UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

# Desarrollo de una herramienta MLOps de apoyo a proyectos de ciencia de datos

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Núñez Molina, Iván

Tutor/a: Monserrat Aranda, Carlos

CURSO ACADÉMICO: 2022/2023

# Resumen

MLOps es importante para el desarrollo de proyectos en ciencia de datos porque permite a los equipos de ciencia de datos trabajar de manera más eficiente, mejorar la calidad del modelo, aumentar la escalabilidad y reducir el riesgo de errores y fallas en la implementación del modelo en producción. Un primer problema con el que nos encontramos en MLOps es que realmente está formado por un universo de aplicaciones (git, DVC, MLEM, gitLab...) que necesitan estar coordinadas para que el proyecto se desarrolle en condiciones. Aunque existen herramientas que permiten trabajar en el universo MLOps de forma coordinada en la nube (https://studio.iterative.ai/), no existe una herramienta cómoda de ayuda a nivel local. El objetivo de este TFG es, pues, el desarrollo de una herramienta que, a nivel local, permita el desarrollo de proyectos de ciencias de datos de forma rápida, fiable y segura dentro del universo MLOps.

**Palabras clave:** bases de datos, CI/CD, control de versiones, MLOps

# Abstract

MLOps is important for the development of data science projects because it allows the developers to work more efficiently, improve the model's quality, increase scalability and reduce the risk of mistakes during the implementation in production. The first issue with MLOps is that it is made up of a myriad of different applications (git, DVC, MLEM, gitLab...) that need to coordinate so the project can function properly. Even though there are tools that offer coordinated cloud services for MLOps projects (https://studio.iterative.ai/), there isn't a useful version of this tool for local development. The objective of this end-of-degree project is to develop a program that, locally, allows for the development of data science projects in a fast, trustworthy and secure way, and making use of the possibilities that MLOps offers.

**Key words:** databases, CI/CD, version control, MLOps

# Índice general

# Índice de figuras



# Índice de tablas

# CAPÍTULO 1
# Introducción

El campo de la informática lleva años produciendo avances masivos para la sociedad, desde el ordenador personal en los años 70, hasta Internet en los 90, el comercio online, o las redes sociales que hoy en día dominan la navegación en línea.

En la actualidad, una de las áreas con una expansión más rápida es la de la ciencia de datos, también conocida popularmente como *Big Data*. El ámbito de la ciencia de datos concentra todas las tecnologías que utilizan cantidades masivas de cualquier tipo de datos para convertirlas en información: es decir, convierte los datos brutos en un recurso utilizable y explotable. Lo verdaderamente revolucionario de esta tecnología es la forma en la que se desarrolla el producto final: en lugar de basarse puramente en la programación de un trabajador, se diseña un sistema que "aprende" automáticamente de una base de datos dada, y extrae patrones para crear un modelo inteligente.

La aplicación más popular de esta tecnología recientemente ha sido ChatGPT: un modelo de inteligencia artificial capaz de conversar con un usuario humano, respondiendo preguntas de forma coherente y con una precisión alta. Sin embargo, a nivel empresarial, la ciencia de datos lleva años siendo una herramienta fundamental. Un buen uso de estas tecnologías puede permitir a una compañía mejorar su estrategia de negocio e incrementar sus beneficios de una manera inaccesible de otra forma, a través de la creación de perfiles de clientes u ofertas personalizadas, entre otras cosas.

Los grandes proyectos informáticos requieren de la colaboración de multitud de trabajadores y pasan frecuentemente por un proceso iterativo de mejora constante hasta que se obtiene el producto final, llegando incluso a seguir desarrollando nuevas versiones después del lanzamiento oficial. Para facilitar este proceso, se han creado a lo largo de los años varias herramientas de control de versiones que le facilitan el trabajo a los desarrolladores, entre las cuales se encuentra *git*, la más popular. Estas herramientas les permiten estandarizar el control de versiones de sus proyectos de forma estructurada, e incluso desplazarse entre ellas a voluntad o revisar independientemente las contribuciones de cada miembro del equipo.

Los proyectos de ciencia de datos tienen simultáneamente necesidades similares y diferentes. Por un lado, pasan también por este mismo proceso de diseño iterativo que requiere de un control estricto de modificaciones. Sin embargo, la naturaleza del desarrollo de estos proyectos requiere de bases de datos que suelen tener tamaños muy grandes; esto supone un problema para las herramientas de control más tradicionales como *git*, que dependen del tamaño reducido de los archivos de código típicos para funcionar correctamente. Con el objetivo de paliar este problema, la empresa estadounidense Iterative Inc ha desarrollado *Data Version Control* (DVC). DVC es un software de ordenador que actúa como una extensión de *git* para darle al desarrollador una forma simple de compatibilizar el uso de *git* con el desarrollo de proyectos de ciencia de datos.

DVC es una herramienta potente y completa para este propósito, pero como todas las herramientas potentes y completas, también requiere una serie de conocimientos previos que no siempre son fáciles de adquirir. Mi objetivo en este Trabajo de Fin de Grado es adaptar este software a un uso más simplificado, apropiado para alumnos de ciencia de datos y principiantes en general, a través de la creación de un programa que actúe como interfaz y guía simultáneamente.

## 1.1 Motivación

Mi interés en este proyecto surge puramente de una inclinación personal hacia proyectos relacionados con la ciencia de datos. Una vez terminada mi formación general en el ámbito de la informática, he descubierto que este área es la que más atractiva me resulta, pero desafortunadamente no hemos tenido la opción de desarrollarla tanto como me gustaría en la carrera de Ingeniería Informática.

Debido a esto, he querido aprovechar la oportunidad que se me ha presentado en este Trabajo de Fin de Grado para acercarme más a la ciencia de datos, y familiarizarme con algunas herramientas de la industria que son muy útiles a la hora de desarrollar proyectos. Espero que trabajar íntimamente, no sólo con estas herramientas sino sobre ellas, me permita aprender a utilizarlas con seguridad y a emplearlas en mis propios proyectos.

## 1.2 Objetivos

El objetivo de este Trabajo de Fin de Grado es el desarrollo de una plataforma de apoyo al desarrollo de proyectos de ciencia de datos, un área también conocida como "MLOps", con almacenamiento local. Esta plataforma tiene como usuario objetivo a individuos con un conocimiento limitado de los principios de la ciencia de datos (por ejemplo, alumnos): los criterios de selección de las características que incluiremos en dicha plataforma se centrarán por tanto en funciones simplificadas pero representativas del proceso real de desarrollo de estos proyectos.

## 1.3 Impacto esperado

Esta plataforma busca proporcionar a alumnos e interesados por la ciencia de datos una herramienta simple con la que iniciarse. Esto haría que su entrada en este área tan reciente y compleja fuera algo más suave, lo que incrementa las posibilidades de que aprendan y mantengan el interés.

## 1.4 Estructura de la memoria

- **Introducción** Se presenta el concepto del TFG, junto a su motivación e informaciones adicionales relevantes.
- **Estado del arte** Se realiza un análisis de las áreas relevantes del entorno del TFG para obtener una imagen fiel y actual.
- **Análisis del problema** Se formula el problema a solucionar por el proyecto para evitar errores de planteamiento.

- **Diseño de la solución** Se prepara la solución que busca dar respuesta al problema planteado, teniendo en cuenta tanto objetivos como limitaciones.
- **Desarrollo de la solución propuesta** Se lleva a cabo la solución preparada anteriormente, documentando el proceso en detalle.
- **Pruebas** Se realizan una serie de pruebas para comprobar que el resultado del proyecto actúa de forma correcta frente al problema a solucionar.
- **Manual de usuario** Se describe en detalle el funcionamiento del programa de forma didáctica y utilitaria.
- **Conclusiones** Se resume el desarrollo del proyecto, comprobando que se han cumplido los objetivos planteados y realizando un sumario del proceso, y se plantea el futuro del proyecto.
- **Anexo: Objetivos de Desarrollo Sostenible** Se relaciona el proyecto desarrollado a una serie de Objetivos de Desarrollo Sostenible relevantes.

## 1.5 Convenciones

El área de ciencia de datos y *Machine Learning* está plagada de abreviaciones y convenciones. Utilizaremos algunas en este trabajo:

- DVC (*Data Version Control*): software de control de versiones de código abierto desarrollado por la empresa norteamericana Iterative, en el que se basa este proyecto.
- MLOps (*Machine Learning Operations*): conjunto de principios de programación y gestión de proyectos derivado de "DevOps", que, como su antecesor, se basa en coordinar el proceso completo de Desarrollo y Operaciones para agilizar el proceso y obtener mejores resultados.

# CAPÍTULO 2
# Estado del arte

Antes de comenzar la preparación de nuestro proyecto, vamos a realizar un análisis del sector en el que recabaremos información sobre plataformas de MLOps similares.

## 2.1 Crítica al estado del arte

El incremento de escala y requisitos de los proyectos informáticos modernos ha puesto de manifiesto la necesidad de establecer protocolos y sistemas para el propio desarrollo de estos proyectos, que faciliten el trabajo de los trabajadores y les permita centrarse en el código en vez de en la gestión del proyecto. Como respuesta a esta necesidad han surgido varias iniciativas que les dan herramientas para esto, pero la más popular es sin duda git.

Git es un sistema distribuido de control de versiones que permite a sus usuarios llevar un registro exhaustivo de su código (o archivos en general), y gestionar un desarrollo paralelo entre varios programadores con el uso de ramas.

- Registro de versiones: git proporciona un historial completo de las versiones guardadas por el usuario con un identificador único, el nombre del autor, la fecha y la rama correspondientes. Este registro le permite desplazarse en el tiempo y consultar cualquier archivo en un estado anterior, o incluso continuar el desarrollo desde un punto anterior. Este registro además está construido utilizando a algoritmos que se basan en diferencias en los archivos en lugar de guardar directamente copias del archivo, con lo que su tamaño es muy inferior al que tendría si tuvieran que guardarse copias del proyecto en cada versión.

- Ramas: aunque por defecto se crea una sola rama, el usuario puede crear manualmente otras. Una rama es una separación del historial de versiones, resultando en un "nuevo historial", al que se pueden guardar nuevas versiones de forma independiente sin afectar a la rama principal (llamada por convención "master"). Las ramas suelen utilizarse para trabajar en grandes añadidos al proyecto principal que requieren de bastante tiempo de desarrollo, de forma que la rama master contenga un proyecto funcional mientras se avanza en la característica añadida.

- Fusión de ramas: una vez es termina el desarrollo de estas características, se pueden unir de forma muy sencilla a la rama principal para integrarlas en el proyecto. Esta integración está gestionada de forma interna por git, para que se realice sin errores de sobreescritura (por ejemplo, la característica modifica un archivo que también fue modificado en el proyecto principal después de la separación de ambas ramas, un caso en el cual estas modificaciones serían deshechas por la integración de la

rama independiente). Git se encarga de que en cada archivo se modifiquen sólo las partes relevantes, dejando el resto sin tocar.

Pese a ser una tecnología muy popular, git sufre ciertas limitaciones que le impiden lograr esa misma popularidad en el campo de la ciencia de datos. En proyectos de este tipo, cada versión suele incluir archivos de mucho tamaño, como los datos de entrenamiento de un modelo de inteligencia artificial. Git tiene una gestión inteligente de sus versiones para reducir su tamaño, pero aún así un archivo de datos puede pesar de varios GB a varios TB de memoria, lo cual hace git inviable. Sin embargo, su utilidad y su popularidad es tal, que otros agentes de la industria informática han creado sistemas que actúan como capa sobre git, y le permiten suplir estas carencias. La empresa Iterative ha desarrollado DVC (Data Version Control), un software que se encarga de gestionar estos archivos de gran tamaño y compatibilizar esta gestión con el sistema de control de versiones de git. DVC incluye las siguientes funcionalidades:

- Guardado de archivos grandes: para gestionar el guardado y versionado de archivos de gran tamaño, DVC simula el proceso de git (`git add [archivo]`) con el comando `dvc add [archivo]`. Este comando crea un `archivo.dvc` que el usuario debe añadir manualmente a git. Este `archivo.dvc`, que es básicamente un puntero hacia el archivo original y es por tanto mucho más pequeño, será el registrado por el sistema de versiones de git. Cuando el usuario cambie entre versiones, sólo tendrá que ejecutar un comando adicional para sincronizar los archivos de gran tamaño con la versión del proyecto a la que haya cambiado.

- Implementación de pipelines: para dar soporte adicional a los proyectos de ciencia de datos, DVC proporciona una implementación sencilla para el concepto de pipelines. Se le llama "pipeline" al proceso que llevan a cabo los trabajadores de ejecutar secuencialmente una serie de scripts que termina con la creación y evaluación de un modelo de IA. DVC permite al usuario diseñar este proceso en un archivo único, y después ejecutarlo con un solo comando `dvc repro`. Además, el sistema de versiones de DVC también registra de forma inteligente los cambios en los archivos de la pipeline, lo que hace que el comando ejecute sólo los scripts cuyas dependencias han sufrido cambios, ahorrando tiempo de cómputo y agilizando el proceso.

- Soporte de métricas: aunque las métricas de evaluación de un modelo de IA pueden variar mucho de proyecto en proyecto, DVC ofrece un sistema simple pero eficaz que permite al usuario marcar en la pipeline ciertos valores como métricas, que luego quedarán registrados en un archivo aparte (normalmente guardado en git por su tamaño reducido). Estas métricas pueden después visualizarse con `dvc metrics show`, o `dvc metrics diff` si quieres ver una comparación con las métricas del último experimento.

- Generación de gráficos: DVC también permite al usuario crear gráficos configurando esta opción en el archivo de la pipeline. Estos gráficos pueden ser de varios tipos, como una matriz de confusión, y se puede tanto acceder a los datos base para representarlos en otras plataformas, exportarlos en un formato de imagen para utilizarlos directamente, o utilizar el formato HTML que ofrece DVC por defecto y que se puede visualizar con un navegador web estándar.

