento, este código puede ser ejecutado más veces, lo que puede registrar modelos no óptimos. Los modelos son registrados en el apartado de artefactos, teniendo seleccionado el directorio con el modelo.



**Figura 6.41.:** Registro modelo

En la figura 6.41 se muestra el registro del modelo desde el interfaz. Posteriormente se asigna este a un conjunto de modelos nuevo a existente.

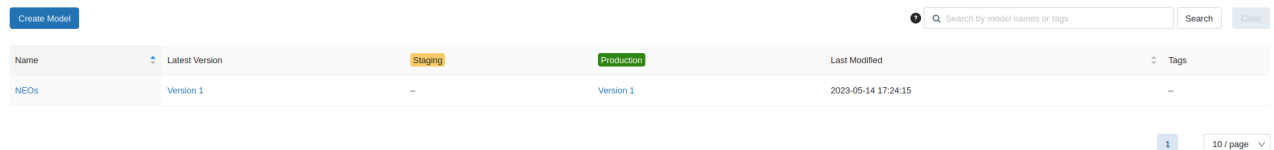


**Figura 6.42.:** Modelo registrado

En la figura 6.42 se puede observar el modelo registrado en su versión 1 con el nombre NEOs, su *signature*, es decir sus parámetros de entrada y salida.

## 6.6. Puesta en producción

Lo primero es marcar el modelo que se va a exportar como modelo en producción. Esto no tienen ningún efecto sobre el modelo, pero sirve para poder mostrar claramente que el modelo estará siendo utilizado, aporta trazabilidad al sistema y la posibilidad de realizar el mantenimiento de su ciclo de vida.



The screenshot shows the MLflow Models Registry interface. At the top, there is a 'Create Model' button and a search bar. Below, a table lists the models. The table has columns: Name, Latest Version, Staging, Production, Last Modified, and Tags. The 'NEOs' model is listed with 'Version 1' as the latest version. The 'Production' column is highlighted in green, indicating the model is in production. The 'Last Modified' date is 2023-05-14 17:24:15. At the bottom right, there is a pagination control showing '1' of 10 pages.

Name	Latest Version	Staging	Production	Last Modified	Tags
NEOs	Version 1	-	Version 1	2023-05-14 17:24:15	-

**Figura 6.43.:** Modelo en versión 1 registrado

En la figura 6.43 se aprecia que la versión 1 del modelo NEOs está siendo usada en producción.

Los modelos almacenados pueden ser explotados de distintas maneras para su consumo. La más simple es a través de la librería de mlflow. Para ello se tiene que establecer una conexión con el servidor de modelos y con el repositorio de artefactos como se muestra en las figuras.

---

### Algoritmo 6.1 Obtencion modelo MLflow

---

```
os.environ["MLFLOW_TRACKING_URI"]="http://ip:puerto"
model=mlflow.sklearn.load_model("models:/NEOs/1")
```

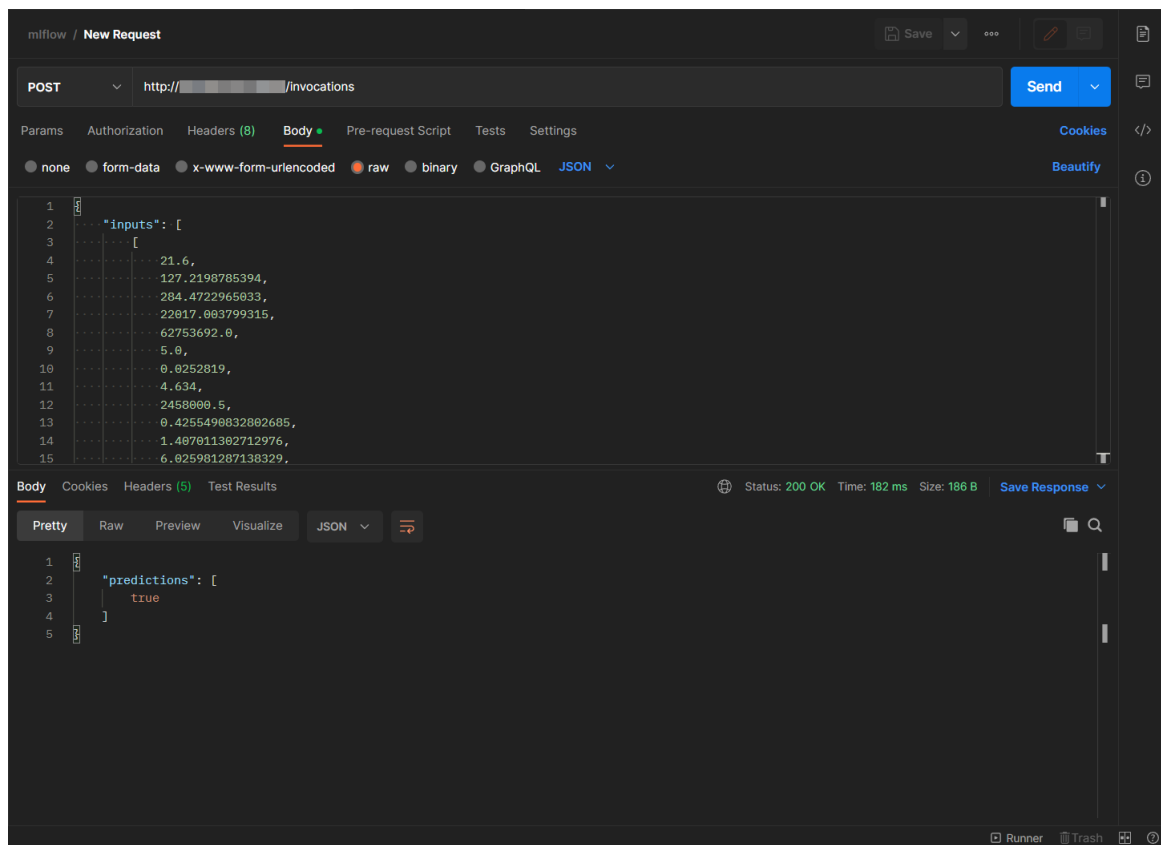
---

```
> /home/alonso/Documentos/TFG/mlflow_env/bin/python -W ignore /home/alonso/Documentos/TFG/predict.py
RandomForestClassifier(n_estimators=10000, n_jobs=-1)
[ True False True ... False False False]
```

**Figura 6.44.:** Modelo usado desde Python

En la anterior 6.44 se observa el modelo obtenido y la salida de este.

Otra opción es exportar el modelo como servicio a través de la CLI (*Command Line Interface*) de MLflow. Dentro de esta se dan varias opciones como el despliegue a través de contenedores Docker o mediante los paquetes gestionados por entornos virtuales ya instalados. Para desplegar el modelo sobre un contenedor es necesario generar el Dockerfile a través del comando: *mlflow models generate-dockerfile*, indicando como opciones el modelo a desplegar a través de su *URI*, y donde será almacenado el archivo Dockerfile, junto con el modelo para el despliegue. Para desplegar el modelo como servicio con un gestor de paquetes instalado en el sistema se deberá usar el comando *mlflow serve*, indicando el modelo a desplegar, el puerto y la red.



**Figura 6.45.:** Consulta con Postman

En la figura 6.45 se puede observar la respuesta del servicio a una petición con una entrada del *dataset* original y la predicción del modelo correspondiente.

## 6.7. Monitorización del modelo

En un escenario completo el modelo debe ser monitorizado para observar su rendimiento a lo largo del tiempo. Debido a que se ha utilizado un *dataset* cerrado, no se generarán nuevos datos, ni existe una evolución de estos, por lo que no es posible realizar un seguimiento. En el escenario completo, se ha de suponer que el modelo empieza a sufrir de *data drifting* y que por lo tanto va a ser necesario actualizarlo cuando disminuya su rendimiento por debajo del umbral establecido. Para esto se deberán registrar los resultados del modelo y evaluar las diferencias con los valores reales.

