UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

DESACOPLAMIENTO DE DATOS Y EXPERIMENTOS EN EL SOFTWARE DASHAI

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERA CIVIL EN COMPUTACIÓN

MILLARAY VALENTINA VALDIVIA SALDÍAS

PROFESOR GUÍA:
FELIPE BRAVO MÁRQUEZ

MIEMBROS DE LA COMISIÓN:
IVÁN SIPIRAN MENDOZA
SERGIO LEIVA VALDEBENITO

SANTIAGO DE CHILE
2024

Resumen

Hoy en día la Inteligencia Artificial se ha hecho presente en industrias muy variadas, por lo que distintos profesionales, dentro y fuera del área de ciencia de datos se están interesando en utilizar modelos de aprendizaje de máquinas para hacer uso sus beneficios.

En este contexto nace la idea de *DashAI*, un software de código abierto que busca democratizar el acceso al *Machine Learning*, eliminando las barreras de entrada para aquellos sin conocimientos de programación e Inteligencia Artificial.

DashAI es un proyecto en continuo desarrollo, cuyo equipo está conformado por memoristas, estudiantes e ingenieros, y que en su situación actual presenta limitaciones relacionadas al uso y conocimiento de los datos utilizados. Para poder subir un nuevo conjunto de datos el usuario debe elegir previamente que tarea quiere realizar con estos en los experimentos, esto sin la posibilidad de explorar qué tipo de datos contiene el archivo subido.

En *Machine Learning* se hace indispensable la exploración y preparación de los conjuntos de datos que se utilizarán en el entrenamiento de los modelos, por lo que se hace vital eliminar las limitaciones que impiden que el usuario pueda conocer y utilizar su conjunto de datos para distintos experimentos.

Esta memoria contempla eliminar los componentes que enlazan cada conjunto de datos a una tarea y entregar al usuario nuevas funcionalidades que le permitan visualizar el contenido de ellos, así poder aprovechar los datos que se tienen en mayor medida y entregar una experiencia más completa y cercana a un proyecto de *Machine Learning*.

Para concluir, el desacoplamiento de componentes se realizó con éxito durante la implementación de la solución, permitiendo subir conjuntos de datos y crear nuevos experimentos con mayor posibilidades de configuración, pero se establece que aún se deben implementar nuevos módulos que sirvan para preparar y preprocesar los datos dentro del *software DashAI* y que le aportarían mayor valor a los usuarios de este.

To be living legends.
-Ten Lee

Agradecimientos

Me gustaría agradecer en primera instancia a mi madre Lucía, que me ha acompañado y brindado apoyo durante toda mi vida y en especial en el trayecto de mi vida universitaria, gracias por no cuestionar mis decisiones y siempre ser el hombro que necesito.

También quiero agradecer a mi padre Juan, a mi pareja Gonzalo, a mis hermanos Noelia, Francisca, Benjamín, Maximiliano y Víctor, y a mis sobrinos Cristian, Violeta y Aylén, que me dan fuerzas todos los días para seguir adelante y forman parte de los motivos por los que hoy continuo avanzando en este proceso.

Agradezco a mis amigas Isidora, Isabel y Javiera que nunca me han dejado de lado y me dan el apoyo primordial siempre que lo busco, sin ustedes no podría haber completado esta etapa en mi vida. Y a mis amigos de la universidad, que desde el primer día en esta institución me han alegrado y acompañado en esta difícil etapa.

Finalmente, agradezco a Felipe, Cristian, Pablo, Rodrigo y todo el equipo de desarrollo de *DashAI* que me ha entregado grandes herramientas y conocimientos durante el proceso de mi memoria, ayudando a mi desarrollo como futura profesional.

Tabla de Contenido

Índice de Ilustraciones

Capítulo 1

Introducción

1.1. Contexto

La Inteligencia Artificial [17], y en particular el *Machine Learning* (o *ML* por sus siglas en inglés) [2], están teniendo un impacto cada vez mayor en diversos aspectos de la sociedad y para manipular los modelos de *ML* es necesario contar con un sólido conocimiento teórico y técnico, lo que puede dificultar el acceso a esta tecnología y limitar el avance en este campo.

Es evidente que existe una brecha en el acceso al *Machine Learning*, lo que pone barreras de entrada a personas que no tienen conocimientos de programación e Inteligencia Artificial. *DashAI*[1] se propone eliminar esta brecha al proporcionar una plataforma de código abierto donde personas con distintos niveles de conocimiento puedan experimentar con distintos modelos. La plataforma estará dotada de distintos modelos que permitirán al usuario realizar tareas específicas con los datos ingresados.