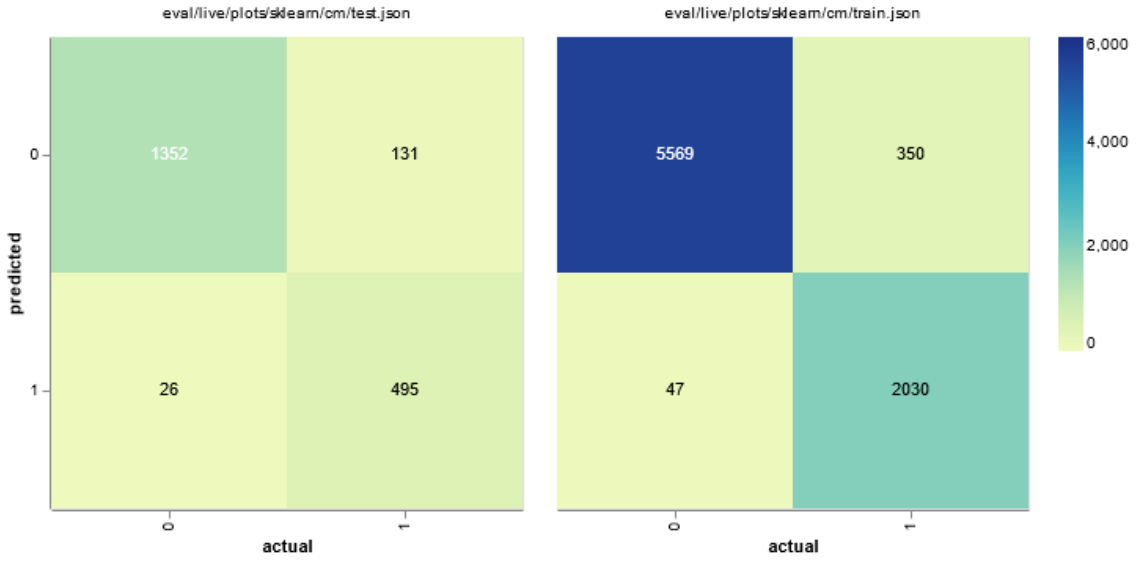


**Figura 2.1:** Ejemplo de matriz de confusión de un proyecto

Pese a ser una industria relativamente nueva, su crecimiento ha propiciado la comercialización de muchas herramientas de MLOps que buscan ocupar el mismo nicho de mercado. Estas herramientas comparten muchas características, generalmente consideradas esenciales para competir en el sector, y disponen de otras más únicas que las desmarcan de otras. Vamos a realizar una comparativa entre 4 opciones populares, y ver qué funciones comparten al igual que cuáles no.

### Iterative Studio[1]

La solución propuesta por la empresa Iterative, que utiliza su propio sistema de código abierto DVC para gestionar los datos de los modelos de Machine Learning. Esto incluye las siguientes funcionalidades:

- Pipelines: se formaliza el proceso de iteración para entrenar el modelo en un documento. Esto agiliza el proceso iterativo para los desarrolladores, y además ahorra tiempo de computación al gestionar el proceso de manera inteligente (evitando reprocesar pasos que no han tenido cambios).

- Registro de modelos: se incorpora un gestor de proyectos en el que manejarlos de forma rápida e intuitiva, para permitir el desarrollo paralelo de modelos diferentes.

- Control de versiones: se lleva un registro minucioso de las modificaciones de cada proyecto utilizando la tecnología de `git`, lo que permite volver a versiones anteriores para consultar datos o resultados, o para continuar el desarrollo desde un punto previo.

- Visualización de métricas: proporciona una interfaz adaptada que muestra todo tipo de métricas de resultados sobre el modelo, ahorrándole al desarrollador el tiempo de calcular los diferentes valores en cada iteración del entrenamiento.

[1] https://studio.iterative.ai/

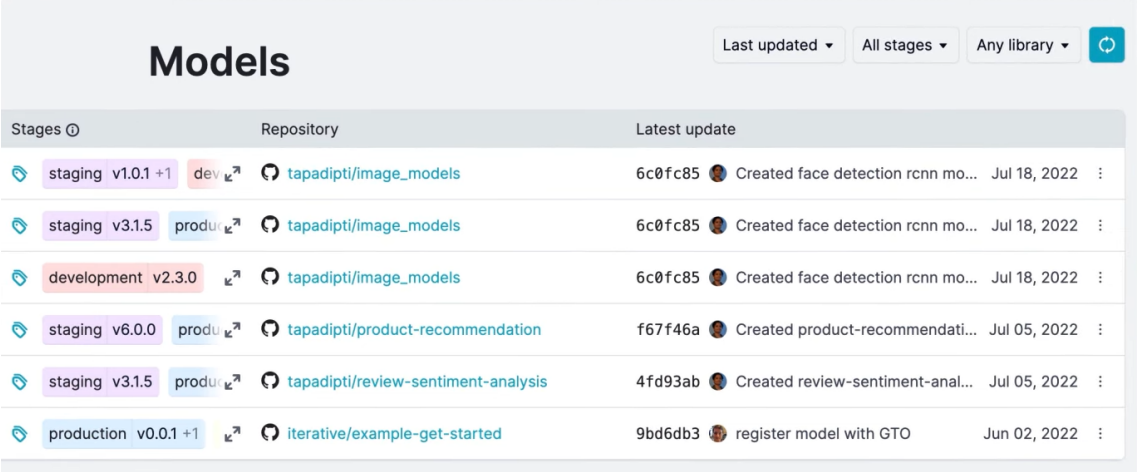


**Figura 2.2:** Panel de registro de modelos de Iterative Studio (Iterative, 2023)

### Comet ML[2]

Desarrollado por Comet con código propietario. Ofrece las mismas funciones de pipelines/registro de modelos/control de versiones/métricas que Iterative Studio, pero tiene como característica adicional la posibilidad de recopilar métricas de desempeño después del despliegue del modelo. Esto puede ayudar a asegurarte de que la calidad del modelo no se degrada con el tiempo, ya sea porque se entrena continuamente o porque las circunstancias a las que se aplica el modelo han cambiado.

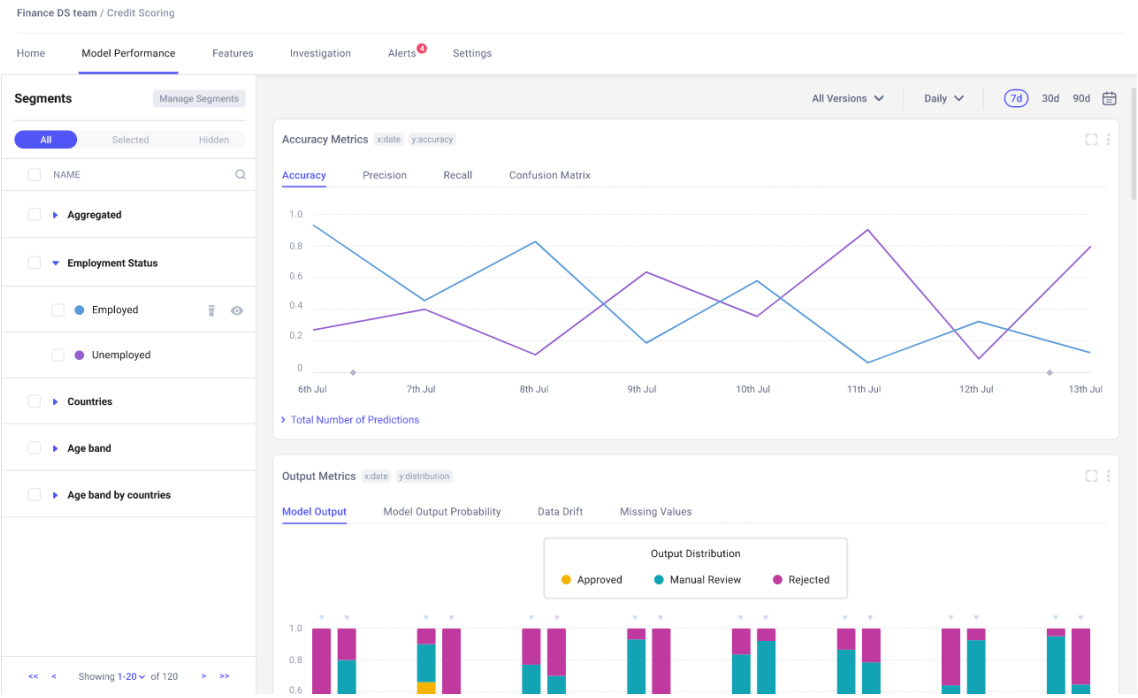


**Figura 2.3:** Panel de evaluación en directo de Comet ML (Comet, 2023)

[2] https://www.comet.com/site/
[3] https://wandb.ai/site

### Weights & Biases[3]

En lugar de métricas en directo, Weights & Biases ofrece la posibilidad de generar reportes después de la producción del modelo. Este sistema permiten a los desarrolladores crear un reporte interactivo que traduce los valores de las métricas a un lenguaje visual, mucho más fácil de interpretar para los propios desarrolladores, pero sobre todo para clientes y otros usuarios.


**Figura 2.4:** Ejemplo de reporte interactivo de Weights & Biases (Weights & Biases, 2023)

### Dataiku[4]

Combinando las funcionalidades de Comet ML y Weights & Biases, Dataiku es una de las plataformas líder en el sector, ya que dispone tanto de reportes interactivos como de métricas en directo. Dispone además de multitud de características que suavizan la experiencia de usuario y permiten a los desarrolladores personalizar la plataforma tanto para ellos como para los clientes. Incluye incluso soporte para "plugins", desarrollados tanto por la empresa como por empresas asociadas, que aportan nuevas funcionalidades al gusto de los desarrolladores.

Dataiku ha demostrado su valía en repetidas ocasiones al convertirse en la plataforma elegida por empresas muy importantes de talla internacional:

- Standard Chartered [1]: utilizan Dataiku para explotar al máximo las inmensas cantidades de datos financieros que poseen, dándoles no sólo los medios técnicos para sobreponerse a las limitaciones del software tradicional de hojas de cálculo que utilizaban anteriormente, sino también la estructura para organizar su nueva capacidad productiva de forma eficaz y eficiente.
- Mercedes-Benz [2]: Dataiku permite al departamento de Data Analytics convertir el paquete de ciencia de datos (limpieza de datos, pipelines, etc) en un plugin y compartirlo con otros departamentos de Mercedes-Benz para utilizarlos de forma más sencilla.
- Royal Bank of Canada [3]: el RBC utiliza Dataiku para hacer comprobaciones de datos financieros en auditorías y encontrar valores atípicos, ahorrándoles más de un 20 % del tiempo que dedicaban originalmente a estas tareas.

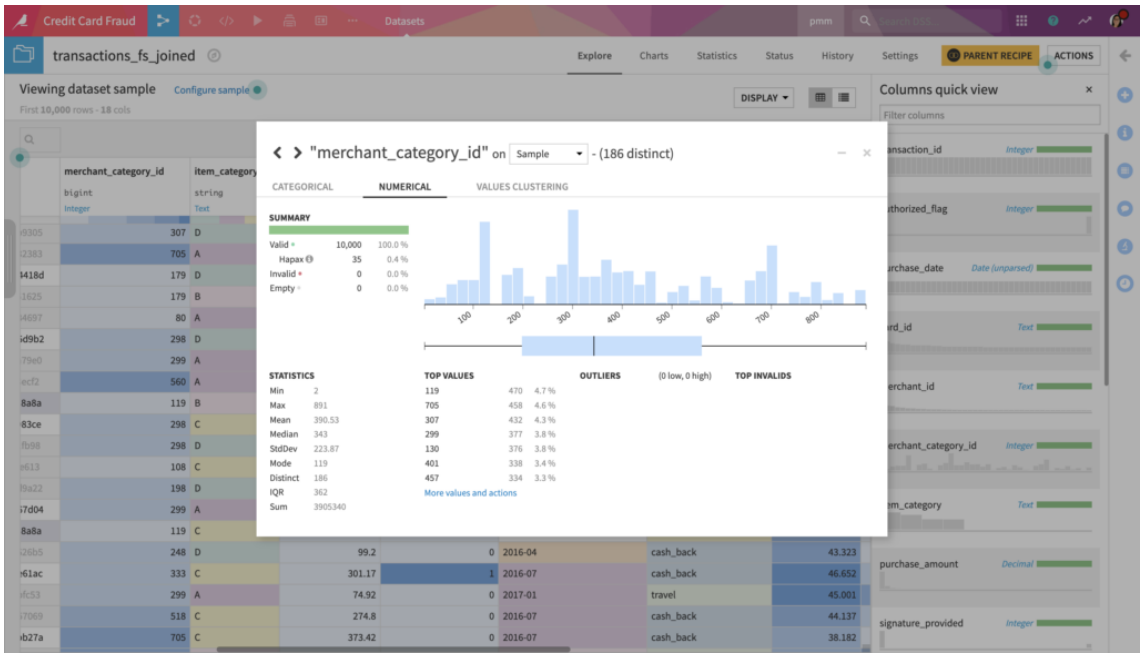


**Figura 2.5:** Panel de métricas de Dataiku (Dataiku, 2023)

[4] https://www.dataiku.com/

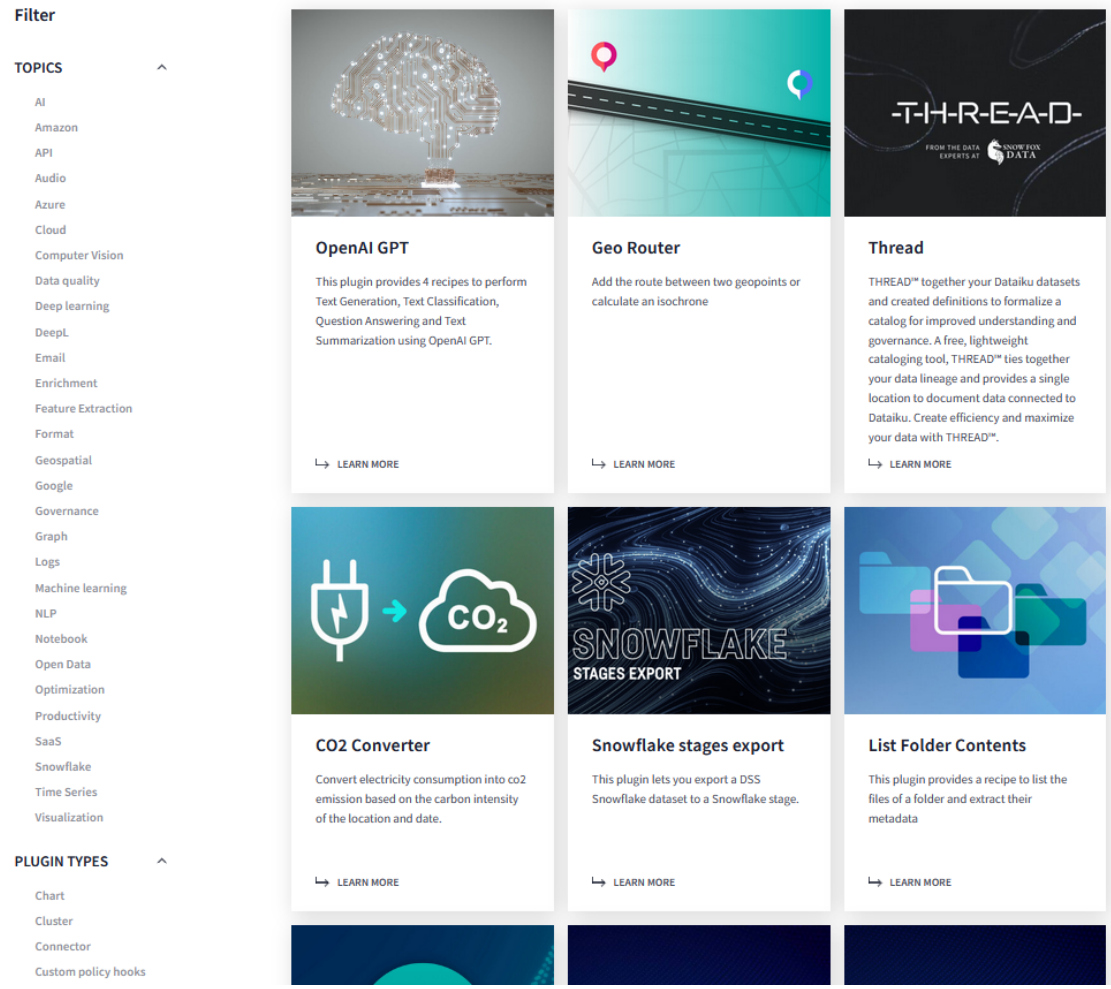


**Figura 2.6:** Tienda de plugins de Dataiku (Dataiku, 2023)

Dataiku es además la herramienta más popular con diferencia. Según una búsqueda orientativa en Google Trends (ver Anexo B.1), el interés por Dataiku es regularmente muy superior al de las otras 3 plataformas, con Weights & Biases muy por detrás, y finalmente Iterative AI y Comet ML a la cola. Esto se debe a varios factores, pero el más importante es la antigüedad de Dataiku: fue fundada en 2013[5], mientras que Weigths & Biases se lanzó en 2017[6] (al igual que Comet ML[7]), e Iterative Studio en 2022[8].