## 6.8. Reentrenamiento o sustitución del modelo

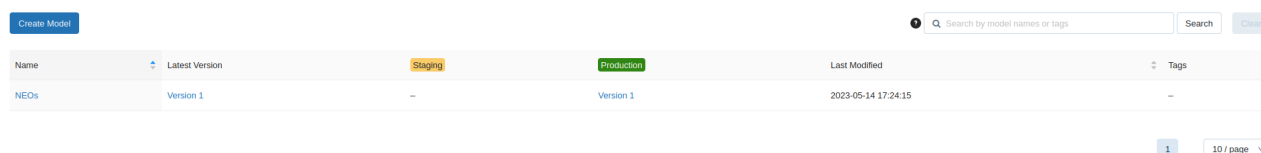
En este punto se repetirá el proceso para la generación de modelos anteriormente descrita y se seleccionará un nuevo modelo como sustituto. De nuevo no es



## 6.8 Reentrenamiento o sustitución del modelo

posible replicar esta fase por la falta de unos datos que cambien con el tiempo. Pero se puede simular la obtención la selección de un segundo modelo, para remplazar al antiguo. Además, el modelo antiguo pasara al estado *archived* y el nuevo a producción.

En este caso se selecciona el segundo mejor modelo de los seleccionados antes y se registra como segunda versión de NEOs.

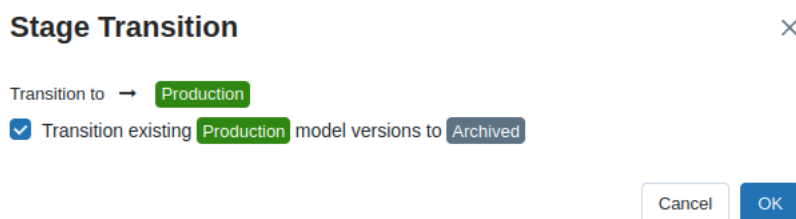


The screenshot shows a table with columns: Name, Latest Version, Stage, Production, Last Modified, and Tags. There is a 'Create Model' button at the top left and a search bar at the top right. The table contains one row for 'NEOs' with 'Version 1' in the 'Latest Version' column, a '-' in the 'Stage' column, 'Version 1' in the 'Production' column, and '2023-05-14 17:24:15' in the 'Last Modified' column. The 'Tags' column is empty. At the bottom right, there is a pagination control showing '1' and '10 / page'.

Name	Latest Version	Stage	Production	Last Modified	Tags
NEOs	Version 1	-	Version 1	2023-05-14 17:24:15	-

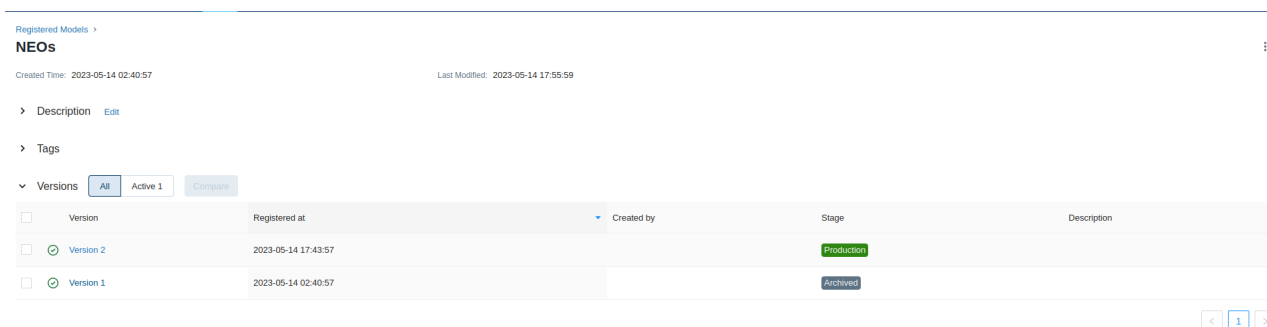
**Figura 6.46.:** Dos modelos registrados

En la figura 6.46 se muestra como la versión 2 esta registrada como modelo también.



**Figura 6.47.:** Transición de estados

En la figura 6.47 se selecciona la opción de pasar a *archived* el modelo anteriormente en producción.



The screenshot shows the details of the 'NEOs' model. It includes a 'Description' section and a 'Tags' section. Below these is a 'Versions' section with tabs for 'All', 'Active 1', and 'Compare'. The 'All' tab is selected, showing a table with columns: Version, Registered at, Created by, Stage, and Description. There are two versions listed: 'Version 2' (registered at 2023-05-14 17:43:57, stage 'Production') and 'Version 1' (registered at 2023-05-14 02:40:57, stage 'Archived'). At the bottom right, there is a pagination control showing '1' and '>'.

Version	Registered at	Created by	Stage	Description
Version 2	2023-05-14 17:43:57		Production	
Version 1	2023-05-14 02:40:57		Archived	

**Figura 6.48.:** Nuevos estados de los modelos

En la figura 6.48 se muestra el estado de los modelos registrados en el sistema, la versión 1 se encuentra archivada y la segunda en producción.



## 7. Análisis de impacto.

Durante todo el trabajo se ha desarrollado una metodología para poder gestionar con los proyectos ML de la mejor manera posible. Para ello se ha revisado porque es necesario una metodología como MLOps y cuáles son sus orígenes, cuales es el problema que se quiere solucionar es particular, el análisis de las herramientas que soportan la metodología, como ha sido despegada la herramienta, la metodología en específico a seguir, ilustrada a través de un proyecto practico.

MLOps surge de la necesidad adaptar los desarrollos ágiles a los desarrollos de ML. Aunque la computación moderna y la inteligencia artificial surgieran de manera simultanea la primera consiguió buenos resultados desde el principio y por lo tanto fue favorecida. Esto ha llevado a una aparición más tardía de las metodologías de desarrollo en ML. MLOps surge para poder solucionar los mismos problemas que DevOps, pero para ML. MLOps ha demostrado tener un rendimiento superior para gestionar el ciclo de vida de desarrollos ML que otras técnicas [Com19]. Además, tiene una organización en niveles lo que permite gestionar el nivel de automatización dependiendo del proyecto, no es necesario la automatización de un proyecto del que solo se quiere un modelo puntual, pero si que lo es de uno que se requiera una precisión constante.

Entre las arquitecturas disponibles se ha seleccionado la más adecuada para el problema que se presenta. MLflow permite la gestión de proyectos ML a nivel 0, aunque a través de herramientas externas es posible el uso de niveles más altos y por lo tanto de más optimización. Para gestionar el despliegue se ha usado la Docker, que permite gestionar las dependencias con el sistema operativo de manera más sencilla y ligera que una maquina virtual, aumentando la replicabilidad del sistema. También se decide que la base de datos que soporte el sistema se gestionado por PostgreSQL y que el repositorio de artefactos requerido este alojado en un NAS y su acceso se realice mediante NFS (UNIX) y SMB (Windows). Esta arquitectura permite el desarrollo proyectos desde cualquier maquina con acceso al servidor MLflow, Postgres y repositorio de modelos.

Una vez conocida seleccionada la herramienta y la arquitectura que seguirá es necesario seguir, es necesario seguir la metodología a seguir. Esta metodología se ha adaptado para aprovechar la herramienta de la mejor manera posible, aprovechando sus puntos fuertes y minimizando los puntos débiles. En este caso se provecha la flexibilidad que proporciona MLflow en cuanto al desarrollo, permite múltiples alternativas de manera nativa y da soporte básico al resto. Aunque no tenga tanta automatización como otras plataformas se puede remediar con herramientas externas como ETLs (Extract, Transform, Load) para el tratamiento de los datos, AutoML de plataformas en las que si que esta dispo-

nible de manera abierta (H2O). Otro de los pilares de MLOps es la sustitución de modelos cuando sus capacidades decaen. En este caso, para reconocerlo es necesario almacenar todas las consultas hacia los modelos. Esta metodología ha sido desarrollada con el objetivo, de proporcionar trazabilidad, replicabilidad y agilidad a los proyectos ML. La trazabilidad es aportada por el registro de todo lo relacionado con los modelos producidos, que permite seguir haciendo pruebas con distintos parámetros sin temor a perder el trabajo ya realizado, con el objetivo de producir mejores modelos. La replicabilidad se ha seguido desde el primer momento, en todo momento habiendo varias instancias replicadas de los servidores para poder realizar pruebas de concepto y arquitectura sobre ellos. El sistema en sí permite la replicabilidad al guardar todo lo relacionado con cómo se han realizado los experimentos y runs, que permite en cualquier momento replicar cualquiera en el caso de ser necesario. La agilidad de la metodología es directamente heredada de DevOps, al tratarse de una extensión a esta. También se replican los escenarios de desarrollo y producción, en este caso entrenamiento y producción.