La plataforma de *DashAI* permite al usuario elegir el modelo más adecuado para resolver un problema específico sin la necesidad de manipular los modelos por sí solo. Además, el software permitirá la incorporación de nuevos datos de entrada que el usuario quiera clasificar, lo que brindará una experiencia completa y satisfactoria, por el momento la funcionalidad que permite predecir está aún en desarrollo. De esta manera, *DashAI* busca democratizar el acceso al *Machine Learning* y convertirse en una herramienta útil para la sociedad.

En ese sentido, para agregarle mayor valor a *DashAI* se necesita que la plataforma entregue mayor información y poder de manipulación de los conjuntos de datos que se utilizan para los entrenamientos. Esto contempla cambios en el flujo completo de la aplicación ya que la arquitectura previa de esta no lo permite.

La necesidad de crear esta nueva funcionalidad surgió a raíz de los comentarios y sugerencias de los usuarios durante un lanzamiento previo de la plataforma *DashAI*. Durante esta etapa de prueba, los usuarios manifestaron que sería extremadamente útil tener la capacidad de visualizar y explorar el contenido de los conjuntos de datos que subían a la plataforma.

[1] Repositorio DashAI: https://github.com/DashAISoftware/DashAI

Actualmente, los conjuntos de datos se cargan sin la posibilidad de comprobar su estado o examinar su contenido, lo que dificulta la comprensión de los datos utilizados en el entrenamiento de los modelos.

En la industria existen diversas herramientas de software que permiten simular entrenamientos de aprendizaje automático sin que los usuarios necesiten programar o tener un conocimiento amplio de las herramientas necesarias, como *WEKA*[2], *RapidMiner*[3], entre otros. Además, también hay aplicaciones que funcionan como estudios de datos y tienen funcionalidades para preparar y modificar conjuntos de datos tales como *OpenRefine*[4].

La ventaja de *DashAI* es que ofrece las funcionalidades de exploración de conjuntos de datos, entrenamiento y predicción en una sola plataforma con modelos de aprendizaje automático de estado del arte. Además, como es de código abierto, es altamente extensible y puede adaptarse a los grandes cambios en el campo del aprendizaje automático que se producen constantemente, asegurando que se mantenga relevante en el tiempo.

En resumen, *DashAI* se presenta como una plataforma de código abierto que busca democratizar el acceso al *Machine Learning*, eliminando las barreras de entrada para aquellos sin conocimientos de programación e Inteligencia Artificial. Con sus funcionalidades de experimentación de distintos modelos y métricas, predicción con nuevos datos y la inclusión de nuevas funcionalidades de exploración de *datasets*, que es el principal objetivo de esta Memoria, la plataforma ofrecerá a los usuarios la capacidad de experimentar con diferentes modelos y comprender mejor los datos utilizados en el entrenamiento.

Este primer Capítulo introductorio del documento contiene distintas secciones donde en primera instancia se habla contexto en el que se desarrolla *DashAI* y la situación en la que se encuentra previo a la memoria para orientar al lector en términos de los problemas que surgen con esta implementación.

Luego en la Sección de problema y relevancia se exponen las limitaciones que presenta la arquitectura e interfaz y porque esto afecta a un correcto ciclo de entrenamientos de *Machine Learning*.

A continuación, en la Sección de objetivos se habla las metas planteadas para poder solucionar los problemas y limitaciones presentes en el uso correcto de conjunto de datos en *DashAI* y que sirven de orientación para poder solucionarlos.

En la Sección de metodologías, se explica como el equipo de *DashAI* lleva a cabo el trabajo y su gestión para poder lograr una mejora continua del estado del *software*.

Posteriormente, se resume la solución a los problemas describiendo las etapas importantes en el desarrollo y que busca resolver cada una.

Finalmente, en la Sección de estructura del documento se detalla el contenido de cada Capítulo con una descripción del tema a tratar en cada uno.

[2]Documentación de *WEKA*: `https://www.cs.waikato.ac.nz/ml/weka/`
[3]Documentación de *RapidMiner*: `https://rapidminer.com/`
[4]Documentación de *OpenRefine*: `https://openrefine.org/`

1.2. Situación actual

La plataforma está compuesta por un *back-end* que consta de *end-points* que hacen llamadas a la *Application Programming Interface* o, por sus siglas, *API* utilizando la librería *FastAPI*[5] de *Python*[6], y de un *front-end* desarrollado mediante el *framework* de *React*[7]. Hasta el día de hoy la plataforma *DashAI* se encuentra en pleno desarrollo. Tanto el motor como el diseño de la interfaz están en continua evolución con el fin de lograr los objetivos del software. La interfaz gráfica actualmente no permite una interacción completa con el usuario como se espera ya que aún faltan los módulos de predicción que lograrán que se pueda clasificar con nuevos datos de entrada.