[5] https://www.crunchbase.com/organization/dataiku
[6] https://www.crunchbase.com/organization/weights-biases
[7] https://www.comet.com/site/about-us/
[8] https://www.businesswire.com/news/home/20220726005244/en/Iterative-Introduces-First-Git-based-Machine-Learning-Model-Registry

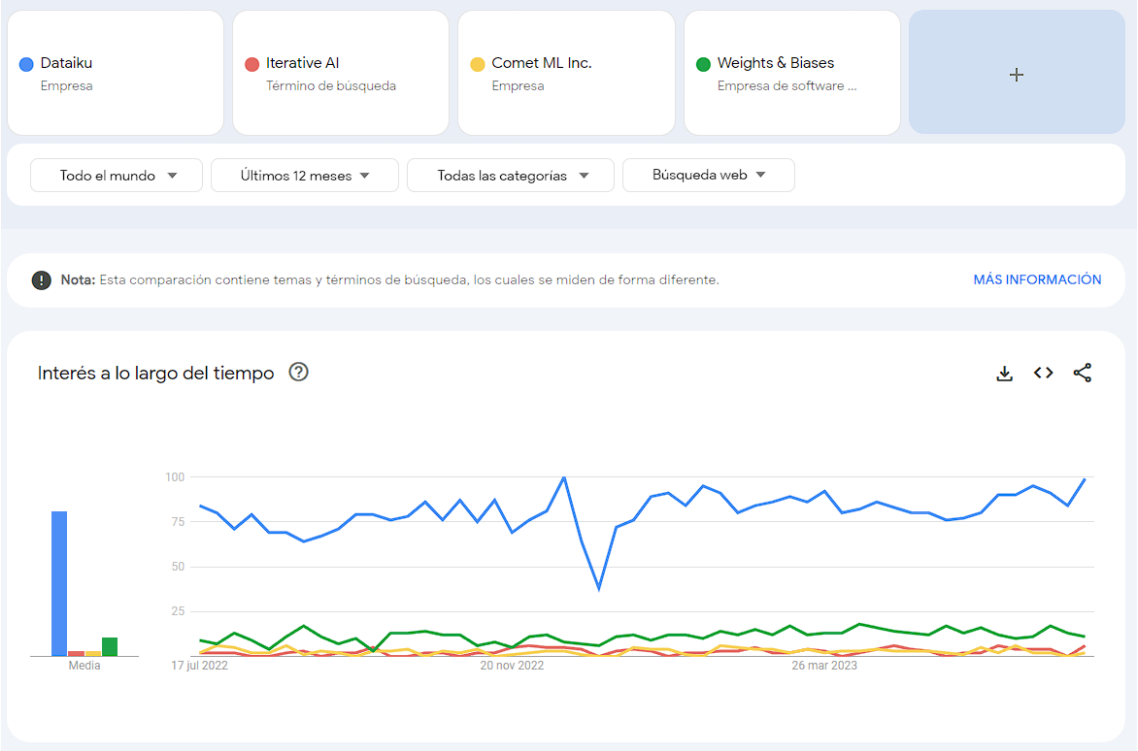


**Figura 2.7:** Popularidad en búsquedas online de las 4 herramientas (Google Trends, 2023)

Estas opciones son potentes y ofrecen una gran variedad de servicios de MLOps, incluyendo funcionalidades tanto básicas para el desarrollo de proyectos de ciencia de datos como adicionales. Sin embargo, todas se centran en el uso empresarial y dependen de almacenamiento en la nube, generalmente utilizado para proyectos de mayor escala.

## 2.2 Propuesta

En la sección anterior hemos visto que todas estas herramientas cubren un nicho de mercado muy amplio, ofreciendo una gran variedad de servicios y utilidades a los desarrolladores. Sin embargo, el enfoque empresarial que tienen las lleva a utilizar el almacenamiento en la nube como una parte integral de sus sistemas.

El objetivo de este TFG será desarrollar una plataforma que ofrezca los servicios básicos de estas plataformas, pero utilizando la máquina del usuario como almacenamiento local para los archivos de datos de su proyecto de ciencia de datos. Buscaremos además simplificar estos servicios de forma que estén adaptados a usuarios noveles, minimizando en la medida de lo posible la pérdida de funcionalidad por esta simplificación. Para llevar a cabo este proyecto, primero realizaremos un análisis de necesidades para determinar estos servicios básicos, apoyándonos en lo que ofrecen estas plataformas, seguido de un esfuerzo por diseñar una interfaz intuitiva y explicativa que permita a los usuarios utilizarla con conocimientos mínimos de la materia.

# CAPÍTULO 3
# Análisis del problema

En esta sección, nos centraremos en preparar el desarrollo de nuestra plataforma: declararemos inequívocamente los requisitos que debe cumplir, las funcionalidades que debe tener para poder cumplirlos, y estableceremos un plan de desarrollo para introducir estas funcionalidades.

## 3.1 Identificación y análisis del problema

Nuestro objetivo ha sido desde el principio crear una plataforma de servicios MLOps que proporcione una alternativa con almacenamiento local a las ya existentes. Comenzaremos definiendo nuestra problemática.

#### ¿Qué es MLOps?

MLOps, o "Machine Learning (and) Operations", es una serie de prácticas derivadas de DevOps, que a su vez significa "Development (and) Operations". Es una filosofía de producción que dicta que las diferentes etapas en el desarrollo de un producto informático deben estar íntimamente relacionadas y comunicar continuamente, con el fin de maximizar la calidad y el valor del producto creado. Estas etapas se conocen generalmente como "Development" y "Operations", refiriéndose respectivamente al desarrollo técnico y programación del producto, y al mantenimiento y comunicación con clientes para recibir feedback. Para incrementar la comunicación entre estas etapas, los impulsores del DevOps se centran en derribar las barreras que separan tradicionalmente estos dos sectores de la empresa, y reducen los ciclos de desarrollo para hacer un seguimiento más continuo del programa. De esta forma, logran cumplir de forma más precisa los requerimientos del cliente, y son capaces de detectar problemas antes. Entre otras cosas, es la base de la metodología ágil de desarrollo.

Aplicando estos principios a la ciencia de datos, MLOps busca optimizar el largo proceso de desarrollo de modelos de Machine Learning simplificando el paso de etapa a etapa. Las plataformas mencionadas en el apartado 2.1 ya siguen estos principios: el uso de pipelines es su manifestación más evidente, ya que están diseñadas específicamente para pasar de una etapa a otra de forma ininterrumpida. Como su nombre indica, se diseña una "tubería" que guía los datos a través del proceso de entrenamiento, sin que el desarrollador tenga que guiarlos en cada iteración.

Otro aspecto importante que tiene que ver con la organización del proceso de desarrollo, aunque se comparte con cualquier industria en la que se escriban documentos (es decir, todas) es el control de versiones: tener varias versiones del mismo documento, cada

una con una serie de cambios y un "v5_FINAL" añadido al final del nombre puede causar problemas de organización e incluso pérdida de información. El MLOps busca solucionar esto dándole un soporte técnico al versionado de documentos, aunque en este caso, los documentos son tanto los datos utilizados (que pueden ser modificados de varias formas diferentes para corregir errores), como el código de entrenamiento, o el propio modelo final.

Como último añadido para facilitar los proyectos de ciencia de datos, MLOps busca reducir el ciclo de desarrollo para incrementar la frecuencia de supervisión, con el fin de seguir más de cerca el proceso y asegurarse de que cumple con los requisitos del cliente. Para esto, se incluye frecuentemente una interfaz dedicada al estudio de las métricas de éxito del proyecto. Esto completa el ciclo de desarrollo del modelo, que comienza en los datos depurados y corregidos, pasando por las pipelines de entrenamiento, y finalizando en la evaluación de los resultados del modelo.

**¿Cómo creamos una plataforma de MLOps?**

Desarrollar de 0 una plataforma de MLOps es una tarea difícil, que estaría fuera del alcance de un Trabajo de Fin de Grado. Sin embargo, no es lo que vamos a hacer: utilizaremos DVC (*Data Version Control*), una tecnología de código abierto que da soporte en línea de comandos para llevar a cabo muchas de las tareas propias de proyectos de ciencia de datos que buscamos para nuestro programa. DVC está desarrollada por la empresa norteamericana Iterative, que también la utiliza para su propia plataforma, Iterative Studio.

## 3.2 Solución propuesta

Nuestro programa se centrará en ofrecer las funcionalidades básicas necesarias para nuestros usuarios, con el objetivo de facilitar el trabajo del desarrollador, especialmente del principiante. Estas funcionalidades son las siguientes:

### 3.2.1. Pipelines

Pipelines personalizables que permiten al desarrollador crear un "camino" por el que avanza cada paso del entrenamiento del modelo. Los componentes necesarios para crear estas pipelines estarán indicados por el propio programa. Generalmente, las pipelines se definen en un archivo específico que actúa como repositorio de referencias a las funciones que se ejecutan; el usuario ejecuta la pipeline previamente diseñada de acuerdo al proceso de entrenamiento (funciones de entrenamiento, parámetros, datos seleccionados...) y esta ejecuta los pasos en el orden determinado.

Las pipelines de Machine Learning suelen estar compuestas de varias etapas genéricas, cuyas especificaciones se adaptan al proyecto:

- Preparación: los datos se revisan para evitar errores, valores incorrectos o ausentes, o problemas de validación.
- Entrenamiento: se toma como input el conjunto de datos producido por la etapa de preparación, y se ejecuta el entrenamiento para crear el modelo de Machine Learning.
- Evaluación: se toma como input el modelo producido por la etapa de entrenamiento, y se evalúa su rendimiento comparándolo con un conjunto de datos de validación.

### 3.2.2. Control de versiones

Control de versiones para los datos y los modelos creados, utilizando el sistema de control de `git` a través del software de DVC. Esta característica es especialmente importante en el proceso de entrenamiento de un modelo: se necesita realizar un seguimiento preciso de los experimentos, ya que es muy fácil perderse y no poder explicar las ganancias/pérdidas en precisión, de forma que el modelo se vuelve irreproducible. Con un control de versiones integrado en el programa, esto se vuelve mucho más fácil y permite al desarrollador desplazar sus esfuerzos de tareas de gestión al propio desarrollo del modelo.

La tecnología de `git` permite al usuario llevar un registro fiable de cambios en el código de su proyecto. Para permitir esto, `git` permite a los desarrolladores trabajar en local y después publicar sus avances para el resto de programadores. Esto lo hace mediante un sistema de tres etapas: en primer lugar, el desarrollador realiza su trabajo de forma local, en un directorio en el que se ha inicializado una instancia de `git`. En segundo lugar, hace un "commit", en el que guarda su trabajo realizado en el registro. Y finalmente, hace un "push", en el que ese trabajo se sube al repositorio compartido y da acceso al resto de desarrolladores.

Hay además otras funcionalidades de `git` utilizadas principalmente en grandes proyectos: las ramas son duplicados del código que se separan en un momento preciso en el tiempo, y después se desarrollan de forma independiente. Esto te permite llevar dos versiones del proyecto que se desarrollan simultáneamente, para en el futuro volver a juntarlas o, por ejemplo, compararlas.

### 3.2.3. Visualización de métricas y parámetros

Pantalla de visualización de métricas y parámetros, que permite al desarrollador evaluar el rendimiento del modelo. Estas métricas suelen ser comunes a la mayoría de modelos de Inteligencia Artificial (*accuracy*, *precision*, etc.), pero según el objetivo del modelo se puede dar más importancia a unas que a otras.

El cálculo de las propias métricas es responsabilidad del usuario, el programa se encargará de mostrárselas de forma que pueda compararlas fácilmente entre sí y vea también qué parámetros son responsables de estos resultados. Esto le ahorrará el trabajo de extraer esta información manualmente, ya que aunque DVC ofrece una serie de funciones específicas para esto, las métricas en sí mismas siguen estando en un archivo que se sobreescribe en cada ejecución y cambia en cada versión guardada.

## 3.3 Plan de trabajo

El desarrollo del programa se llevará a cabo durante 5 semanas, que incluirán únicamente tiempo de programación y pruebas de funcionamiento.

| Semana | Trabajo a realizar |
|---|---|
| Semana 1 | Estructura general<br>Visualización de commits |
| Semana 2 | Métricas y parámetros<br>Gestión de pipelines |
| Semana 3 | Ejecución de pipelines<br>Creación de commits |
| Semana 4 | Creación de commits<br>Comprobación de casos límite |
| Semana 5 | Pruebas<br>Corrección de errores |

**Tabla 3.1:** Plan de trabajo de 5 semanas

La primera semana estará dedicada integralmente a diseñar la estructura del programa y la ventana principal, que incluirá de un vistazo una lista de los commits del proyecto y las métricas y parámetros de cada commit, si son aplicables. Esto permite al usuario tener toda la información relevante para comparar experimentos en un sólo vistazo.

En la segunda semana, después de añadir las métricas y parámetros a esta ventana, trabajaremos en la gestión de pipelines: esto incluye tanto visualizarlas de forma clara, como editarlas o ampliarlas, así como tener en cuenta una serie de potenciales errores para evitar problemas de ejecución del programa.

La tercera semana estará dedicada mayoritariamente a la ejecución de estas pipelines, particularmente a introducir una serie de opciones de conveniencia para el usuario; la ejecución en sí misma es simple, pero hay multitud de potenciales errores que tratar y también debemos comprobar los datos provistos por el usuario en la fase anterior.