En cuanto a su relación con el desarrollo sostenible y los objetivos de la ONU este proyecto se alinea con el noveno objetivo: “Industria, Innovación e Infraestructuras”. MLOps es una metodología desarrollada en los últimos 8 años para la gestión de los proyectos, introducida por primera vez por Google [HZRS16], cuyo objetivo es la agilización de los proyectos ML, para obtener mejores resultados de ellos. No solo provoca una innovación su uso, sino que esta destinada al servicio de la investigación. Reduce la necesidad de tener que realizar varias veces el mismo entrenamiento para la obtención de los datos, lo que reduce el gasto de energía, en la parte más costosa del proceso.

## 8. Conclusiones.

En este trabajo se ha cumplido los objetivos que en un principio se había propuesto: se ha seleccionado Mlflow para poder hacer uso de la metodología MLOps, se ha desplegado sobre una arquitectura Docker, se ha diseñado una metodología que siga MLOps y que se adapte a Mlflow y C.R.I.D.A., por ultimo se ha comprobado mediante un caso práctico.

- Aunque Mlflow solo tenga nivel MLOps 0, a través de otras herramientas se puede obtener la automatización buscada.
- Gran parte de las automatizaciones de las herramientas con nivel 2, dependen de usar los paquetes de servicios al completo de las empresas que las ofrecen. Por el contrario, el uso de una herramienta de nivel 0, permite la selección flexible de la forma de automatización.
- Se ha construido una arquitectura resistente a fallos, a través de las funciones que da Docker para el almacenamiento permanente y la recuperación de servicios tras un error.
- La metodología soporta herramientas externas para la automatización, como pueden ser las ETLs (Extract, Transform, Load) para la obtención de datos o el uso de AutoML de H2O para la agilización de los entrenamientos.
- El uso de la herramienta a la hora de realizar entrenamientos es simple. Solo es necesario añadir las ordenes necesarias, al código de entrenamiento, para que se almacene el modelo, parámetros, métricas y artefactos.
- El buen funcionamiento de la herramienta depende del seguimiento de la metodología que hagan los usuarios.
- Permite la entrega rápida y continua de modelos, y su reentrenamiento o sustitución sencilla.
- La creación de los modelos es replicable. No solo es útil para ofrecer trazabilidad, si no, también para exponer y explicar resultados.



## 9. Líneas futuras de mejora

Este proyecto cuenta con dos principales deficiencias técnicas y hereda los retos de los niveles más bajos de MLOps. Como ya se ha comentado durante el diseño de la arquitectura, el uso de HDFS ha generado muchos problemas y ha sido descartado, aunque tuviera mejores características que la combinación de NFS y SMB. Tampoco se cuenta con gestión de usuarios funcional, debido a que en el momento de realización de este TFG es una característica experimental e incompleta. Los niveles más bajos no cumplen con CI/CD por lo que el trabajo no automatizado recae sobre los desarrolladores.

HDFS cuenta con varias ventajas con respecto a la combinación de NFS/SMB. La primera y más importante, es un sistema totalmente distribuido y resistente a fallos, por el contrario, NFS/SMB son protocolos para compartir archivos. HDFS puede seguir funcionando con uno de sus nodos caído debido a la replicación de información en los distintos nodos, pero NFS/SMB en el caso de una caída del servidor dejan de funcionar. La estructura para escribir y leer datos sigue totalmente Write-Once-Read-Many. La segunda ventaja es el acceso nativo a las herramientas nativas del ecosistema Hadoop como Spark o Kafka. Aunque, parte de estas herramientas sí que se pueden usar fuera del ecosistema, como Spark para realizar procesamiento MapReduce, no se cuenta con la misma facilidad para ello. Por último, otra ventaja es que las conexiones hacia HDFS no necesitan distintos protocolos dependiendo del sistema operativo, lo que también simplifica la arquitectura.

La gestión de usuarios está siendo desarrollada en este momento en Mlflow y todavía es experimental, no hay una jerarquía, roles, ni unas operaciones permitidas y restringidas. La gestión usuarios no es un tema únicamente de seguridad o privacidad, si no de organización y de minimización del riesgo de todos los errores posibles en este sistema. Una solución temporal hasta la inclusión de final de los usuarios en Mlflow el uso de proxy inverso como NGINX, aunque esto solo trata el acceso a la herramienta, no el acceso a los distintos experimentos y ejecuciones de cada uno de los proyectos, por no hablar de los permisos de lectura, escritura o borrado.

Como se ha comentado varias veces durante el desarrollo de este trabajo, el nivel MLOps de la herramienta es 0 de partida y 1 se usan herramientas externas como herramientas ETL para la automatización de parte de los procesos. Aun así, gran parte de la gestión recae sobre los desarrolladores al no haber un automatismo generalizado. En el caso de que un proyecto solo se quiera realizar una vez y no sea necesario replicarlo se puede optar por el nivel 0. En el caso de que se quiera usar el nivel 1 es necesario que la limpieza de datos y los entrenamientos queden automatizados para poder lanzarlos de manera automática una vez que

se considere necesario con la posibilidad de definir un *trigger*. Toda la gestión de esto pipelines se tiene que realizar de manera manual, lo que se puede convertir en un problema según aumenta su número. [Clo21b].

Por último, el sistema es dependiente del buen uso que le den sus usuarios, es necesario que sigan las buenas prácticas expuestas en la metodología para un correcto aprovechamiento de la herramienta.



## A. Apendice

### A.1. Archivos de configuración docker necesarios para el despliegue de la arquitectura.

#### A.1.1. Dockerfile Mlflow

##### A.1.1.1. Dockerfile Mlflow

```
FROM ubuntu:22.04
#Utilidades
RUN apt-get update
RUN apt-get install -y wget
RUN apt-get install -y libpq-dev
#Python
RUN apt-get -y install python3
# Usuario sin privilegios
# --login: Fuerza a la ejecucion de los archivos de configuración
# -c Permite que todos los argumentos vengan dentro del mismo string
# y los separa
SHELL ["/bin/bash", "--login", "-c"]
ENV USER=devteam
ENV HOME=/home/${USER}
# Recomendación para designar usuarios sin privilegios (UID, GID)
ENV UID 1000
ENV GID 100
# Crea el usuario sin contraseña,
# fuera de la lista de sudoers con los parametros anteriormente
# definidos
RUN adduser --disabled-password \
    --gecos "Non-root user" \
    --home ${HOME} \
    --uid ${UID} \
    --gid ${GID} \
    ${USER}
# Copiar ficheros necesarios
COPY entrypoint.sh ${HOME}
RUN chown ${UID}:${GID} ${HOME}/entrypoint.sh && \
    chmod 0755 ${HOME}/entrypoint.sh
COPY .bashrc ${HOME}
RUN chown ${UID}:${GID} ${HOME}/.bashrc
RUN mkdir /home/MLFLOWS
```

```

RUN chmod 0770 /home/MLFLOWS
RUN chown devteam:users /home/MLFLOWS
# Conda
USER ${USER}
ENV CONDA_DIR=${HOME}/conda
RUN wget --quiet \
    https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh
    -O ~/miniconda.sh
RUN chmod 0700 ~/miniconda.sh
RUN bash ~/miniconda.sh -b -p ${CONDA_DIR}
RUN rm ~/miniconda.sh
ENV PATH=${CONDA_DIR}/bin:$PATH
RUN echo ". ${CONDA_DIR}/etc/profile.d/conda.sh" >> ~/.profile
RUN conda init bash
#Entorno
RUN mkdir ${HOME}/artifacts
RUN mkdir ${HOME}/mlruns
ENV PROYECTOS_DIR ${HOME}/mlflowProjects
RUN mkdir ${PROYECTOS_DIR}
WORKDIR ${PROYECTOS_DIR}
#Configurar conda
ENV ENV_PREFIX=mlflow_env
RUN conda update --name base --channel defaults conda
RUN conda create -n $ENV_PREFIX
RUN conda clean --all --yes
RUN conda activate ${ENV_PREFIX} && \
    python3 -m pip install --upgrade pip && \
    pip install mlflow && \
    pip install psycpg2-binary && \
    pip install hdfs
ENTRYPOINT ["/home/devteam/entrypoint.sh"]
# Lanzamiento servidor
EXPOSE 80