1.2.1. Componentes de la arquitectura

En cuanto a la arquitectura de *DashAI*, el *back-end* de la aplicación esta estructurada entorno a distintos componentes fundamentales para poder generar el proceso y toda la lógica que lleva a un usuario a poder entrenar nuevos experimentos creados. Entre otros, los componentes son:

- ***Datasets:*** Entidad que representa los conjuntos de datos, los que son extensiones de los conjuntos de datos de la librería *Datasets* de *Hugging Face*[8][9]. Guarda las configuraciones del conjunto de datos, las columnas que se utilizarán como entrada y salida a los modelos como las de salida, las configuraciones de las particiones que se utilizarán para entrenamiento, validación y testeo, entre otros.
- ***Experiments:*** Entidad que representa los experimentos creados por el usuario, guarda tanto los modelos que se quieren entrenar como los componentes relacionados, es decir, el conjunto de datos y la *Task* (que se definirá a continuación).
- ***Runs:*** Entidad que representa un proceso que corre un modelo de un experimento, contiene el estado del experimento y los resultados de las métricas para lo modelos seleccionados.

1.2.2. Componentes extensibles

Los componentes extensibles son los que pueden extenderse a más formas de realizar un proceso, por lo que en un futuro existirán más tipos implementados en la aplicación. Los componentes extensibles de *DashAI*, donde cada objeto es una clase de *Python*, son:

- ***Task:*** El usuario escoge la tarea que se quiere realizar, ejemplos de estas son:

[5]Documentación de *FastAPI*: `https://fastapi.tiangolo.com/`

[6]Documentación del lenguaje de programación *Python*: `https://www.python.org/`

[7]Documentación de *React*: `https://react.dev/`

[8]Documentación de la librería *Datasets*: `https://huggingface.co/docs/datasets/index`

[9]La librería Dataset de HuggingFace soluciona la compatibilidad de los conjuntos de datos con herramientas como NumPy, pandas, PyTorch, Tensorflow 2 y JAX para que estos puedan ser utilizados y manipulados.

- *Tabular Classification Task*: Predice las categorías de los datos organizados de forma tabular.
- *Text Classification Task*: *Natural Language Processing[4] Task* que asigna automáticamente categorías predefinidas a los textos de acuerdo a su contenido.
- *Translation Task*: *Natural Language Processing Task* que convierte el texto de un idioma a otro.
- *Image Classification Task*: Clasifica imágenes de acuerdo a los modelos.

Dependiendo de la *Task* se definen las columnas que los modelos utilizan como entrada y las columnas de salida y sus respectivas cardinalidades, utilizando los tipos disponibles de la librería *Dataset* de *HuggingFace* para que luego sea posible mapear la entrada y salida con los modelos de *Machine learning* supervisados que están implementados en *DashAI*.

- ***DataLoader:*** Corresponden a objetos responsables de recibir como entrada los datos en distintos formatos y transformarlos a la representación interna de *DashAI*. Los *DataLoaders* dependen de la tarea que se esté intentando resolver y por ende son compatibles con una o más de ellas. Por el momento, *DashAI* tiene disponible los siguientes *DataLoaders*:
  - *CSV DataLoader*: Acepta archivos *Comma Separated Values* o, por sus siglas, *CSV*.
  - *JSON*[10] *DataLoader*: Acepta archivos *JSON*.
  - *Image DataLoader*: Acepta archivos de imagen.
- **Modelos:** Son los objetos que encapsulan modelos que definen como entrenar y predecir de acuerdo a las *Tasks* que se buscan trabajar y con la que los modelos son compatibles.
- **Métricas:** Son los objetos que encapsulan métricas que evalúan el rendimiento de los modelos. Las *Tasks* contienen métricas que se ejecutan cuando se evalúan, registrando sus resultados.

Con la arquitectura descrita, los *DataLoader* dependen de la *Task* elegida por el usuario. Por ejemplo, si se requiere realizar un experimento de clasificación tabular, el usuario debe elegir *Tabular Classification* de entre toda la lista de *Tasks* disponibles. Luego tiene que elegir el *DataLoader*, pero solo se le entregará una lista con los que son compatibles con la tarea que escogió, es decir, para *Tabular Classification* podrá elegir entre *CSV y JSON*. Finalmente, deberá subir un conjunto de datos que sea compatible con ese *DataLoader* en específico.