El final de la semana y parte de la cuarta se emplearán en la creación de nuevos commits, para que el usuario pueda guardar los experimentos (o avances en el desarrollo). Esta etapa incluye no sólo el acto de guardar el commit en sí mismo, sino también gestionar la selección los archivos añadidos al commit, tanto de git como de DVC, y por último asegurarnos la integridad de los datos a la hora de manipular las versiones de los datos con los que trabaja el usuario. Procuraremos asegurarnos también de que los casos de uso principales están cubiertos, a la vez que los posibles errores en casos más extremos tienen salvaguardas apropiadas en el programa.

La quinta semana estará dedicada exclusivamente a realizar comprobaciones con un repositorio de prueba en el que introduciremos modificaciones, nuevos archivos, realizaremos commits, ejecutaremos pipelines, etc. Y, por supuesto, corregiremos los errores que se revelen en estas pruebas.

# CAPÍTULO 4
# Diseño de la solución

En este capítulo, veremos cuál es la estructura del sistema que va a dar soporte a nuestro programa y el razonamiento que se ha seguido para diseñarla de esta forma.

## 4.1 Arquitectura del sistema

La arquitectura de nuestro sistema viene dada en buena parte por el software que vamos a utilizar para el desarrollo. DVC es una capa de utilidad que se encarga de gestionar `git` para adaptarlo para un uso de ciencia de datos. Por tanto, lo que vamos a hacer es proporcionar una nueva capa interfaz, que a su vez facilite el uso de DVC.

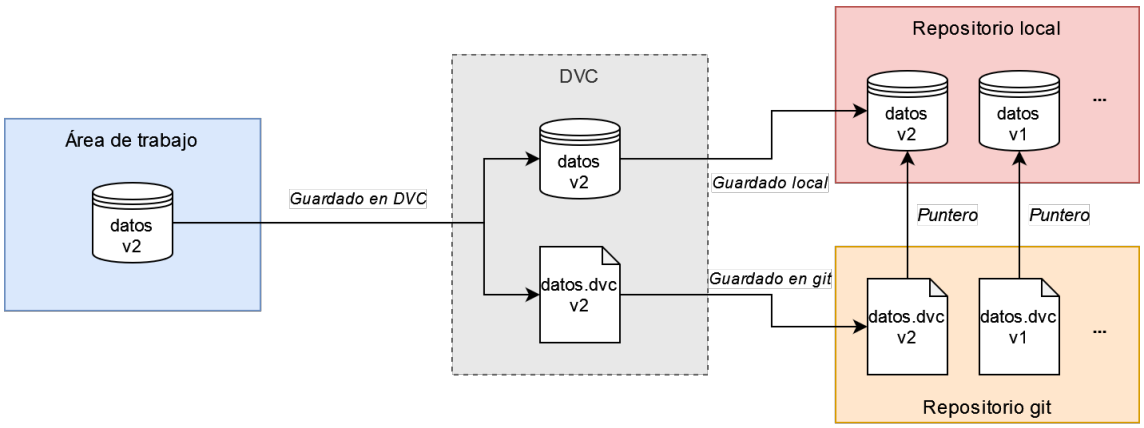


**Figura 4.1:** Arquitectura de un sistema DVC

Cuando el desarrollador está modificando los datos (o cualquier archivo, modelo, etc) y guarda su trabajo a través de DVC, lo que hace este software es crear un archivo `.dvc` que guarda los metadatos del archivo original. Este archivo `.dvc` es el que se guarda en git, mientras que el archivo original se guarda en el repositorio local. Esto hace que git sólo tenga que tratar con un archivo pequeño, mientras que los datos están guardados en otro lugar.

Cuando el desarrollador quiere recuperar una instancia anterior de una serie de datos (o de nuevo, de cualquier archivo), puede hacerlo con dos comandos ejecutados de manera consecutiva:

```
git checkout [branch] [datos].dvc
dvc checkout
```

Esto recuperará el archivo `datos.dvc` de git, y a continuación recuperará los datos asociados a ese archivo `.dvc`. Nuestro trabajo consiste en crear una interfaz que facilite este proceso (junto a varias funcionalidades más de DVC), según el siguiente esquema, en el que el usuario busca recuperar la tercera versión de una serie de datos:

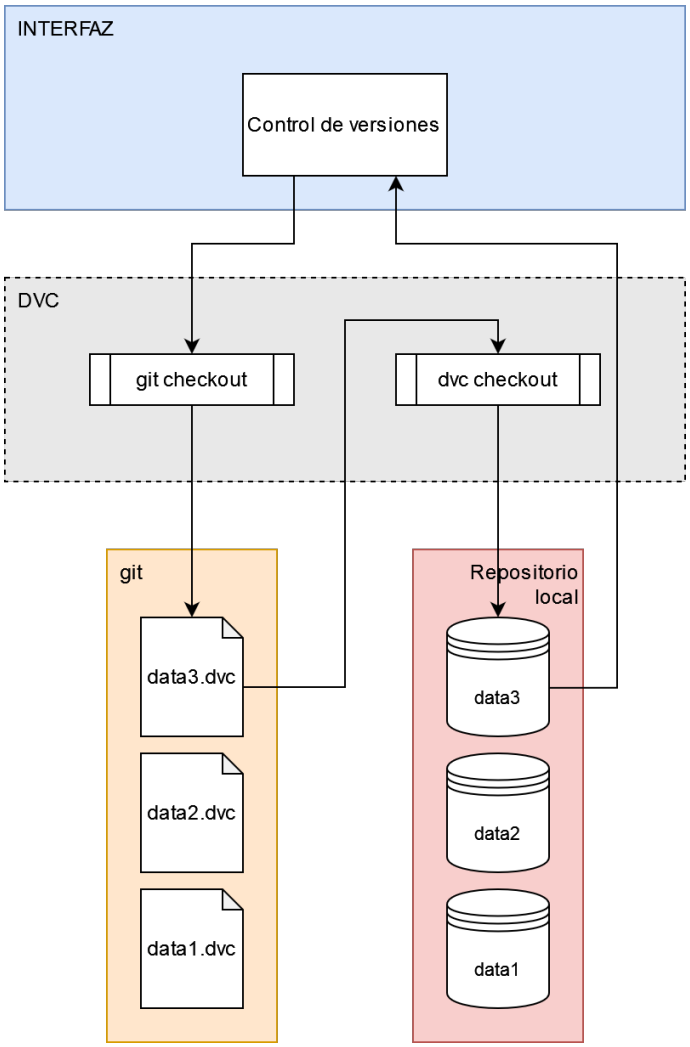


**Figura 4.2:** Diagrama del control de versiones utilizando DVC

Además, daremos también soporte a las pipelines como se describen en el apartado 3.2.1. Esto significa que el usuario tendrá una representación visual de la pipeline contenida en el archivo `dvc.yaml`, incluyendo los subapartados de cada etapa y sus contenidos. Esta funcionalidad será accesible a través de una nueva ventana dedicada a esto, en la que el usuario podrá tanto visualizar la pipeline, como modificarla y ejecutarla.

## 4.2 Diseño detallado

La base del programa a diseñar es simple: debe ofrecer una interfaz amigable al desarrollador principiante, en la que introduzca los datos que pretende utilizar para el entrenamiento de su modelo, configurar la pipeline de entrenamiento, y lanzarlo para después poder evaluar su rendimiento con una serie de métricas introducidas. Después, podrá realizar sucesivos entrenamientos y mantener las versiones anteriores de sus datos/modelos/métricas, así como volver a consultarlos.

### 4.2.1. Control de versiones

El control de versiones se lleva a cabo utilizando principalmente git como software de gestión; es el estándar de la industria en su especialidad, y posee una gran cantidad de herramientas y opciones que permiten a los desarrolladores realizar modificaciones al código en diferentes puntos en el tiempo, contribuir colaborativamente en un mismo proyecto sin problemas de sobreescritura, o llevar varios desarrollos paralelos que después juntar o descartar.

En el caso de este proyecto, limitaremos las opciones para utilizar git de manera lineal, y haremos una implementación transparente de DVC que permita al usuario ignorar las pequeñas gestiones en línea de comandos necesarias para sincronizar todo el proceso. También permitiremos cambiar la versión del espacio de trabajo a una más antigua, para que el usuario pueda acceder a los archivos de versiones anteriores, y realizar modificaciones para un nuevo commit "desde el pasado". Para conseguir esto, simularemos el proceso que realizaría un humano en línea de comandos:

```
git checkout [commit]
```

```
dvc checkout
```

Cuyo equivalente en un script de Python sería ejecutar el siguiente código, que utiliza el módulo `subprocess` para mandar comandos a una terminal:

```
subprocess.run(['git', 'checkout', commit])
```

```
subprocess.run(['dvc', 'checkout'])
```

### 4.2.2. Pipelines

La forma de configurar pipelines que nos proporciona DVC es a través de un archivo `dvc.yaml` con un formato concreto. En este archivo, se especifica cada etapa de la forma siguiente:

```
etapas:
    preparacion:
        comando: python preparacion.py datosBase.xml
        dependencias:
            - preparacion.py
            - datosBase.xml
        parametros:
            - preparacion.seed
            - preparacion.split
        output:
            - datosPreparados.xml
    entrenamiento:
        comando: python entrenamiento.py datosPreparados.xml
        dependencias:
            - entrenamiento.py
            - datosPreparados.xml
        output:
            - modelo
    ...
```

En este ejemplo en pseudo-código, se ve claramente la estructura de "pipeline": se ejecuta el comando especificado que a su vez ejecuta un script de Python junto a un archivo de datos dado como argumento; al terminar la ejecución se genera otro archivo de datos, que a su vez se convierte en el argumento de un nuevo comando con un nuevo script de entrenamiento. Mediante este formato se pueden incluir muchas más etapas.

La implementación en el programa consistirá en una ventana específica, que mostrará la pipeline del último commit del repositorio; o lo que es lo mismo, accederá al archivo `dvc.yaml` y mostrará su contenido en pantalla de forma organizada en una tabla. Una vez accedido a esta pantalla, se podrá modificar la pipeline editando el archivo `dvc.yaml` a través de un acceso directo, o editar el archivo de parámetros `params.yaml` de la misma forma. Una vez modificada la pipeline, se podrá ejecutar desde el propio programa y visualizar rápidamente las métricas de esta ejecución, para que el usuario tenga una idea de los resultados del experimento. Después de hacer un commit, podrá ver estos resultados junto a sus parámetros correspondientes en la ventana principal del programa. El acceso a los datos de estos archivos es fácil ya que el formato `.yaml` está diseñado para ser legible por el ser humano, de forma similar al formato `.json`, y dispone de amplio soporte: utilizaremos el módulo `pyyaml` de Python para cargar los datos del archivo en un diccionario, un tipo de datos de Python fácil de manipular.

El usuario también podrá acceder a la pipeline de versiones anteriores, cambiando de commit activo en la ventana principal (en la lista de commits) como se describe en la sección anterior y entrando entonces a la ventana de pipelines.

### 4.2.3. Visualización de métricas y parámetros

En DVC, el concepto de "métricas" es muy amplio y su soporte limitado. Lo que permite el software es añadir un output de una etapa de la pipeline añadiendo una nueva sección llamada `metrics`. Esto se hace generalmente en la etapa de evaluación, y marca este archivo para que sea leído por el comando `dvc metrics show`, resultando en una etapa con una estructura similar a la siguiente:

```
etapas:
    ...
    entrenamiento:
        comando: python entrenamiento.py datosPreparados.xml
        dependencias:
            - entrenamiento.py
            - datosPreparados.xml
        output:
            - modelo
        metricas:
            - metricas.json
```

El archivo `metricas.json` se tiene que crear en el script `entrenamiento.py`, incluyendo el cálculo de las propias métricas: lo único que hace DVC es marcarlo de forma interna para gestionarlo como métricas que se pueden comparar posteriormente.

Para visualizar las métricas en el programa, vamos a incluirlas en la pantalla principal del programa junto a sus parámetros correspondientes. La ventana principal consiste en una lista de los commits del repositorio seleccionado, junto a sus métricas y parámetros asociados. Las métricas y los parámetros exactos de cada proyecto pueden variar, tanto en tipo como en número; el programa debe adaptarse a esto y crear dinámicamente una interfaz que los muestre independientemente tanto de cantidad como de tipo.

## 4.3 Tecnología utilizada

El desarrollo de este programa se ha realizado íntegramente en Python, utilizando además Tkinter como módulo para la implementación de la interfaz, y git y DVC como módulos para el desarrollo de la funcionalidad del programa.

**Python** Uno de los lenguajes de programación más populares en la ciencia de datos con diferencia, con numerosos módulos que facilitan mucho estos proyectos. El programa se ha desarrollado en Python 3.10. Ha sido elegido parcialmente porque la API de DVC, parte esencial del programa, sólo estaba disponible en Python.

**Tkinter** Un módulo que añade objetos de interfaz a Python, como cuadros de texto, botones, o menús desplegables. Se ha elegido por su relativa simpleza y su popularidad, que hace que sea fácil encontrar documentación y tutoriales.

**git** Aunque git en sí mismo es un software independiente, dispone de un módulo de Python que implementa muchas de sus funciones en código. En ocasiones, se han tenido que hacer llamadas de línea de comandos.

**DVC** De la misma forma que git, DVC es un software de línea de comandos que funciona de forma autónoma (junto a git). También dispone de un módulo de Python que implementa algunas de sus funciones, aunque muchas de ellas no lo están y deben llamarse a través de comandos mandados desde un script de Python.

# CAPÍTULO 5
# Desarrollo de la solución propuesta

El programa se ha diseñado siguiendo el principio simple de acceder lo más rápido posible a la información más demandada por el usuario, que generalmente es la versión del experimento junto a las métricas de rendimiento de dicho experimento, y tener las opciones de edición del experimento en una posición accesible. Se ha llamado "Data History and Experiment Manipulation", o DHEM.

## 5.1 Ventana principal

Al acceder al programa, el usuario se encuentra con una ventana vacía con la que sólo se puede interactuar a través del menú de opciones en la parte superior. Este menú incluye las siguientes opciones:

- Nuevo (5.2.1): el usuario selecciona la ruta del nuevo proyecto y le pone un nombre, inicializando un repositorio git y DVC.
- Abrir (5.2.2): carga el proyecto seleccionado en el programa, o lanza una alerta si la carpeta seleccionada no es un repositorio de git.
- Commit (5.3): un menú desplegable que le da la opción al usuario de crear un nuevo commit o eliminar el último del repositorio. Ambas opciones dan un error si no hay un repositorio cargado.
- Pipelines (5.4): abre una nueva ventana que presenta la pipeline del commit seleccionado en la ventana principal, y da acceso a opciones adicionales. Si no existe la pipeline, da un error.
- Salir (5.2.3): cierra el programa, devolviendo el repositorio al commit master si se ha cambiado y restaurando los cambios realizados en master.

Debido a la variabilidad de métricas y parámetros, de los cuales el usuario puede incluir una cantidad teóricamente ilimitada, la ventana principal tiene una barra de desplazamiento tanto vertical como horizontal, asegurando la correcta visualización de todos los datos en una sola pantalla.