```

#### A.1.1.2. Entrypoint

```

#!/bin/bash
set -e
conda run -n $ENV_PREFIX $@

```

#### A.1.1.3. .bashrc

```

alias las='ls -las'
alias cls='clear'

```

A.1 Archivos de configuración docker necesarios para el despliegue de la arquitectura.

---

### A.1.2. Dockerfile Postgres

FROM postgres:15.2

### A.1.3. Docker compose y .env

#### A.1.3.1. Compose.yaml

```
version: "3.5"
services:
  mlflow-mlflow:
    # Localizacion de la imagen nombre de la imagen y del contenedor
    build: ./docker/mlflowDocker
    image: mlflow_image
    container_name: mlflow
    depends_on:
      - mlflow-postgresql
    environment:
      - MUID=$UID
      - MGID=$GID
      - ARTIFACTS=$ARTIFACTS
    # Dependencias y politica de reinicio
    restart: always
    # Exportar los puertos (abrirlos)
    ports:
      - 80:80
    # Comando de lanzamiento (Explicado en profundidad en Intalacion.md)
    command: mlflow server -h 0.0.0.0 -p 80 \
      --backend-store-uri $BACKEND \
      --default-artifact-root $ARTIFACTS/artifacts
    volumes:
      #- /artifacts:/artifacts:rw
      - /home/MLFLOWS:/home/MLFLOWS:rw
    # networks:
    #   - frontend

  mlflow-postgresql:
    build: ./docker/postgreSQL
    image: postgresql_image
    container_name: postgres
    environment:
      - MUID=$UID
      - MGID=$GID
      - POSTGRES_DB=mlruns
      - POSTGRES_USER=mlflow
      - POSTGRES_PASSWORD=$POSTGRES_PASSWORD
      - PGDATA=/var/lib/postgresql/data/database
    restart: always
```

```
ports:
  - 543210:543210
volumes:
  - /home/devteam/postgres/mlflowDatabase:/var/lib/postgresql/data:rw
```

#### A.1.3.2. .env

```
BACKEND=postgresql://mlflow:${POSTGRES_PASSWORD}@192.168.1.2:543210/mlruns
ARTIFACTS=file:///home/MLFLOWS
```

### A.1.4. Manual instalación.

#### A.1.4.1. Instalación MLFlows

**Docker** Esta sección esta extraída de la documentación oficial de instalación de Docker, que se pueden encontrar en <https://docs.docker.com/engine/install/ubuntu/#set-up-the-repository> y <https://docs.docker.com/desktop/install/ubuntu/>. Es recomendable copiar los comandos directamente desde los links anteriores. Si en alguno de los pasos se devuelve un error con una segunda ejecución de todos los pasos de esta sección se arregla, esto es causado por que algunos de los paquetes modifican los archivos de configuración necesarios. Posiblemente lo cause la instalación de docker.io. Si se interrumpe la instalación de los paquetes que se obtienen desde el nuevo repositorio no es necesario volver a configurar los paquetes solo ejecutar.

- Base docker

```
sudo apt install docker.io
```

- Incluir el repositorio docker

```
# Update the apt package index and install packages to allow apt
# to use a repository over HTTPS
# Obtención de librerías necesarias para la instalación
sudo apt-get update
sudo apt-get install \
    ca-certificates \
    curl \
    gnupg

# Add Dockers official GPG key
# Descarga de la clave GPG para conectarse al repositorio de docker
sudo install -m 0755 -d /etc/apt/keyrings curl -fsSL \
    https://download.docker.com/linux/ubuntu/gpg \
    | sudo gpg --dearmor -o \
    /etc/apt/keyrings/docker.gpg
sudo chmod a+r /etc/apt/keyrings/docker.gpg
```

## A.1 Archivos de configuración docker necesarios para el despliegue de la arquitectura.

---

```
# Use the following command to set up the repository
# Añadir el repositorio a la lista de repositorios en los que se
#confía
echo \
"deb [arch="$(dpkg --print-architecture)"\
signed-by=/etc/apt/keyrings/docker.gpg]\
https://download.docker.com/linux/ubuntu \
"$(. /etc/os-release && echo "$VERSION_CODENAME)" stable" | \
sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

### ■ Docker engine

```
sudo apt-get install \
docker-ce \
docker-ce-cli \
containerd.io \
docker-buildx-plugin \
docker-compose-plugin
```

### ■ Docker desktop

```
sudo apt-get install ./docker-desktop-<version>-<arch>.deb
```

**Docker sin sudo** Las siguientes instrucciones están extraídas desde el siguiente link: <https://docs.docker.com/engine/install/linux-postinstall>.

- Crear grupo docker si no está creado, la manera más fácil de realizar esto es directamente intentar crear el grupo, en el caso de que ya exista fallara el comando, mostrando como motivo el grupo ya existe.

```
sudo groupadd docker
```

- Se añade al grupo Docker el usuario que vaya a usar Docker

```
sudo usermod -aG docker $USER
```

- Se aplican los cambios

```
newgrp docker
```

- Se reinicia el terminal para que se queden aplicados los cambios.
- Comprobar que el usuario este añadido en el grupo, el nombre del grupo ha de aparecer.

```
groups
```

**NFS**

- El servidor nfs puede ser instalado donde de desee. Para instalar el servidor es necesario usar la librería. El servidor es el encargado de mantener los datos, concretamente los artefactos que han de ser almacenados en un sistema de ficheros.

```
sudo apt install nfs-kernel-server
```

- En el caso de querer usar el NAS como servidor NFS es necesario añadir al usuario a los que tengan permiso en la carpeta, con su IP y permisos de lectura y escritura.
- Los clientes de nfs han de estar instalados y añadidos a /etc/fstab para que puedan ser montados en el arranque del sistema. El directorio cliente ha de estar localizado en la misma ruta para la máquina que realice labores de servidor de mlflows y para el cliente de mlflows. Es necesario la instalación de la librería nfs-common.

```
sudo apt install nfs-common
```

- Para montar las carpetas es se monta se usa el comando mount IP\_servidor:/ruta/servidor /ruta/cliente

```
mount 192.168.1.2:/ruta/MLFLOWS /ruta/MLFLOWS
```

- Para garantizar el acceso tanto de los clientes de mlflows como del servidor mlflows los permisos unix del directorio han de ser los siguientes:

- Permisos drwxrws\_\_\_. Directorio con todos los permisos para su propietario y con todos para el grupo propietario. Además, todos los usuarios pertenecientes al grupo usaran el grupo propietario para realizar las escrituras dentro del sistema de archivos nfs.

```
chmod 2770 /ruta/MLFLOWS
```

- Propietarios del sistema de archivos. Usuario pude ser cualquiera. El grupo a ser uno al que tengan acceso todos los usuarios que se requiera que usen mlflow, por ejemplo mlflow.

```
chown owner:mlflow /ruta/mlflows
```

**SMB** Para los usuarios de mlflows que quieran usar Windows se montara el directorio de artefactos con SMB en vez de con NFS. Es necesario realizar dos pasos para poder montar una carpeta con la misma ruta que en Linux.

1. El primer paso es importar la carpeta por el protocolo SMB.