Al subir un nuevo conjunto de datos, el usuario debe presionar el botón *'New Dataset'* ubicado en la esquina superior derecha de la lista de *datasets* que se encuentra en la Figura 1.1. Luego, como se muestra en la Figura 1.2, se despliega una vista que pide al usuario seleccionar la *Task*. Y finalmente, al configurar el conjunto de datos no es posible observar cómo es el conjunto de datos que el usuario quiere subir, lo que se ve en la Figura 1.3 que contiene la vista al subir el conjunto de datos en el flujo actual.

[10]Documentación de *JSON*: `https://www.json.org/json-es.html`

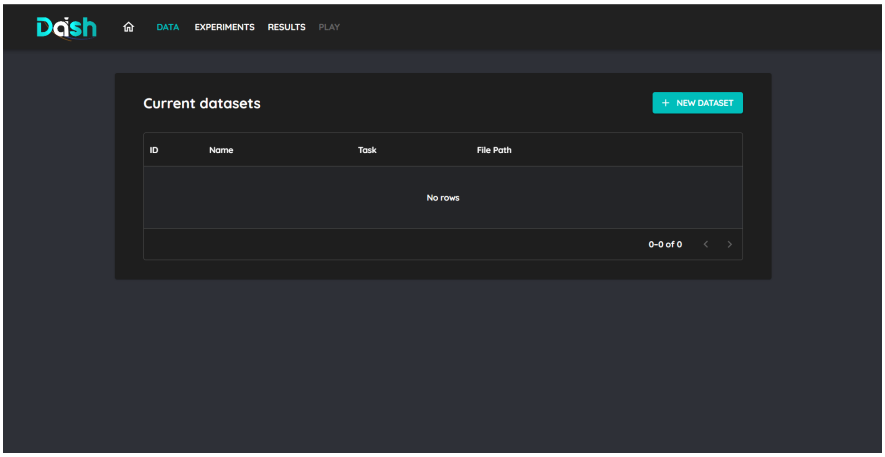

Figura 1.1: Captura de la vista de *Datasets*.

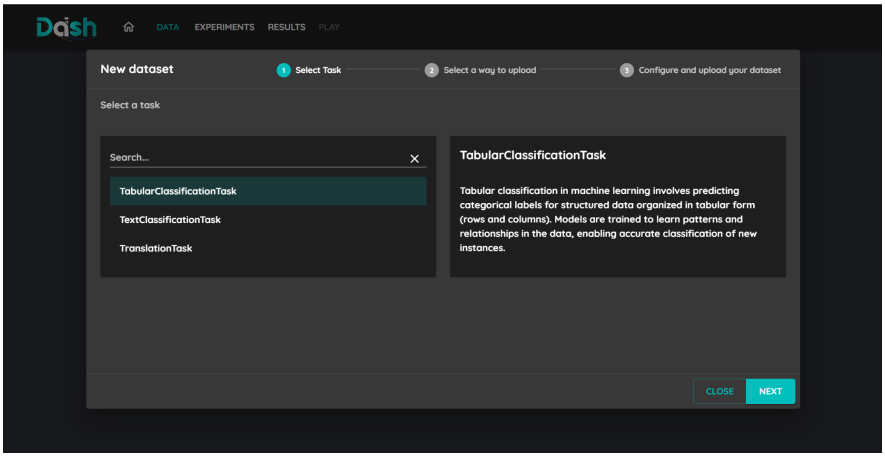

Figura 1.2: Captura del flujo con elección de *Task*.

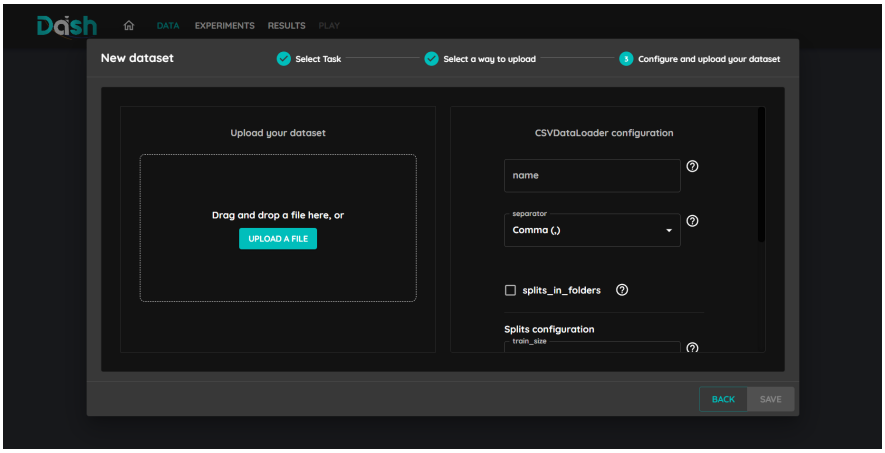

Figura 1.3: Captura del flujo al cargar un conjunto de datos.