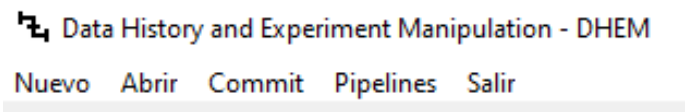


**Figura 5.1:** Menú principal de DHEM

## 5.2 Gestión del proyecto

### 5.2.1. Crear proyecto

Crear un proyecto es posiblemente la funcionalidad más simple de todo el programa, incluso más que cerrarlo. Se abre una pequeña ventana que incluye dos cuadros de texto, uno para el nombre del nuevo proyecto, y otro para la ruta en la que guardarlo, pudiendo también hacerse clic en un botón para seleccionar la ruta e introducirla en el cuadro de texto. Después se confirma la acción, y el programa se encarga de crear la carpeta contenedora e inicializar dentro un repositorio tanto de git como de DVC.

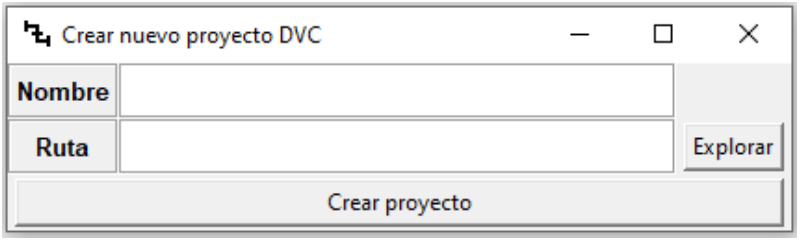


**Figura 5.2:** Menú de creación de proyectos de DHEM

La creación de estos repositorios se crea mediante una llamada directa a los comandos `git init` y `dvc init` de forma secuencial, a través del módulo de Python `subprocess` de la forma que se describe en el apartado 4.2.1:

```
subprocess.run(['git', 'init'])
subprocess.run(['dvc', 'init'])
```

Durante este proceso se realizan algunas comprobaciones para evitar errores en la creación de la carpeta contenedora, y el usuario ya tiene disponible su proyecto vacío para introducir archivos.

### 5.2.2. Abrir proyecto

Esta es una de las funciones principales del programa. Al abrir el proyecto, debe plasmar en pantalla toda la información relevante para el usuario; el código del commit, la fecha, el mensaje, y por último las métricas y parámetros que pueda (o no) haber. Para conseguir esto, se obtienen primero los commits del proyecto bajo la forma de una lista de objetos "Commit" que se pasa por una de las funciones auxiliares descritas en el apartado 5.5, y después se construye la interfaz en una tabla cuyas primeras 4 columnas consisten siempre de: botón, columna de código, columna de fecha, columna de mensaje.

El botón permite al usuario cambiar el commit activo del repositorio, con el añadido de que guarda los cambios aún no registrados en el commit master en un "almacén" que ofrece git bajo el comando `git stash`. Este comando permite al usuario apartar los cambios que tenga a medias y no quiera registrar todavía en el repositorio; una vez guardados de esta forma, ya puede moverse temporalmente de versión en versión. Cabe destacar que el programa no guarda de la misma manera los cambios realizados en versiones anteriores del proyecto, mostrando una advertencia en el caso de que el usuario vuelva a cambiar de commit y el programa detecte cambios en una versión anterior, así que tendría que hacer un nuevo commit con esos cambios para guardarlos manualmente.

| | Código | Fecha | Mensaje |
|---|---|---|---|
| Activo | 018092142770075b771ba1370e350d421ef07eb3 | 29 Aug 2023 16:31 | Experiment 6 |
| Cambiar | 4600276452892e4a6d5504985f4b77b40c0ab8ea | 29 Aug 2023 15:48 | Fixes to data2.xml.dvc |
| Cambiar | 70821e24eb221551dfbd0b29c978171c34cd687a | 29 Aug 2023 14:14 | Experiment 5 |
| Cambiar | 73cbdca90a47c93633d788f7b64507f59720d56a | 29 Aug 2023 13:41 | Experiment 4 |
| Cambiar | 85b2e86e57d4adb44ac8cbaec18c4ab5365bc5ad | 28 Aug 2023 22:29 | data2.xml added |
| Cambiar | 8398423a9329250c75a29dd59d392db8a03e14ad | 10 Aug 2023 18:11 | index.html changed |
| Cambiar | a547197ef8a48abd772a687223e64c4456177e34 | 02 Aug 2023 16:05 | Experiment 3 |
| Cambiar | ebf60f0e1025cf5b7bae25673281649745c1ad9b | 02 Aug 2023 15:58 | Experiment 2 |
| Cambiar | 28fc6710a954e78077f5e2b9f8b8a7cff91e5711 | 02 Aug 2023 15:46 | Experiment: max_features 50 -> 200 and ngrams 1 -> 2 |
| Cambiar | cc7db11388ce5cd89410e3754d42fb8a1b1e2484 | 02 Aug 2023 14:55 | Create evaluation stage |
| Cambiar | 11255bfb05834b63cb93eb57f2a46bc0011030b5 | 02 Aug 2023 14:25 | Tests de parámetros |
| Cambiar | 774057d8c9e6101eb59c9d1988ce7a9600da4a6e | 02 Aug 2023 14:21 | Second pipeline repro, params.yaml/n_est 50 to 100 |
| Cambiar | 0f00e1a6e13b2db06f5f40e5e6d270c7f7f3154d | 02 Aug 2023 14:19 | First pipeline repro |
| Cambiar | 4ba10ce8f9e73a465ae719680dad519c53e9d846 | 02 Aug 2023 14:17 | Pipeline stages added 'featurize' and 'train' |
| Cambiar | a57e25dcd3cada74325fabaa0cecd1096adb3949 | 02 Aug 2023 12:23 | Data tracked in DVC and 'prepare' pipeline stage set up |
| Cambiar | c62f16ac85bbef9fd7297285423b0a4ba0546afc | 02 Aug 2023 12:21 | Stopped tracking data/data.xml in git |
| Cambiar | 8e6c0b7ec170735ed1be5a796630acefdb2a3ded | 02 Aug 2023 12:04 | Initial commit |

**Figura 5.3:** Historial de commits de DHEM (botón, mensaje, fecha y código)

Además de esas cuatro columnas, el programa también obtiene las métricas y parámetros de cada commit y las representa en la interfaz. Ya que el número de métricas y parámetros puede cambiar de versión en versión (aunque no es muy frecuente), el programa lee las métricas y parámetros del commit master, y entonces busca los valores correspondientes a estas de todos los commits. Tiene este funcionamiento en lugar de representar todas las métricas/parámetros que se puedan encontrar potencialmente ya que se asume que la versión más actual es en la que el programador ha codificado los valores más importantes para evaluar y entrenar el modelo. Una vez tiene esta información, el programa simplemente genera un número de columnas acorde al número de métricas y parámetros, y les asigna iterativamente los valores que les corresponden, dejando la celda vacía si la métrica/parámetro no existía en el commit.

| Métricas | | | | Parámetros | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| avg_prec/train | avg_prec/test | roc_auc/train | roc_auc/test | prepare/split | prepare/seed | featurize/max_features | featurize/ngrams | train/seed | train/n_est | train/min_split |
| 0.99338 | 0.95388 | 0.99704 | 0.96745 | 0.2 | 20170428 | 1000 | 10 | 20170428 | 100 | 0.01 |
| 0.992 | 0.9497 | 0.99632 | 0.96534 | 0.2 | 20170428 | 800 | 6 | 20170428 | 100 | 0.01 |
| 0.992 | 0.9497 | 0.99632 | 0.96534 | 0.2 | 20170428 | 800 | 6 | 20170428 | 100 | 0.01 |
| 0.9895 | 0.94426 | 0.99529 | 0.9627 | 0.2 | 20170428 | 600 | 4 | 20170428 | 100 | 0.01 |
| 0.98566 | 0.93886 | 0.99335 | 0.95846 | 0.2 | 20170428 | 400 | 3 | 20170428 | 100 | 0.01 |
| 0.98566 | 0.93886 | 0.99335 | 0.95846 | 0.2 | 20170428 | 400 | 3 | 20170428 | 100 | 0.01 |
| 0.98566 | 0.93886 | 0.99335 | 0.95846 | 0.2 | 20170428 | 400 | 3 | 20170428 | 100 | 0.01 |
| 0.97514 | 0.92353 | 0.98713 | 0.94428 | 0.2 | 20170428 | 200 | 2 | 20170428 | 100 | 0.01 |
| 0.92351 | 0.84268 | 0.96038 | 0.89506 | 0.2 | 20170428 | 50 | 1 | 20170428 | 100 | 0.01 |
| 0.92351 | 0.84268 | 0.96038 | 0.89506 | 0.2 | 20170428 | 50 | 1 | 20170428 | 100 | 0.01 |
| | | | | 0.2 | 20170428 | 50 | 1 | 20170428 | 100 | 0.01 |
| | | | | 0.2 | 20170428 | 100 | 1 | 20170428 | 50 | 0.01 |
| | | | | 0.2 | 20170428 | 100 | 1 | 20170428 | 50 | 0.01 |
| | | | | 0.2 | 20170428 | 100 | 1 | 20170428 | 50 | 0.01 |
| | | | | 0.2 | 20170428 | 100 | 1 | 20170428 | 50 | 0.01 |
| | | | | 0.2 | 20170428 | 100 | 1 | 20170428 | 50 | 0.01 |
| | | | | 0.2 | 20170428 | 100 | 1 | 20170428 | 50 | 0.01 |

**Figura 5.4:** Historial de commits de DHEM (métricas y parámetros)

### 5.2.3. Cerrar proyecto

El cierre del proyecto era inicialmente muy sencillo, pero se complicó una vez entró en juego el cambio de versiones. Al permitir al usuario cambiar de commit activo, y por tanto cambiar el estado de los archivos de la carpeta contenedora, era importante gestionar en qué versión se quedaba el repositorio al cerrar el programa. Sabiendo que era posible que el usuario modificara el repositorio manualmente sin utilizar el programa, optamos por dejar siempre el repositorio en el commit master (el más actual) y evitar problemas de cambios realizados en versiones antiguas que no disponen de los cambios más recientes, haciendo así una implementación lineal del control de versiones de git.

Esto implica comprobar el estado del repositorio, y si la versión activa no es master, avisar al usuario de que cualquier cambio que haya realizado sin hacer un nuevo commit será perdido debido a este cambio de versión que fuerza el programa. Si bien es una decisión que puede frustrar a usuarios más experimentados, este programa tiene como público objetivo estudiantes y personas con pocos conocimientos de estos sistemas; hemos optado por asegurar la integridad de los datos y evitar confusiones en cuanto a la edición de versiones antiguas. En la misma línea, tanto volver manualmente al commit master como cerrar el programa extraen de `git stash` cualquier cambio que estuviera almacenado y lo re-aplica al repositorio, reforzando este concepto de progreso lineal en lugar de ramificado.

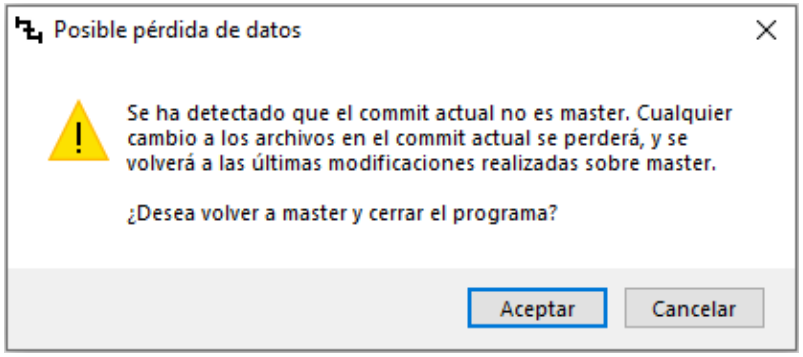


**Figura 5.5:** Aviso de cambio de versión al cerrar DHEM

## 5.3 Commits

### 5.3.1. Crear commit

Esta funcionalidad aparenta ser sencilla, pero terminó siendo la más compleja del programa. Esto se debe a las diferentes opciones que tienen los comandos involucrados, y a querer permitir a los usuarios hacer uso de esta flexibilidad. La interfaz de creación de un nuevo commit está también estructurada en una cuadrícula, dividida en dos partes separadas por el botón de "Añadir archivos". En la parte superior tenemos un cuadro de texto sólo de lectura que muestra el estado del repositorio, utilizando directamente el comando `git status` y su equivalente en DVC `dvc data status`. Este cuadro se actualiza cuando el usuario añade archivos o crea el commit, para ofrecer una vista actualizada del proyecto. Justo después se encuentran los selectores de archivos a añadir tanto a través de git como de DVC, aunque tienen una diferencia esencial, y es que se puede seleccionar la opción "Todos" para git y no para DVC. Esto se debe a que git es el sistema que realizará el trabajo en el "día a día", mientras que el usuario sólo añadirá archivos a través de DVC cuando se añadan archivos grandes como una nueva fuente de datos, cosa que rara

vez ocurre. En el caso de que ocurra, el programa se encarga de añadir también el archivo puntero (`archivo.dvc`) a través de git, de forma que el usuario no tiene que preocuparse de ello.

Una vez el usuario ha añadido los archivos, tiene que escribir el mensaje del commit en el cuadro correspondiente (o se mostrará un error, ya que git requiere que se incluya un mensaje en los commits) y tendrá 3 opciones para elegir los archivos que quiere incluir en el commit:

- No seleccionar ningún archivo: esto hará que el commit incluya sólo los archivos añadidos manualmente por el usuario en el apartado anterior.
- Archivos seleccionados: el commit sólo incluirá los archivos seleccionados manualmente por el usuario, que deben haber sido añadidos previamente en el apartado anterior.
- "Todos": el commit incluirá cualquier archivo que haya sido modificado, haya sido añadido manualmente o no. En la práctica, esta opción vuelve inútil la sección anterior para añadir manualmente archivos, pero tiene el problema de que no incluye nuevos archivos introducidos en la carpeta del proyecto, sólo modificados que ya estuvieran seguidos por git (o DVC).

Todas estas opciones, tanto a la hora de añadir archivos como a la hora de realizar el commit, están clarificadas en una pequeña ventana de ayuda que se muestra al pulsar el botón correspondiente "?" a la izquierda de cada sección.