```
net use M: \\192.168.1.2\MLFLOWS contraseña /USER:usuario /PERSISTENT:NO
```

2. El Segundo paso consiste en sincronizar la carpeta que tenga la dirección deseada con el volumen recién creado. Es necesario usar cmd con permisos de administración.

```
mklink /d \ruta\MLFLOWS M:
```

**Lanzamiento servidor** Es necesario dar un valor a la variable `POSTGRES_PASSWORD` porque es la variable, que posteriormente, leerá Docker compose para configurar la contraseña de la base de datos postgres y la contraseña con la mlflow se conecta.

- Ejecutar desde la carpeta raíz del proyecto
  - Cuando no haga falta volver a construir las imágenes

```
export POSTGRES_PASSWORD=password && \  
docker compose up -d
```
  - Cuando haga falta volver a construir las imágenes.

```
export POSTGRES_PASSWORD=password && \  
docker compose up -d -build
```
- En el caso de que fuera necesario volver a construir las imágenes sin haber realizado ningún cambio en los Dockerfile se pueden volver a construir de la siguiente manera. Después se vuelven a lanzar con los comandos anteriores.

```
docker compose build no-cache
```
- Comprobar que los servicios estén levantados, debería aparecer dos servicios con estatus running, uno mlflow y otro postgres.

```
docker ps -a
```
- Detener la ejecución

```
docker compose stop
```
- Eliminar todo lo definido en el archivo compose

```
docker compose down
```

## A.2. Uso de la herramienta

### A.2.1. Manual de usuario

#### A.2.1.1. Conceptos

En esta sección presentamos los diferentes conceptos que se deben comprender para un correcto uso de la herramienta.

1. Experimentos: Se pueden reutilizar para generar en ellos distintas runs. Se suelen agrupar runs con mismo objetivo para comparar los distintos resultados de las runs.

- a) Ejecuciones (runs): Son ejecuciones del programa (se pueden reutilizar runs activas, pero por diseño no lo haría), estas ejecuciones tienen distintos componentes que son almacenados.
  - 1) Etiquetas: Etiquetas para identificar las runs (Proponer sistema de etiquetado: #<nombreProyecto>, #<Algoritmo>, #<Libreria>, #ASupervisado, #ANoSupervisado...).
  - 2) Parámetros: Parámetros utilizados en la ejecución (run). Se debe añadir el nombre del dataset como parámetro.
  - 3) Métricas: Métricas utilizadas para evaluar los resultados (intentar en la medida de lo posible mantener las mismas métricas en todas las ejecuciones).
  - 4) Artefactos: Todos los ficheros que intervienen en la ejecución (inputs, outputs, modelos, ficheros de propiedades...) (cuidado con el artifactStorage, quizás es necesario montar un hdfs).
- 2. Proyectos (se almacenan con los artefactos): Puedes ejecutar un proyecto siempre que contenga un entryPoint.
  - a) MLproject file: Información del proyecto (es importante ya que se puede utilizar luego para realizar la ejecución).
  - b) Entorno: Requisitos del entorno de ejecución (libs por ejemplo).
  - c) Entry points: Fichero o comando utilizado para la ejecución del proyecto (main, comando de ejecución...).
  - d) Modelo: Modelo (pkl).
- 3. Modelos: Directorio con multiples archivos, incluyendo un MLModel. Estos modelos pueden ser desplegados como servicios para ser utilizados por servicios REST.
  - a) Flavors: Estandar utilizado para desplegar los modelos (sklearn, keras, h2o...).
  - b) Model Signature: Descripción de las entradas y salidas del modelo (Uso obligado de Column-based Signature Example).
  - c) Model Input Example: Ejemplo de la entrada de datos (Uso obligado de Column-based Example).
  - d) Artefactos: Con la función MLModel.evaluate() permite generar información relativa a la evaluación, entre esta gráficos y métricas que se almacenan junto al modelo.
- 4. Registro de modelos: Se utiliza para centralizar el almacenaje de los modelos, añadirles versiones, ciclo de vida y otros metadatos (puede registrarse desde el código o desde la UI).
  - a) El ciclo de vida, versionado y mantenimiento de los modelos es muy completo.

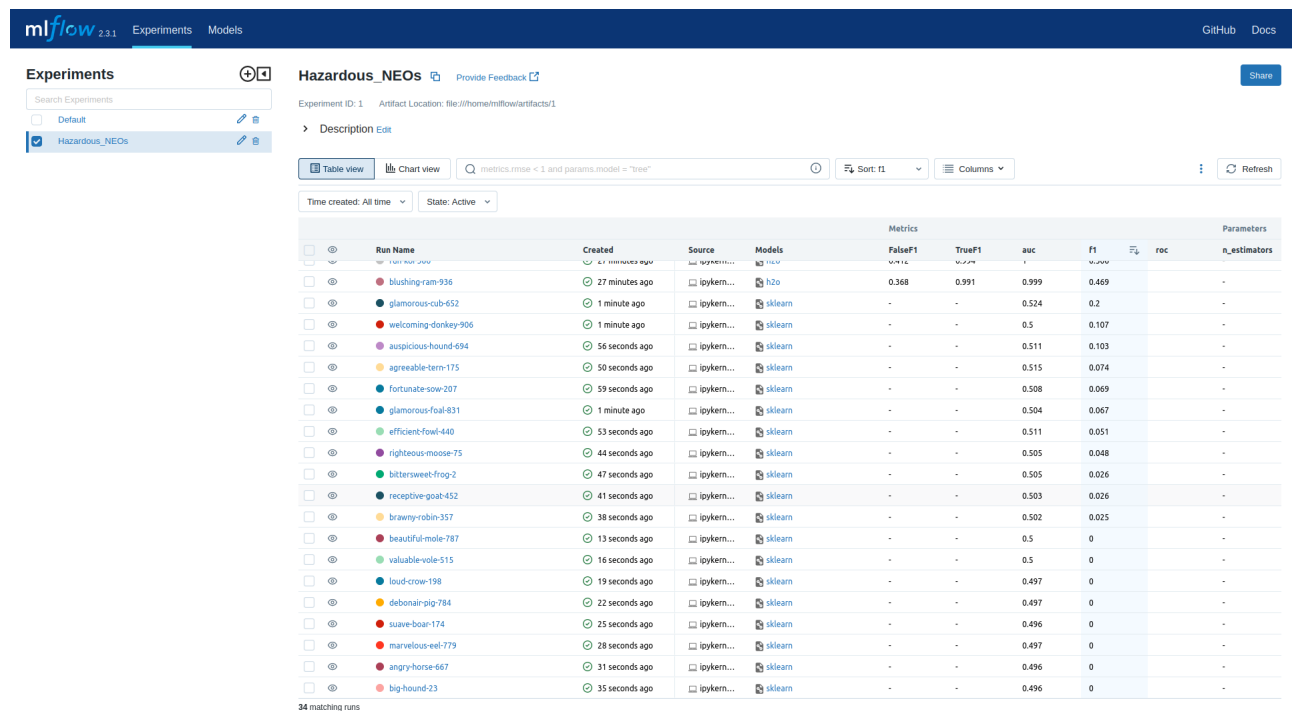


## A.2 Uso de la herramienta

- b) No borrar modelos a no ser que hayan sido creados por error.
  - c) Permite el uso de modelos que no tengan flavors con limitaciones (debes "crear" el flavor tú mismo).
5. Autolog: Registro automático de parámetros, métricas y ciertos artefactos. Solo esta disponible en algunos "flavours" y las métricas guardadas son sobre el conjunto de entrenamiento. Por lo que es necesario guardar las métricas sobre el conjunto de test que se considere oportunas. La "signature" del modelo es preferible realizarla manualmente con el dataset completo.

### A.2.1.2. Interface web de MLFlow

**Experimentos** El registro de los distintos experimentos y ejecuciones se puede ver en la interfaz web del servidor MLFlow. En la imagen podemos observar las distintas ejecuciones asociadas a los experimentos adjuntos.



The screenshot shows the MLFlow web interface for the 'Hazardous\_NEOS' experiment. The interface includes a search bar, a list of experiments, and a table of runs. The table columns are: Run Name, Created, Source, Models, FalseP1, TrueP1, auc, f1, roc, and n\_estimators. The runs are sorted by 'Created' time, showing the most recent runs at the top.

Run Name	Created	Source	Models	FalseP1	TrueP1	auc	f1	roc	n_estimators
blushing-ram-936	27 minutes ago	ipykem...	h2o	0.368	0.991	0.999	0.469	-	-
glamorous-cub-652	1 minute ago	ipykem...	sklearn	-	-	0.524	0.2	-	-
welcoming-donkey-906	1 minute ago	ipykem...	sklearn	-	-	0.5	0.107	-	-
auspicious-hound-694	56 seconds ago	ipykem...	sklearn	-	-	0.511	0.103	-	-
agreeable-term-175	50 seconds ago	ipykem...	sklearn	-	-	0.515	0.074	-	-
fortunate-sow-207	59 seconds ago	ipykem...	sklearn	-	-	0.508	0.069	-	-
glamorous-foal-831	1 minute ago	ipykem...	sklearn	-	-	0.504	0.067	-	-
efficient-fowl-440	53 seconds ago	ipykem...	sklearn	-	-	0.511	0.051	-	-
righteous-moose-75	44 seconds ago	ipykem...	sklearn	-	-	0.505	0.048	-	-
bittersweet-frog-2	47 seconds ago	ipykem...	sklearn	-	-	0.505	0.026	-	-
receptive-goat-452	41 seconds ago	ipykem...	sklearn	-	-	0.503	0.026	-	-
brawny-robin-357	38 seconds ago	ipykem...	sklearn	-	-	0.502	0.025	-	-
beautiful-mole-787	13 seconds ago	ipykem...	sklearn	-	-	0.5	0	-	-
valuable-vole-515	16 seconds ago	ipykem...	sklearn	-	-	0.5	0	-	-
loud-crow-198	19 seconds ago	ipykem...	sklearn	-	-	0.497	0	-	-
debonair-pig-784	22 seconds ago	ipykem...	sklearn	-	-	0.497	0	-	-
suave-boar-174	25 seconds ago	ipykem...	sklearn	-	-	0.496	0	-	-
marvelous-eel-779	28 seconds ago	ipykem...	sklearn	-	-	0.497	0	-	-
angry-horse-667	31 seconds ago	ipykem...	sklearn	-	-	0.496	0	-	-
big-hound-23	35 seconds ago	ipykem...	sklearn	-	-	0.496	0	-	-

**Figura A.1.:** Pestaña de experimentos

Además, pinchando en uno de los experimentos podemos ver los detalles de este y el modelo resultante (si ha sido almacenado).

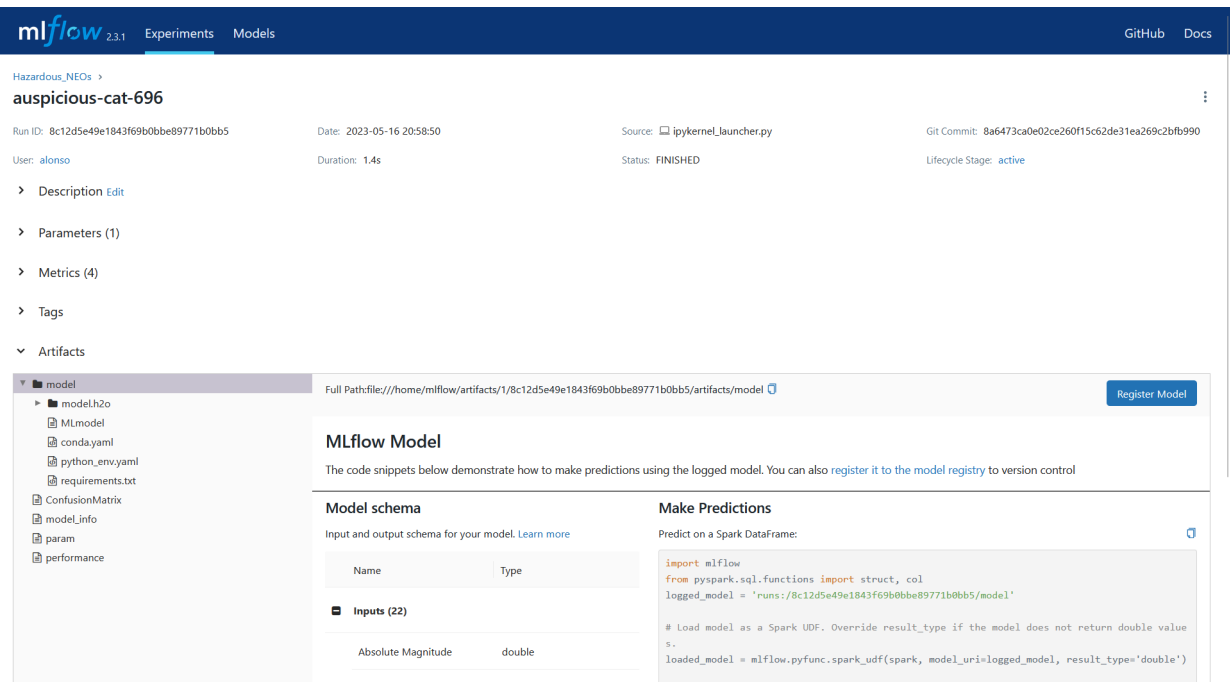


Figura A.2.: Información asociada a una ejecución

Por otra parte, se puede realizar una comparación entre los distintos experimentos y ejecuciones para analizar las diferencias entre cada una de las ejecuciones y los modelos resultantes.

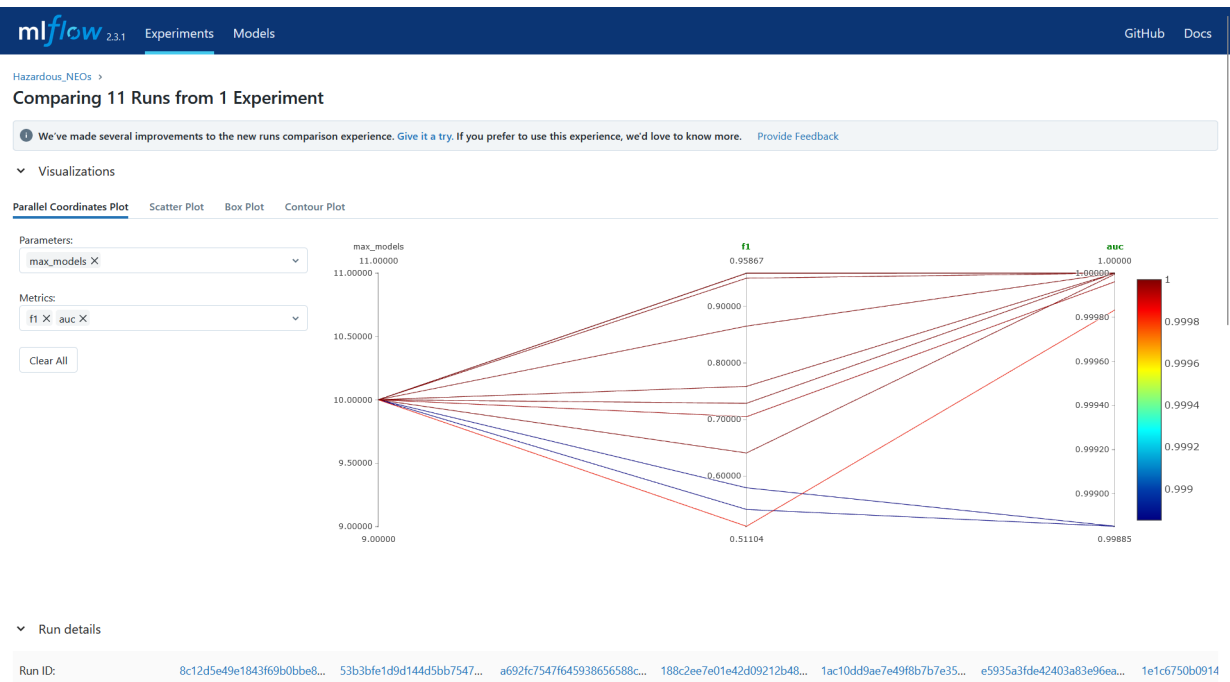
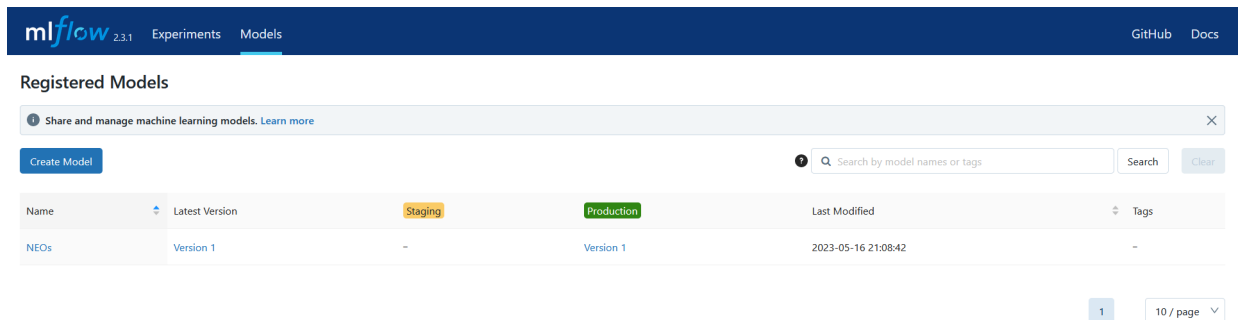


Figura A.3.: Ventana de comparación entre las distintas ejecuciones

## A.2 Uso de la herramienta

Todo esto permite tener un registro exhaustivo que se puede consultar desde la web y permite analizar y comprar los resultados de las distintas ejecuciones con total fiabilidad.

**Modelos** La pestaña de modelos muestra todos los modelos registrados, junto con su versión y su ciclo de vida. Si quieres acceder a los archivos asociados al modelo, debes consultar la ejecución que tiene asociada.



**Figura A.4.:** Pestaña de modelos

Para realizar el registro del entrenamiento de los modelos, deberemos incorporar pequeños fragmentos de código embebidos en nuestro propio código. Esto se realiza para indicar que partes son las que queremos almacenar y donde queremos que esto se realice.

El archivo adjunto contiene un ejemplo con el código necesario para:

1. Conectarte al servidor MLFlow.