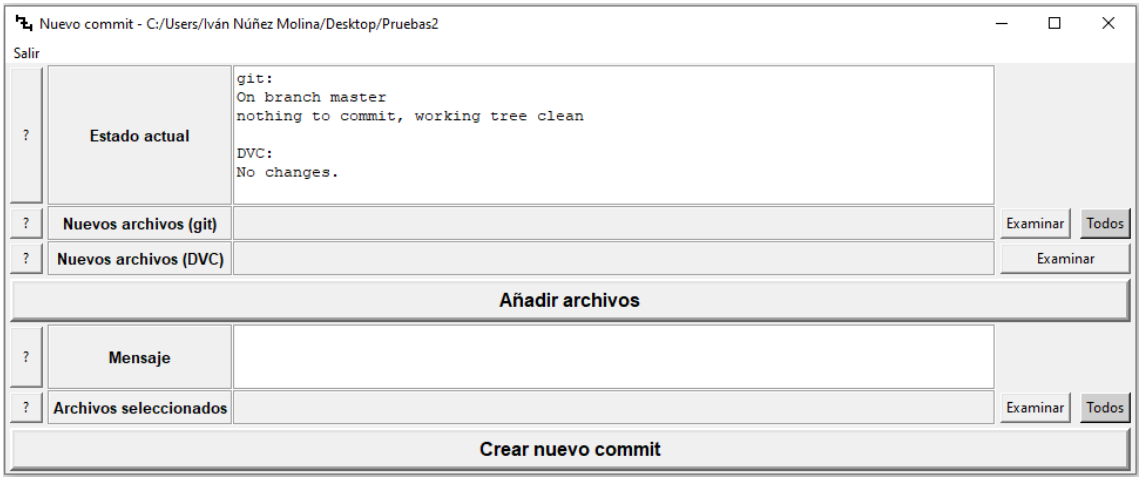


**Figura 5.6:** Ventana de creación de commit de DHEM

### 5.3.2. Eliminar último commit

Eliminar el último commit es, por otro lado, una función mucho más simple. Consiste básicamente en llamar los siguientes 4 comandos a través del módulo `subprocess`:

`git reset --hard HEAD~` Borra el último commit del registro, dejando las diferencias con el penúltimo commit como modificaciones.

`git restore *` Restaura esas modificaciones para igualar el repositorio actual con la última versión.

`git checkout master` Realiza el cambio de commit de forma ”oficial”, sincronizando el registro de versiones con los contenidos de la carpeta del proyecto.

`dvc checkout` Sincroniza los archivos seguidos a través de DVC con el commit master.

Después de comprobar que el usuario realmente desea borrar el último commit e informándole de que es una acción sin vuelta atrás, esta simple secuencia de comandos se asegura de devolver el repositorio a un estado anterior.

## 5.4 Ventana de pipelines

La ventana de pipelines hace a la vez de vitrina para la pipeline del proyecto y de acceso a las diferentes funcionalidades que se derivan de ella. La pipeline está definida, como se ha mencionado anteriormente, en el archivo `dvc.yaml` que se encuentra en la carpeta contenedora del proyecto. Se puede leer usando una vez más el módulo de Python `pyyaml`, obteniendo así un objeto de código fácil de manipular, y después de eso tenemos que ocuparnos de presentárselo de una forma clara y legible al usuario. Debido a la naturaleza teóricamente ilimitada de las etapas y opciones de una pipeline, esta ventana también dispone de barras de desplazamiento tanto vertical como horizontal.

Para conseguir esto, recorremos las opciones (dependencias, outputs, parámetros, etc) de cada etapa, y nos aseguramos de que todas las opciones de todas las etapas aparezcan en pantalla, aunque algunas de las etapas no tengan algunas de las opciones. En el siguiente ejemplo, se ve cómo la etapa ”evaluate” no tiene `params`, y el resto de etapas no tienen `metrics`:

Pipelines - C:/Users/Iván Núñez Molina/Desktop/Pruebas2/dvc.yaml

Añadir etapa Editar pipeline Editar parámetros Ejecutar pipeline Salir

| | prepare | featurize | train | evaluate |
|---|---|---|---|---|
| cmd | python src/prepare.py data/data.xml | python src/featurization.py data/prepared data/features | python src/train.py data/features model.pkl | python src/evaluate.py model.pkl data/features |
| deps | data/data.xml<br>src/prepare.py | data/prepared<br>src/featurization.py | data/features<br>src/train.py | data/features<br>model.pkl<br>src/evaluate.py |
| params | prepare.seed<br>prepare.split | featurize.max_features<br>featurize.ngrams | train.min_split<br>train.n_est<br>train.seed | |
| outs | data/prepared | data/features | model.pkl | eval/importance.png<br>eval/live/plots {'cache': False}<br>eval/prc {'cache': False} |
| metrics | | | | eval/live/metrics.json {'cache': False} |

**Figura 5.7:** Ventana de pipelines de DHEM

Una vez se establecen cuáles son las etapas y cuáles son los parámetros, es simplemente cuestión de rellenar las celdas de datos de forma iterativa como si fuera un mapa de coordenadas, gracias al tipo de datos ”diccionario” de Python, que asocia a una clave única un valor. Por ejemplo, para acceder a las dependencias de ”prepare”, seguiríamos el siguiente proceso:

- dvc.yaml → stages
- stages → prepare
- prepare → deps
- deps → lista de dependencias

### 5.4.1. Crear etapa

La ventana de creación de etapas es simplemente una ventana que actúa como interfaz entre el usuario y el comando de DVC `dvc stage add [argumentos]`, que añade una etapa al archivo `dvc.yaml` siguiendo las instrucciones del usuario. Se comporta de la siguiente manera:

`dvc stage add` El comando inicial.

`--name` Especifica el nombre de la etapa. Obligatorio.

`--deps` Especifica las dependencias de la etapa. Opcional, acepta varias dependencias llamando repetidas veces a `--deps`.

`--outs` Especifica los outputs de la etapa que sí se siguen con DVC. Opcional, acepta varios outputs llamando repetidas veces a `--outs`.

`--outs-no-cache` Especifica los outputs de la etapa que no se siguen con DVC. Opcional, acepta varios outputs llamando repetidas veces a `--outs-no-cache`.

`--params` Especifica los parámetros de la etapa a partir de `params.yaml`. Opcional, acepta varios parámetros dando como argumento una lista de elementos separada con ",".

`--metrics-no-cache` Especifica los outputs que se marcan como métricas y no se siguen con DVC. Opcional, acepta varias métricas llamando repetidas veces a `--metrics-no-cache`.

`--plots` Especifica los gráficos generados por el script de la etapa. Opcional, acepta varios gráficos llamando repetidas veces a `--plots`.

`[vacío]` Después de todos los argumentos, cualquier texto que se introduzca sin especificar el argumento será considerado como el comando a ejecutar. Obligatorio.

El comando en sí tiene más argumentos disponibles, pero hemos seleccionado esos para incluir en el programa debido a que son los más relevantes para un usuario novel. En la siguiente figura, se ve claramente como la interfaz del programa imita directamente la estructura del comando a través de cuadros de texto y botones.

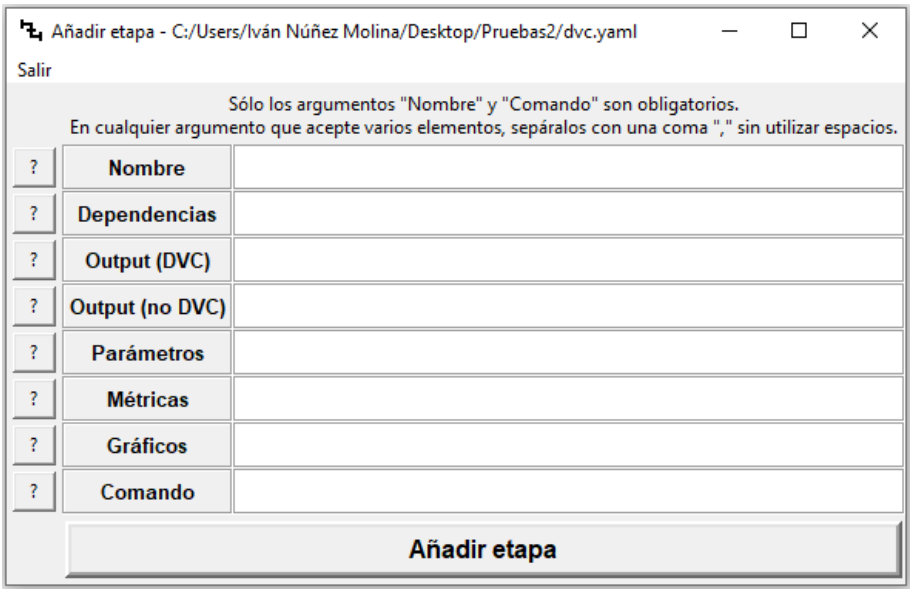


**Figura 5.8:** Ventana de creación de etapas de DHEM

El usuario dispone además de botones de ayuda "?" a la izquierda de cada opción que explican brevemente su funcionamiento, y una advertencia arriba del todo para evitar errores de formato. También cabe destacar que no tiene por qué llamar varias veces a ningún argumento, o notar la diferencia entre la forma de poner varios parámetros con `--params` y con los otros argumentos que aceptan varios elementos, el programa se encarga de hacer ese proceso transparente.

### 5.4.2. Editar pipeline, editar parámetros

Estas dos opciones tienen exactamente el mismo funcionamiento: se limitan a abrir un cuadro de texto que permite al usuario editar directamente los archivos `dvc.yaml` y `params.yaml` respectivamente: utilizan el módulo `pyyaml` para cargar el archivo en memoria, lo colocan en un cuadro de texto en una ventana nueva, y permiten al usuario editarlo como un archivo de texto. Después de eso, el usuario decide si guardar las modificaciones o descartarlas y salir de la ventana.

La funcionalidad más destacable de esta ventana es similar a la que se activa al cerrar la ventana principal y detectar cambios en el repositorio: en este caso, cerrar la ventana con modificaciones en el texto avisa al usuario de que van a perderse, y le pide una confirmación adicional. Esto se hace extrayendo el texto del cuadro de texto y comparándolo con el archivo original, que se vuelve a cargar de memoria en modo lectura. Hubo un problema inesperado, y es que en los primeros intentos, cerrar la ventana siempre detectaba cambios en el archivo aunque el usuario no realizara ninguno. Tras varias pruebas, descubrí que esto se debía a que cargar el texto del archivo en un cuadro de texto de Tkinter y volver a extraerlo le añadía un salto de línea al final (carácter "\n"), lo que creaba esas modificaciones. Para solucionarlo, el texto extraído del cuadro tiene siempre el carácter final borrado. Esto también ocurre al guardar el archivo, por lo que se ha seguido el mismo procedimiento.

### 5.4.3. Ejecutar pipeline

La ejecución de una pipeline es un único comando en DVC, por lo que la mayoría del trabajo ha sido de diseño de la interfaz. DVC proporciona muchas facilidades, y para realizar el nuevo experimento, el usuario sólo tiene que ejecutar el comando `dvc repro`, que automáticamente detecta el archivo `dvc.yaml`, las modificaciones a las etapas (o nuevas etapas) y sus dependencias, y ejecuta sólo las que se han visto afectadas por estas modificaciones.

Hacer clic en esta opción del menú muestra un pequeño aviso para pedir la confirmación del usuario, y después ejecuta el comando con el módulo `subprocess`. Al terminar la ejecución, el programa muestra una pequeña ventana similar a la ventana principal, pero que sólo contiene las métricas y parámetros del estado actual del repositorio, o lo que es lo mismo, de la pipeline que acaba de ejecutar el programador. Con esto puede ver de un vistazo los resultados del experimento, y decidir si quiere hacer un nuevo commit y registrar esos resultados junto al resto, o seguir probando.

## 5.5 Funciones auxiliares

Las funciones auxiliares son todas las que no interactúan de ninguna forma con la interfaz. Están definidas en un archivo separado, `functions.py`.

### 5.5.1. Lista de commits

La lista de commits mostrada en la ventana principal está creada por una función definida aparte llamada `get_commit_list_dvc()`, que toma como argumento la ruta del repositorio. El primer paso es extraer la información del mismo repositorio, una tarea que realizamos en varias etapas.

Recordamos que buscamos, para cada commit: código, fecha, mensaje, métricas, parámetros. Para acceder a esta información, contamos con la API de DVC que nos facilita el trabajo:

`dvc.api.scm.all_commits()` Devuelve una lista con los códigos identificadores de los commits del repositorio especificado como argumento.

`dvc.api.scm.metrics_show()` Devuelve un diccionario con las métricas del commit cuyo código se ha especificado como argumento.

`dvc.api.scm.params_show()` Devuelve un diccionario con los parámetros del commit cuyo código se ha especificado como argumento.

Una vez tenemos el código del commit gracias al primer método, podemos utilizarlo para obtener tanto métricas como parámetros, y también para obtener el resto de información utilizando la API de git. Finalmente, la función crea iterativamente objetos "Commit" (un tipo de objeto específico creado también en el archivo `functions.py`) cuyos atributos son los 5 datos que necesitamos, y devuelve una lista de estos objetos Commit.

Existe la posibilidad de que el repositorio seleccionado por el usuario no sea de DVC, sino sólo de git. En este caso los métodos de DVC darían error, así que recurrimos a diseñar la función auxiliar `get_commit_list_codes()`, que actúa como sustituta de la función `dvc.api.scm.all_commits()` (aunque es más lenta) y devuelve los códigos de cada commit. En este caso, al no ser un repositorio DVC no existen métricas y parámetros, así que la función principal devuelve una lista de objetos Commit que sólo contiene código, fecha y mensaje, que es lo que después se muestra en la ventana principal. Aunque este no era el objetivo principal del proyecto, fue un pequeño añadido que costaba poco esfuerzo, y sumaba algo de funcionalidad al programa.

### 5.5.2. Funciones de formato

Para simplificar el código en el archivo principal (`start.py`), se han añadido dos funciones simples al archivo auxiliar: la primera es `flatten_dict()`, que se encarga de transformar los diccionarios (en este caso, de métricas y parámetros) para "aplanarlos". Esto significa que, al ser estos diccionarios anidados (se contienen unos a otros en varios niveles), la función recoge todas las claves hasta llegar al valor final, une todas las claves en una sola, y le asigna el valor final. Como ejemplo práctico, el diccionario `stages` viene de un archivo `.yaml`:

```
1  stages:
2      prepare:
3          name: ABC
4          deps: DEF
5          cmd: GHI
6      featurize:
7          ...
```

Tenemos el diccionario `stages`, que contiene como valor la lista de diccionarios `prepare` y `featurize`, que a su vez contienen unas claves y valores. Al usar la función, obtendríamos algo así:

```
stages/prepare/name: ABC
stages/prepare/deps: DEF
stages/prepare/cmd: GHI
stages/featurize/...
```

Aprovechamos esta funcionalidad para facilitar el acceso a los datos, y poder representarlos más fácilmente en la ventana principal.

Por otro lado tenemos la función `get_yamljson`, cuyo objetivo es devolver el contenido del archivo (ya sea `.yaml` o `.json`) que se le haya pasado como argumento, buscándolo de forma recursiva en la carpeta que se le da y en las carpetas que esta contiene. Esta función realiza en el programa una tarea similar a la que se hace al cargar las métricas y parámetros en la ventana principal al abrir un proyecto, pero se utiliza al ejecutar la pipeline para mostrarlas después de la ejecución.