```
os.environ['MLFLOW_TRACKING_URI'] = 'http://ip:puerto'
os.environ['MLFLOW_EXPERIMENT_NAME'] = 'pruebaSklearnConector'
# Nombre del experimento/proyecto
```

2. El dataset puede ser obtenido tanto de la base de datos, como de ficheros .csv. Es necesario guardar la consulta o el dataset en todos los runs. En el caso de usar un dataset de grandes dimensiones, es necesario comnatarlo vara llegar a una manera de guardar los datasets de la mejor manera posible.

- a) En el caso de tratarse de un dataset obtenido de un fichero .csv se ha de guardar el dataset completo.

```
mlflow.log_artifact("ruta/hasta/csv", "dataset")
```

- b) En el caso de haber obtenido el dataset mediante una query es necesario almacenar la query con la que se obtenido.

```
mlflow.log_text(variableConLaQuery, "query")
```

3. Hacer un registro de experimentos y las ejecuciones (runs) asociadas a dichos experimentos.

4. La información que se almacena es la siguiente (toda esta información se puede complementar con tags, descripciones y otros atributos):

- a) Modelo/Modelo registrado y signature (Es recomendable siempre generar la signature de manera manual.

```
signature=mlflow.models.infer_signature(data.drop("target", axis=1),
```

```
# Registrar un modelo
```

```
mlflow.flavor.log_model(variableConElModelo, "model",
```

```
registered_model_name=
```

```
signature=signature)
```

```
# Almacenar un modelo
```

```
mlflow.flavor.log_model(variableConElModelo,
```

```
"model", signature=sig
```

- b) Parámetros.

```
mlflow.log_param("nombreParametro", variableParametro)
```

- c) Métricas.

```
mlflow.log_metric("nombreMetrica", variableMetrica)
```

- d) Artefactos.

```
mlflow.log_artifact(rutaLocalAlArchivo, nombreArchivoEnDestino)
```

- e) Además es posible almacenar artefactos de distintas maneras: `log_text`, `log_dict` o `log_image`.

- f) La librería `mlflow_extend` permite el almacenamiento de más clases de artefactos.

#### A.2.1.3. Conector SMB

Para poder usar Mlflow es necesario tener conexión con el repositorio de artefactos localizado en el NAS, se han incluido dos archivos para la conexión, la

primera necesita permisos de administración y prepara lo necesario para la conexión, la segunda, se debe ejecutar cada vez que se quiera usar Mlflow después de encender el ordenador.

**IMPORTANTE: NO RELIZAR LA CONEXIÓN PUEDE CAUSAR PROBLEMAS A LA HORA DE RELIZAR EXPERIMENTOS.** El conector comprueba la conexión con el servidor de ficheros y se puede ha de añadir su lanzamiento a los archivos Python que se usen para entrenar.

El conector (conectorMlflow.bat) en la ruta C:\ para tener contar con una ruta absoluta para su ejecución desde los scripts de entrenamiento.

Es necesario modificar la primera línea de ambos conectores para especificar el usuario y contraseña en el NAS.

### A.2.1.4. Ejemplo completo

```
import h2o from h2o.automl
import H2OAutoML
h2o.init()
import pandas as pd
from sklearn.model_selection import train_test_split
import mlflow
from mlflow_extend.logging import log_confusion_matrix
from mlflow_extend.logging import log_confusion_matrix
import os os.environ['MLFLOW_TRACKING_URI']='http://localhost'
os.environ['MLFLOW_EXPERIMENT_NAME']=f"Hazardous_NEOs"

def getMediumMetrics(metric):
    numFalse=data[["Target"]].countmatches("False").sum()
    numTrue=data[["Target"]].countmatches("True").sum()
    return (numFalse*metric[0][0]+numTrue*metric[0][1])/(numTrue+numFalse)

data=h2o.import_file("../data/preparedData.csv")

train, test = data.split_frame(ratios=[0.8])
train[:, -1] = train[:, -1].asfactor() test[:, -1] = test[:, -1].asfactor()

max_models=10
aml=H2OAutoML(max_models=max_models)
aml.train(x=train.columns[0:-1], y=train.columns[-1],
          training_frame=train)
lb = aml.leaderboard lb.head(rows=lb.nrows)

for i in range(0, len(aml.leaderboard)-1):
    with mlflow.start_run():
        model=h2o.get_model(aml.leaderboard[i, "model_id"])
        dataPd=pd.read_csv("../data/preparedData.csv")
```