# CAPÍTULO 6
# Pruebas

Para verificar el correcto funcionamiento del programa, se ha planteado una lista de pasos que daría un usuario para utilizarlo, desde la creación del proyecto hasta la comparación de métricas de varios experimentos. En este escenario, el usuario quiere obtener un modelo y evaluarlo, aportando sólo los scripts en Python y el archivo de datos. Realiza las siguientes acciones:

1. Crea un nuevo proyecto.
    - **Resultado esperado**: se crea una carpeta con un repositorio de git y uno de DVC inicializados.
2. Inserta en la carpeta del proyecto la carpeta `src`, que contiene los archivos de código fuente `prepare.py`, `featurize.py`, `train.py` y `evaluate.py`.
    - **Resultado esperado**: el cuadro de estado del repositorio de la ventana "Nuevo commit" muestra la introducción de la carpeta `src` en el repositorio.
3. Añade estos archivos a git, realiza un commit, y vuelve a abrir el proyecto para ver los cambios.
    - **Resultado esperado**: el cuadro de estado de la ventana "Nuevo commit" muestra los archivos como *staged* al añadirlos a git, y al hacer el commit, desaparecen. Al volver a abrir el proyecto, aparecen los datos de este commit en pantalla.
4. Inserta en la carpeta del proyecto la carpeta `data`, que contiene el archivo de datos `data.xml`.
    - **Resultado esperado**: el cuadro de estado de la ventana "Nuevo commit" muestra la introducción del archivo `data.xml` en el repositorio, bajo la sección de git.
5. Añade el archivo `data.xml` a DVC, realiza un commit, y vuelve a abrir el proyecto para ver los cambios.
    - **Resultado esperado**: al añadir el archivo `data.xml` a través de DVC, el cuadro de estado muestra el nuevo archivo `data.xml.dvc` como *staged* en la sección de git, mientras que muestra el archivo `data.xml` como *staged* en la sección de DVC. Al volver a abrir el proyecto, se añaden los datos de este commit a la lista en pantalla.

6. Añade los parámetros, añade el archivo a git, realiza un commit, y vuelve a abrir el proyecto para ver los cambios.

    - **Resultado esperado**: se crea el archivo `params.yaml` con los contenidos introducidos. El cuadro de estado muestra el archivo de parámetros, que cambia a "staged" al añadirlo a git, y desaparece al hacer el commit. Al volver a abrir el proyecto, se añaden los datos de este commit. Aparece también la sección de parámetros, que muestra los valores introducidos. Los commits que no disponen de parámetros muestran celdas vacías en su lugar.

7. Crea la pipeline, añadiendo las etapas de una en una, añade el archivo a git, realiza un commit, y vuelve a abrir el proyecto para ver los cambios.

    - **Resultado esperado**: se crea el archivo `dvc.yaml` con las etapas introducidas a través del botón de "Añadir etapa". El cuadro de estado muestra el archivo, que cambia a "staged" al añadirlo a git, y desaparece al hacer el commit. Al volver a abrir el proyecto, se añaden los datos de este commit. No aparece aún la sección de métricas, ya que no se han calculado todavía.

8. Ejecuta la pipeline, realiza un commit, y vuelve a abrir el proyecto para ver los cambios.

    - **Resultado esperado**: al terminar la ejecución se abre una ventana que muestra los parámetros y métricas. El cuadro de estado muestra cambios en varios archivos relacionados con los outputs de la pipeline y en `dvc.lock`, que estarán en la sección de git o de DVC según las especificaciones en la pipeline. Al añadir los archivos de la forma que corresponda, se marcan como *staged* y desaparecen al hacer el commit. Al volver a abrir el proyecto, se añaden los datos de este commit. Aparece también la sección de métricas, que muestra los valores calculados. Los commits que no disponen de métricas muestran celdas vacías en su lugar.

9. Realiza cambios en el archivo de datos `data.xml`, registra los cambios en DVC, realiza un commit, y vuelve a abrir el proyecto para ver los cambios.

    - **Resultado esperado**: el cuadro de estado muestra el archivo `data.xml` en la sección de DVC. Su estado cambia a *staged* al añadirlo, y desaparece cuando se realiza el commit. Al volver a abrir el proyecto, se añaden los datos de este commit.

10. Realiza cambios en los parámetros.

    - **Resultado esperado**: el cuadro de estado muestra el archivo `params.yaml` en la sección de git.

11. Ejecuta la pipeline, realiza un commit, y vuelve a abrir el proyecto para ver los cambios.

    - **Resultado esperado**: al terminar la ejecución se abre una ventana que muestra los parámetros y métricas. El cuadro de estado muestra cambios en varios archivos relacionados con los outputs de la pipeline y en `dvc.lock`, que estarán en la sección de git o de DVC según las especificaciones en la pipeline. Al añadir los archivos de la forma que corresponda, se marcan como *staged* y desaparecen al hacer el commit. Al volver a abrir el proyecto, se añaden los datos de este commit.

12. Cambia el estado del repositorio desplazándose a un commit anterior.

- **Resultado esperado**: el botón que corresponde a master se vuelve gris y su texto cambia a "Cambiar", mientras que el correspondiente al commit anterior se vuelve verde y su texto cambia a "Activo". La carpeta del proyecto se actualiza al commit seleccionado, y el cuadro de estado muestra que el repositorio se haya en un commit diferente a master con la anotación "HEAD detached".

13. Realiza cambios en los parámetros.

    - **Resultado esperado**: el cuadro de estado muestra el archivo `params.yaml` en la sección de git.

14. Ejecuta la pipeline.

    - **Resultado esperado**: al terminar la ejecución se abre una ventana que muestra los parámetros y métricas. El cuadro de estado muestra cambios en varios archivos relacionados con los outputs de la pipeline y en `dvc.lock`, que estarán en la sección de git o de DVC según las especificaciones en la pipeline.

15. Intenta volver al commit master.

    - **Resultado esperado**: el programa muestra una advertencia sobre la pérdida de datos por cambiar de versión sin haber guardado los cambios con un commit.

16. Realiza un commit y vuelve a abrir el proyecto para ver los cambios.

    - **Resultado esperado**: Al añadir los archivos de la forma que corresponda, se marcan como *staged* y desaparecen al hacer el commit. Al volver a abrir el proyecto, se añaden los datos de este commit y se actualiza automáticamente el repositorio al commit master.

El programa ha funcionado correctamente en la mayoría de estos pasos, reflejando el resultado esperado. El programa ha dado un error inesperado al volver a abrir el proyecto, ya que aunque cerrando el programa se vuelve automáticamente al commit master, esto no estaba implementado al abrir un proyecto cuando ya tenías otro abierto y en un commit antiguo. Esto ha sido corregido durante las pruebas.

Otro error que se detectó por azar durante las pruebas fue que, debido a la forma en la que está programada la función de abrir un proyecto, el programa "olvidaba" la ruta del proyecto si el usuario hacía clic en el botón de "Abrir proyecto" pero cancelaba la selección. Esto hacía que no pudiera acceder a los menús de creación de commits o pipeline, y que tampoco pudiera cambiar de commit. Ha sido corregido.

# CAPÍTULO 7
# Manual de usuario

Esta sección está dedicada a guiar al usuario en el uso del programa. Prioriza dar explicaciones prácticas sobre el uso, y sirve como referencia cuando el usuario tiene dudas sobre el funcionamiento exacto de alguna parte de DHEM.

## 7.1 Instalación

El programa se distribuye en un archivo comprimido `DHEM.rar` que tiene que descomprimirse. La carpeta resultante contiene, además de todos los ficheros del programa y el archivo ejecutable `DHEM.exe`, un acceso directo. Este acceso directo puede sacarse de la carpeta y ponerse en cualquier sitio para la comodidad del usuario, pero el ejecutable debe mantenerse dentro de la carpeta. La carpeta en cuestión puede ponerse donde el usuario desee, aunque la ruta más popular de instalación es `C:/Program Files/`.

## 7.2 Requerimientos de uso

Para asegurar el correcto funcionamiento del programa, deben cumplirse una serie de requisitos, especialmente en cuanto a la estructura de los archivos del proyecto:

- El proyecto abierto debe ser un repositorio de git, o de DVC y git, pero no sólo de DVC.
- Si no existen parámetros o métricas, estas secciones no aparecerán en el programa.
- Si existen parámetros, deben estar en un solo fichero llamado `params.yaml`, que se encontrará en la carpeta base del proyecto (no dentro de subdirectorios).
- Si existe la pipeline, debe estar en un solo fichero llamado `dvc.yaml`, que se encontrará en la carpeta base del proyecto (no dentro de subdirectorios).
- Si existen métricas, deben estar en un solo fichero llamado `metrics.json`. Este archivo puede estar en cualquier subdirectorio.

DHEM es una implementación lineal de un repositorio git y DVC. Esto significa que no está pensado para hacer ramas o modificaciones en el historial, si no que simplifica su uso.

## 7.3 Descripción de funciones

### 7.3.1. Gestión del proyecto

**Nuevo**

El usuario puede crear un nuevo proyecto haciendo clic en el botón "Nuevo" al abrir el programa. Esto le pide un Nombre, que corresponde al nombre de la carpeta que se creará, y Ruta, que corresponde a la ubicación donde se creará el proyecto. La Ruta se puede introducir manualmente o seleccionarla con el botón "Examinar". Hasta que el usuario no realice el primer commit, la ventana principal del programa sólo mostrará las cabeceras de Código, Fecha y Mensaje.

**Abrir**

El usuario puede abrir un proyecto ya existente haciendo clic en el botón "Abrir", que abre un diálogo estándar de Windows para seleccionar una carpeta. Al confirmarla, el programa lee sus contenidos para mostrar la lista de commits en pantalla, y las métricas y parámetros si existen. Abrir otra vez el proyecto es también la forma de recargar la pantalla para mostrar los nuevos commits.

**Salir**

El usuario puede cerrar el programa a través del botón estándar de cierre de Windows, o a través del botón "Salir", ambos funcionan de forma idéntica. Esto pide una confirmación al usuario, y en caso de que haya activado un commit anterior, le informa de que cualquier modificación que haya realizado en ese commit anterior se perderá. Esto es porque al salir, el programa vuelve automáticamente al commit más reciente que haya hecho el usuario, y borra cualquier cambio en el commit antiguo.

### 7.3.2. Gestión de commits

**Commit → Nuevo commit**

El usuario puede crear un nuevo commit o consultar el estado del repositorio entrando al menú "Commit" y haciendo clic en el botón "Nuevo commit". La ventana resultante permite al usuario añadir archivos tanto a git como a DVC, escribir el mensaje del commit, y realizar el commit tanto especificando los archivos elegidos como eligiéndolos todos de golpe, haciendo clic en la opción "Todos". Para ver estos cambios, deberá volver a abrir el repositorio, aunque el cuadro de estado los muestra nada más realizarse. El usuario también dispone de pequeños botones de ayuda a la izquierda de cada opción, con una breve explicación para cada una de ellas.

**Commit → Eliminar último commit**

El usuario tiene la opción de eliminar el último commit que ha registrado en el repositorio haciendo clic en el botón "Eliminar último commit". Esto muestra al usuario una advertencia incidiendo en que es irreversible, y realiza la eliminación. No se eliminará sólo el último commit, **sino también cualquier cambio que no haya sido guardado**: la intención de esta opción es revertir cambios que no quieres registrar en el repositorio, no modificar el historial de forma retroactiva. La eliminación también pone el repositorio en el commit master (que previamente era el penúltimo).

### 7.3.3. Gestión de pipelines

**Pipelines**

El usuario puede acceder a la ventana de pipelines haciendo clic en el botón correspondiente. Esta ventana es una interfaz que muestra los contenidos del archivo `dvc.yaml` de forma visual, pero no permite interactuar directamente con ella para modificarla. Su función principal es mostrar la pipeline, y dar acceso al resto de funciones que permiten interactuar con ella.

**Pipelines → Añadir etapa**

El usuario puede añadir etapas a la pipeline a través del botón "Añadir etapa". Este botón presenta una serie de cuadros de texto que corresponden a las diferentes secciones de una etapa, junto a una explicación de cada una en un pequeño botón de ayuda a la izquierda. Entre estas secciones, sólo Nombre y Comando son necesarias. También son las únicas que aceptan un sólo elemento (aunque el comando contenga varias palabras separadas por espacios, se considera un solo elemento), mientras que el resto de secciones aceptan varios elementos separados por una coma (",") sin ningún espacio. En el caso de que el archivo `dvc.yaml` no exista todavía, esta opción lo crea en la carpeta base del proyecto.

**Pipelines → Editar pipeline**

El botón "Editar pipeline" abre el archivo `dvc.yaml` en un cuadro de texto editable, creándolo si no existe. El usuario puede editarlo libremente, así que debe tener cuidado de no romper las reglas de formato propias del archivo. Puede entonces decidir si guardar los cambios o salir y descartarlos, aceptando una advertencia que se le muestra si cierra el cuadro de texto sin guardar sus cambios.

**Pipelines → Editar parámetros**

El botón "Editar parámetros" abre el archivo `params.yaml` en un cuadro de texto editable, creándolo si no existe. El usuario puede editarlo libremente, así que debe tener cuidado de no romper las reglas de formato propias del archivo. Puede entonces decidir si guardar los cambios o salir y descartarlos, aceptando una advertencia que se le muestra si cierra el cuadro de texto sin guardar sus cambios.

**Pipelines → Ejecutar pipeline**

El usuario puede ejecutar la pipeline usando el botón "Ejecutar pipeline", que muestra un diálogo de confirmación. Al aceptarlo, la pipeline se ejecuta con el comando `dvc repro`, lo que resulta en tiempos de ejecución muy variables debido al sistema de caché que usa DVC. La ejecución se puede seguir en detalle en la consola que se abre con el programa, y al terminar, se abrirá una nueva ventana con las métricas de la ejecución (si existen) y los parámetros correspondientes. Esta ventana de resultados sólo puede cerrarse, y el usuario deberá hacer un nuevo commit y abrir el proyecto de nuevo para ver los resultados en la ventana principal del programa.

**Pipelines → Salir**

Cierra la ventana de pipelines. Esta acción no afecta a los archivos de ninguna forma, sólo a la interfaz.

# CAPÍTULO 8
# Conclusiones

A lo largo de este proyecto, se ha desarrollado un programa que buscaba ofrecer una interfaz amigable e intuitiva para que usuarios novatos pudieran interactuar con el sistema de gestión de versiones git y su extensión DVC de una forma sencilla. En este sentido, el proyecto ha sido un éxito: pese a la gran complejidad de estos sistemas cuando se exploran en profundidad, la limitación de características necesaria para adaptar el programa a usuarios noveles nos ha permitido limitar el alcance de nuestro trabajo a algo más manejable.

El desarrollo ha comenzado con una exploración de las funciones de git, para familiarizarme con las funcionalidades del sistema base. Las más relevantes para el proyecto fueron la dinámica de add/commit para registrar versiones, el cambio de versiones con checkouts, y el stash para guardar temporalmente los cambios del usuario antes de cambiar de versión desde master. Después de esto fue el turno de descubrir las capacidades de DVC de una forma más práctica, realizando el proyecto de pruebas paso a paso que Iterative ofrece en la sección "Use Cases" de su página web https://dvc.org/. Esto incluía tanto probar los comandos para controlar el repositorio como estudiar el funcionamiento de estos, a fin de poder implementarlos en el programa teniendo en cuenta potenciales errores de input del usuario y preverlos.

Una vez me documenté lo suficiente, llegó la hora de desarrollar el programa en sí mismo. Cada ventana fue diseñada previamente en papel para tener una idea del aspecto visual de cada una, y asegurarme de que no había olvidado elementos importantes. Al usar en Tkinter una cuadrícula para colocar los elementos de la interfaz en lugar de posiciones relativas, fue simple trasladar el diseño del papel al código, debido a poder usar el valor de columnas y filas como "coordenadas" para colocar los elementos.

En cuanto al desarrollo del código, considero que el área en la que más me he enriquecido ha sido todo lo que constituye la creación de un programa completo, lo que se conoce como desarrollo "full-stack". Pese a ser un programa relativamente simple, incluye el uso de APIs, operaciones de lectura/escritura, control de excepciones, compilación para distribuir un programa utilizable por el usuario final, diseño de interfaces, manipulación de archivos/directorios, y conocimiento práctico sobre la programación en Python que nunca había necesitado hasta ahora.

Por otro lado, aunque no era el objetivo del trabajo, he adquirido conocimiento práctico sobre git que no tenía anteriormente, y he descubierto y manipulado directamente DVC. Ambos factores son muy enriquecedores para mí, ya que este proyecto estuvo inicialmente motivado por mi interés por la ciencia de datos. Ahora dispongo tanto de este programa para realizar proyectos de ciencia de datos sin preocuparme demasiado de la gestión de los archivos, como de conocimientos sobre los sistemas subyacentes para poder realizar tareas más complejas si tuviera la necesidad o las ganas de hacerlo.

## 8.1 Relación del trabajo con los estudios cursados

Durante este proyecto, han sido aplicados muchos pedazos de conocimiento adquiridos durante la carrera de Ingeniería Informática. Entre ellos encontramos por ejemplo el proceso de diseño de la interfaz, que busca ser simple e intuitiva para el usuario, incluyendo explicaciones de sus funcionalidades de forma no-invasiva pero accesible. También se han aplicado conocimientos de lenguajes de programación, por supuesto especialmente de Python, que han acelerado mucho el proceso de aprendizaje a la hora de buscar la forma de implementar ciertas funcionalidades. Un último aspecto también destacable ha sido aplicar la idea de que durante la ejecución del programa, debe garantizarse que el usuario puede utilizarlo sin errores fatales. Esto me ha forzado a considerar posibles interacciones por parte del usuario que podrían dar errores, y ha influenciado el diseño de las pruebas.

Un aspecto fácil de olvidar pero que es muy importante durante el desarrollo de cualquier proyecto informático es la facultad de buscar información, tanto para corregir errores como para aprender más sobre algún área específica. El enfoque práctico de la carrera y los numerosos proyectos evaluados hacen que como alumno tengas que esforzarte en crear cosas funcionales también fuera del plano teórico, lo que te lleva frecuentemente a consultar muchas fuentes en tu búsqueda de respuestas. Esto es algo que me ha ayudado mucho en este trabajo, ya que al utilizar en él varios sistemas de los que tenía pocos conocimientos al principio, he tenido que aprender de 0 el funcionamiento de casi todo y resolver muchas dudas que iban surgiendo.

## 8.2 Trabajos futuros

Aunque el proyecto en sí mismo cumple los objetivos que se habían planteado, es fácil imaginar cómo podría continuarse su desarrollo. Una vía razonable sería aumentar las funcionalidades que ofrece, implementando por ejemplo las ramas de git, o dando soporte a los gráficos que se pueden generar con DVC. Estas funcionalidades son más complejas y no tan necesarias para el usuario principiante, pero si se introducen de una forma clara, intuitiva y con naturaleza opcional (como son las métricas en el proyecto actual, técnicamente), pueden convertirse en un segundo paso natural en el aprendizaje del usuario. Sin embargo, sería especialmente importante centrarse en el aspecto educativo: parte del valor que ofrece el programa es precisamente su simpleza y accesibilidad, ya que de otra forma se convierte en un competidor de los programas mencionados en el estado del arte cuya única ventaja es el uso de almacenamiento local.

Un cambio interesante a menor escala sería cambiar el diseño de la ventana de pipelines para que la pipeline en sí misma tuviera una representación de gráfico de dependencias, permitiendo al usuario acceder a la información de cada etapa haciendo clic sobre ella. Aunque no sería necesariamente buena idea eliminar la vista actual de cuadrícula, podría implementarse una opción para cambiar la vista. Este cambio le haría el trabajo ligeramente más sencillo a los usuarios, pero también requiere de un cambio importante de las tecnologías utilizadas, ya que Tkinter no soporta este tipo de vista de forma simple. Es algo que se podría hacer a mano a base de botones, líneas y una programación inteligente, pero desde luego sería mucho más complicado que buscar un sistema que dé soporte directo a este tipo de visualización.

# Bibliografía

[1] FP&A at Standard Chartered Bank: Building Collective Intelligence with Dataiku, consultado el 17 de julio de 2023. Consultado en https://www.dataiku.com/stories/detail/standard-chartered/.

[2] Democratizing Automated Forecasting at Mercedes-Benz, consultado el 17 de julio de 2023. Consultado en https://www.dataiku.com/stories/detail/democratizing-automated-forecasting-at-mercedes-benz/.

[3] Royal Bank of Canada — Bringing Together Auditors & Analysts in a Control Testing Framework, consultado el 17 de julio de 2023. Consultado en https://www.dataiku.com/stories/detail/rbc-control-testing/.

# APÉNDICE A
# Objetivos de Desarrollo Sostenible

## A.1 Relación con ODS

| Objetivos de Desarrollo Sostenible | Alto | Medio | Bajo | No Procede |
|---|---|---|---|---|
| ODS 01. Fin de la pobreza | | | | × |
| ODS 02. Hambre cero | | | | × |
| ODS 03. Salud y bienestar | | | | × |
| ODS 04. Educación de calidad | × | | | |
| ODS 05. Igualdad de género | | | | × |
| ODS 06. Agua limpia y saneamiento | | | | × |
| ODS 07. Energía asequible y no contaminante | | | | × |
| ODS 08. Trabajo decente y crecimiento económico | | × | | |
| ODS 09. Industria, innovación e infraestructuras | | × | | |
| ODS 10. Reducción de las desigualdades | | | | × |
| ODS 11. Ciudades y comunidades sostenibles | | | | × |
| ODS 12. Producción y consumo responsables | | | | × |
| ODS 13. Acción por el clima | | | | × |
| ODS 14. Vida submarina | | | | × |
| ODS 15. Vida de ecosistemas terrestres | | | | × |
| ODS 16. Paz, justicia e instituciones sólidas | | | | × |
| ODS 17. Alianzas para lograr objetivos | | | | × |

**Tabla A.1:** Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS)

## A.2 Reflexión

Si bien no es un proyecto a gran escala con grandes efectos en la industria o el mercado, el programa presentado en este trabajo tiene una relación más o menos estrecha con ciertos Objetivos de Desarrollo Sostenible. La relación se basa principalmente en los objetivos que el programa busca cumplir más que en el programa en sí, ya que como plataforma de desarrollo, es una herramienta en lugar de un producto final con un uso directo para el ciudadano. Los ODS relevantes son principalmente los que tienen que ver con la educación y el desarrollo económico e industrial, debido por un lado a la naturaleza de nuestro proyecto, y por otro al estado actual de la ciencia de datos en el mundo. Están presentados a continuación, junto a una breve explicación sobre su relación con el proyecto:

**Educación de calidad** Como hemos mencionado desde el principio, el objetivo de DHEM no es sólo proporcionar un soporte a git/DVC para usar un repositorio local en lugar de depender de almacenamiento en la nube, sino también proporcionar una herramienta intuitiva y fácil de utilizar para usuarios principiantes o estudiantes. En este sentido, las decisiones de diseño de la interfaz y limitaciones de funciones de git y DVC han ido siempre dirigidas a cumplir este objetivo, de forma que el uso del programa sirva como puente entre el conocimiento rudimentario que el usuario tenía previamente, y las habilidades prácticas que se derivan de un uso real de estos conocimientos teóricos. A través del uso de este programa como "iniciación", esperamos que más personas mantengan su interés por el mundo de la ciencia de datos y aprendan sobre este área.

**Trabajo decente y crecimiento económico** La expansión del uso de ciencia de datos en la mayoría de industrias la ha convertido en un componente esencial para muchas empresas, especialmente las más grandes, que disponen de bases de datos que explotar económicamente, ya sea de forma directa (vendiendo sus modelos como producto) como indirecta (utilizando estos modelos predictivos para potenciar otras áreas de negocio). Este tipo de programa contribuye a reducir la abstracción entre lo que un usuario normal considera la inteligencia artificial y lo que realmente es, potencialmente incrementando la cantidad de personas que se interesan por trabajar en este área. Estos trabajos, debido a la popularidad de la ciencia de datos mencionada anteriormente, están demandados y bien remunerados, lo que mejora las perspectivas laborales de esos trabajadores y contribuye satisfactoriamente a este ODS.

**Industria, innovación e infraestructuras** De la misma forma que el uso de este tipo de programas beneficia a los individuos que acceden de una forma más sencilla al mundo de la ciencia de datos, también beneficia a las empresas que buscan empleados con estos conocimientos. Incrementar el grupo de empleados en el mercado laboral con estas habilidades permitirá a las empresas aprovechar sus propios recursos de datos y potenciará el uso de la ciencia de datos en la industria, retroalimentando su popularidad y motivando el desarrollo de nuevas funcionalidades. Estos factores no sólo permitirán maximizar el rendimiento de los sistemas que ya se han desarrollado o se están desarrollando, sino que fomentará la creación de nuevas aplicaciones para estos recursos.

# APÉNDICE B
# Anexo 1

## B.1 Popularidad de plataformas de MLOps

La siguiente tabla proporciona datos sobre la popularidad en el motor de búsqueda Google de cada una de las plataformas de MLOps que comparamos en la sección 2.1. Para esto utilizamos Google Trends, que nos da una idea del interés del público general por estas plataformas. Google Trends no proporciona datos numéricos absolutos, sino que nos da una comparación relativa en la que el 100 es el interés máximo que ha habido en cualquiera de los 4 términos de búsqueda, y el resto de valores están en valor porcentual de este interés máximo (ocurrido en la semana del 11/12/2022 para Dataiku).

Cabe destacar que esta comparativa no es perfecta, y no debe tomarse como tal: al ser la empresa Iterative la más reciente de todas, no se puede seleccionar como "Empresa" al realizar la comparativa en Google Trends, como sí se puede hacer con Dataiku, Comet ML y Weights & Biases. Además, al ser "iterative" un término inglés real y común en el campo de la informática, los resultados se sesgan mucho. Es por esto que he utilizado en la comparativa "Iterative AI", de forma que la búsqueda sea más acotada.

| Semana | Dataiku | Iterative AI | Comet ML | Weights & Biases |
|---|---|---|---|---|
| 2022-07-17 | 84 | 2 | 2 | 9 |
| 2022-07-24 | 80 | 2 | 6 | 7 |
| 2022-07-31 | 71 | 2 | 5 | 13 |
| 2022-08-07 | 79 | 0 | 2 | 9 |
| 2022-08-14 | 69 | 0 | 2 | 4 |
| 2022-08-21 | 69 | 2 | 6 | 11 |
| 2022-08-28 | 64 | 3 | 1 | 17 |
| 2022-09-04 | 67 | 0 | 3 | 11 |
| 2022-09-11 | 71 | 2 | 2 | 7 |
| 2022-09-18 | 79 | 2 | 0 | 10 |
| 2022-09-25 | 79 | 5 | 3 | 3 |
| 2022-10-02 | 79 | 0 | 3 | 13 |
| 2022-10-09 | 78 | 0 | 4 | 13 |
| 2022-10-16 | 86 | 2 | 0 | 14 |
| 2022-10-23 | 76 | 2 | 3 | 12 |
| 2022-10-30 | 87 | 0 | 2 | 12 |
| 2022-11-06 | 75 | 2 | 4 | 6 |
| 2022-11-13 | 87 | 2 | 0 | 8 |
| 2022-11-20 | 69 | 5 | 1 | 5 |
| 2022-11-27 | 76 | 6 | 2 | 11 |
| 2022-12-04 | 81 | 5 | 3 | 12 |

| | | | | |
|---|---|---|---|---|
| 2022-12-11 | 100 | 5 | 3 | 8 |
| 2022-12-18 | 64 | 4 | 1 | 7 |
| 2022-12-25 | 38 | 0 | 0 | 6 |
| 2023-01-01 | 72 | 3 | 0 | 11 |
| 2023-01-08 | 76 | 4 | 5 | 12 |
| 2023-01-15 | 89 | 3 | 4 | 9 |
| 2023-01-22 | 91 | 0 | 4 | 12 |
| 2023-01-29 | 84 | 2 | 1 | 12 |
| 2023-02-05 | 95 | 2 | 0 | 10 |
| 2023-02-12 | 91 | 3 | 6 | 14 |
| 2023-02-19 | 80 | 3 | 5 | 12 |
| 2023-02-26 | 84 | 5 | 4 | 15 |
| 2023-03-05 | 86 | 2 | 4 | 12 |
| 2023-03-12 | 89 | 2 | 2 | 17 |
| 2023-03-19 | 86 | 4 | 4 | 12 |
| 2023-03-26 | 92 | 3 | 2 | 13 |
| 2023-04-02 | 80 | 0 | 3 | 13 |
| 2023-04-09 | 82 | 2 | 3 | 18 |
| 2023-04-16 | 86 | 4 | 4 | 16 |
| 2023-04-23 | 83 | 6 | 3 | 14 |
| 2023-04-30 | 80 | 4 | 3 | 13 |
| 2023-05-07 | 80 | 3 | 3 | 12 |
| 2023-05-14 | 76 | 0 | 2 | 17 |
| 2023-05-21 | 77 | 2 | 1 | 13 |
| 2023-05-28 | 80 | 2 | 5 | 16 |
| 2023-06-04 | 90 | 6 | 2 | 12 |
| 2023-06-11 | 90 | 4 | 6 | 10 |
| 2023-06-18 | 95 | 4 | 2 | 11 |
| 2023-06-25 | 91 | 4 | 2 | 17 |
| 2023-07-02 | 84 | 0 | 0 | 13 |
| 2023-07-09 | 99 | 6 | 2 | 11 |

**Tabla B.1:** Popularidad de plataformas de MLOps entre el 17/07/2022 y el 16/07/2023 (Google Trends, 2023)