```
f1Score=model.F1()
mlflow.log_metric("FalseF1", f1Score[0][0])
mlflow.log_metric("TrueF1", f1Score[0][1])
mlflow.log_metric("f1", getMediumMetrics(f1Score))
mlflow.log_metric("auc", model.auc())
mlflow.log_text(str(model.confusion_matrix().to_list()),
                "ConfusionMatrix")
mlflow.log_text(str(model.params), "param")
mlflow.log_text(str(model), "model_info")
mlflow.log_text(str(model.model_performance()),
                "performance")
mlflow.log_param("max_models", max_models)

signature=mlflow.models.\
    infer_signature(dataPd.drop("Target",
                                axis=1), dataPd[["Target"]])
mlflow.h2o.log_model(model, "model", signature=signature)
```

## Bibliografía

- [Ana21] Anaconda. *Managing environments*. Anaconda, Inc., 2021.
- [BBvB<sup>+</sup>01] Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, and Dave Thomas. Principles behind the agile manifesto. <https://agilemanifesto.org/iso/en/principles.html>, 2001.
- [CD23] MLflow Developer Community and Databricks. MLflow 2.3.0 release. <https://mlflow.org/news/2023/04/18/2.3.0-release/index.html>, April 2023.
- [Clo21a] Google Cloud. Google cloud vertex ai. <https://cloud.google.com/vertex-ai>, 2021.
- [Clo21b] Google Cloud. Mlops: Continuous delivery and automation pipelines in machine learning. <https://cloud.google.com/architecture/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning>, 2021.
- [Com19] McKinsey & Company. Operationalizing machine learning in processes. <https://www.mckinsey.com/capabilities/operations/our-insights/operationalizing-machine-learning-in-processes>, 2019.
- [Com23] Kubernetes Community. Kubernetes. <https://kubernetes.io/>, 2023.
- [Cos22] Fernando Perez Costoya. Transparencias sistemas distribuidos (sistemas de ficheros distribuidos). *Universidad Politécnica de Madrid*, 2022.
- [CRI23] CRIDA. Sobre crida. <https://cida.es/webcida/index.php/sobre-crida>, 2023.
- [DAR22] DARPA. Ai next campaign. <https://www.darpa.mil/work-with-us/ai-next-campaign>, 2022.
- [Dat23] Databricks. Databricks - unified analytics. <https://www.databricks.com/>, 2023.
- [DE23] DB-Engines. Db-engines ranking. <https://db-engines.com/en/ranking/relational+dbms>, 2023.

- [DG04] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. In *Proceedings of the 6th Symposium on Operating System Design and Implementation*. USENIX Association, 2004.
- [Dic19] Ben Dickson. Why do 87 <https://venturebeat.com/ai/why-do-87-of-data-science-projects-never-make-it-into-production/>, 2019.
- [DJP<sup>+</sup>20] Prafulla Dhariwal, Heewoo Jun, Christine Payne, Jong Wook Kim, Alec Radford, and Ilya Sutskever. Jukebox: A generative model for music, 2020.
- [Doc22a] Docker. Compose file reference. <https://docs.docker.com/compose/compose-file/>, 2022.
- [Doc22b] Docker. Docker documentation: Container networking. <https://docs.docker.com/config/containers/container-networking/#published-ports>, 2022.
- [Doc22c] Docker. Use volumes. <https://docs.docker.com/storage/volumes/>, 2022.
- [Doc23] Docker. Bind mounts. <https://docs.docker.com/storage/bind-mounts/>, 2023.
- [dtD22a] MLflow development team and Databricks. MLflow model registry documentation. <https://mlflow.org/docs/latest/model-registry.html#registry>, 2022.
- [dtD22b] MLflow development team and Databricks. MLflow models documentation. <https://mlflow.org/docs/latest/models.html#models>, 2022.
- [dtD22c] MLflow development team and Databricks. MLflow: Open source platform for the complete machine learning life cycle. <https://github.com/mlflow/mlflow>, 2022.
- [dtD22d] MLflow development team and Databricks. MLflow projects documentation. <https://mlflow.org/docs/latest/projects.html#projects>, 2022.
- [dtD22e] MLflow development team and Databricks. MLflow tracking documentation. <https://mlflow.org/docs/latest/tracking.html#tracking>, 2022.
- [Fou22] Python Software Foundation. *Python 3 Documentation: Virtual Environments*. Python Software Foundation, 2022.
- [Gar19] L. Chris Garry. Apollo-11. <https://github.com/chrislgarry/Apollo-11>, 2019.
- [GC22] Google and Kubeflow Community. Kubeflow. <https://www.kubeflow.org/>, 2022.



- [GGL03] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system. In *ACM SIGOPS Operating Systems Review*, volume 37, pages 29–43. ACM, 2003.
- [GPAM<sup>+</sup>14] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.
- [H2O22] H2O.ai. H2o mlops. <https://h2o.ai/resources/product-brief/h2o-mlops/>, 2022.
- [Had21] Apache Hadoop. Docker official image packaging for apache hadoop. <https://hub.docker.com/r/apache/hadoop>, 2021.
- [Had23] Apache Hadoop. Apache Hadoop. <https://github.com/apache/hadoop>, 2023. Licencia: Apache License 2.0.
- [HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016, <https://proceedings.neurips.cc/paper/2015/file/86df7dcfd896fc2674f757a2463Paper.pdf>.
- [IBM23] IBM. Exploratory data analysis. <https://www.ibm.com/topics/exploratory-data-analysis>, 2023.
- [Int18] Intel. Multimodality: A new frontier in cognitive ai. <https://community.intel.com/t5/Blogs/Tech-Innovation/Artificial-Intelligence-AI/Multimodality-A-New-Frontier-in-Cognitive-AI/post/1407751>, 2018.
- [kag] Kaggle. <https://www.kaggle.com/>.
- [Lab22] Domino Data Lab. Domino data lab. <https://www.dominodatalab.com/>, 2022.
- [LLD<sup>+</sup>18] Jie Lu, Anjin Liu, Fan Dong, Feng Gu, Joao Gama, and Guangquan Zhang. Learning under concept drift: A review. *IEEE Transactions on Knowledge and Data Engineering*, pages 1–1, 2018, <https://doi.org/10.1109>
- [MB19] Mercedes-Benz. The prometheus project launched in 1986 - pioneering autonomous driving. <https://group-media.mercedes-benz.com/marsMediaSite/en/instance/ko/The-PROMETHEUS-project-launched-in-1986-Pioneering-autonomous-driving.xhtml?oid=13744534>, 2019.
- [mdcD22] mlflow’s developer community and Databricks. Mlflow. <https://mlflow.org/>, 2022.
- [Meh18] Shruti Mehta. NASA Asteroids Classification. <https://www.kaggle>.

- com/datasets/shrutimehta/nasa-asteroids-classification, 2018. License: CC0: Public Domain".
- [Mic22] Microsoft. Azure machine learning. <https://azure.microsoft.com/es-es/products/machine-learning#tabx66cb371a6af64f9ba871278b526e28ac>, 2022.
- [nas] NASA Near Earth Object Program. <http://neo.jpl.nasa.gov/>.
- [nas22] El impacto de DART cambio el movimiento de un asteroide en el espacio. <https://www.nasa.gov/press-release/el-impacto-de-dart-cambi-el-movimiento-de-un-asteroide-en-el-espacio>, 2022.
- [Ope23] OpenAI. Gpt-4 technical report, 2023.
- [Pra19a] Pravega. Docker hdfs. <https://github.com/pravega/docker-hdfs>, 2019. License: MIT.
- [Pra19b] Pravega. Pravega/hdfs. <https://hub.docker.com/r/pravega/hdfs>, 2019.
- [Rob20] David Robinson. How to put machine learning models into production. <https://stackoverflow.blog/2020/10/12/how-to-put-machine-learning-models-into-production/>, 2020.
- [RPG<sup>+</sup>21] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation, 2021.
- [Ser22a] Amazon Web Services. Amazon sagemaker. <https://aws.amazon.com/en/sagemaker/>, 2022.
- [Ser22b] Amazon Web Services. Why use amazon sagemaker projects? <https://docs.aws.amazon.com/sagemaker/latest/dg/sagemaker-projects-why.html>, 2022.
- [sta21] Docker compose postgres upgrade initdb error - directory '/var/lib/postgresql/data' not empty. <https://stackoverflow.com/questions/62697071/docker-compose-postgres-upgrade-initdb-error-directory-var-lib-postgresql-da>, 2021.
- [The21] The Apache Software Foundation. Apache Hadoop. <https://hadoop.apache.org/>, 2021.

Este documento esta firmado por



<b>Firmante</b>	CN=tfgm.fi.upm.es, OU=CCFI, O=ETS Ingenieros Informaticos - UPM, C=ES
<b>Fecha/Hora</b>	Wed May 31 20:16:27 CEST 2023
<b>Emisor del Certificado</b>	EMAILADDRESS=camanager@etsiinf.upm.es, CN=CA ETS Ingenieros Informaticos, O=ETS Ingenieros Informaticos - UPM, C=ES
<b>Numero de Serie</b>	561
<b>Metodo</b>	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)