Se describirá con mayor profundidad el flujo que tiene que realizar el usuario para subir conjuntos de datos y preparar experimentos en el Capítulo 3, Sección de Flujo previo 3.1.

1.3. Problema y relevancia

Los *data scientists* en sus proyectos realizan distintos procesos que los llevan al modelamiento de operaciones de *Machine Learning*, *data mining*, entre otros. En esta Sección primero se ahondará en por qué la etapa de exploración y procesamiento de datos es importante para lograr buenos modelos y cómo la situación previa del software de *DashAI* no permite aprovechar los datos recolectados gracias a su arquitectura, lo que va en contra el objetivo fundamental de *DashAI* de entregar una herramienta completa para acercar el *Machine Learning* tanto a *data scientists* y *data engineers* como a usuarios que no conocen de los conceptos básicos de este.

Los *data scientists* buscan generar el mejor modelo que resuelvan las tareas que requieren y para esto se basan y siguen distintos procesos que han sido estudiados y que ayudan a los científicos a agilizar sus tareas.

Un ejemplo de lo anterior es modelo *Cross Industry Standard Process for Data Mining* o, por sus siglas, *CRISP DM* [10], que bastante usado para minería de datos. *CRISP DM* es un modelo de que sirve como base para los procesos de ciencias de datos y que sigue las fases mostradas en la Figura 1.4.

Según este modelo una de las etapas más importantes en la minería de datos ya que consume alrededor del 80 % del tiempo de los procesos de *data science* es la preparación de los datos, donde se preparan y organizan los conjuntos de datos para su posterior modelamiento. Además, cabe recalcar que es común que durante su recolección el *data scientist* utilice conjuntos de datos que ya utilizó con anterioridad.

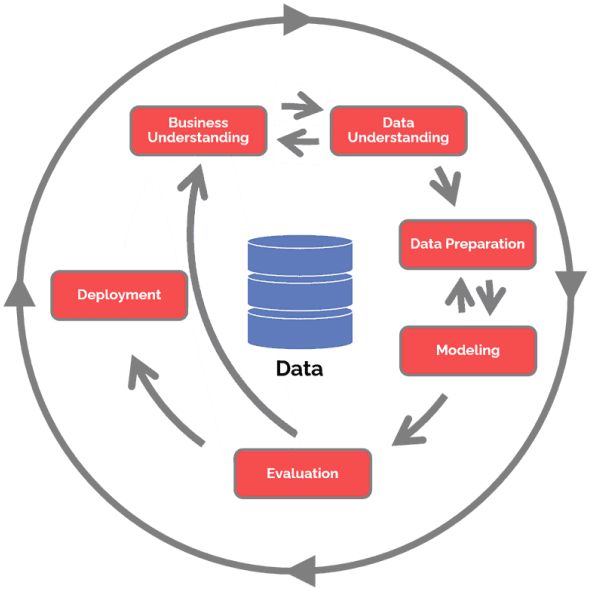

Figura 1.4: Diagrama de *CRISP DM* obtenido desde [10].

Esta etapa consiste en seleccionar los datos a utilizar, limpiarlos en caso de que existan datos basura, construir y crear nuevos conjuntos de datos y darles el formato necesario para realizar operaciones con ellos. Aquí es muy importante chequear que los datos usados estén

en la forma correcta y sean válidos para poder procesarlos.

Otro ejemplo de procesos en *data science* que podemos tomar en cuenta son específicos de *Machine Learning* donde también existen modelos de procesos como el descrito para minería de datos. Los *Machine Learning Operations* o *MLOps*[20] son un componente crítico de todo proyecto exitoso de esta área ya que ayuda a la organización, generando valor y reduciendo el riesgo asociado.

Solo un 10 % de las etapas en los procesos de los proyectos de *Machine Learning* consiste en entrenar código[6] y el resto son distintas etapas de un *pipeline* que busca agilizar la creación de los modelos. Según estos *pipelines* puede ser incluso mejor volver a la etapa de preparación de datos, donde se divide el conjunto de datos entre particiones de entrenamiento, validación y testeo para verificar que la utilizada sea la que entrega resultados óptimos.

Existen varias versiones de los *pipelines* que se siguen en los proyectos básicos, uno común y básico es el presentado en la Figura 1.5, donde se encuentra destacada la etapa de desarrollo de los modelos. Específicamente, *DashAI* es un proyecto centrado en el entrenamiento de modelos de *ML* por lo que es competente centrarse en la primera de estas etapas.

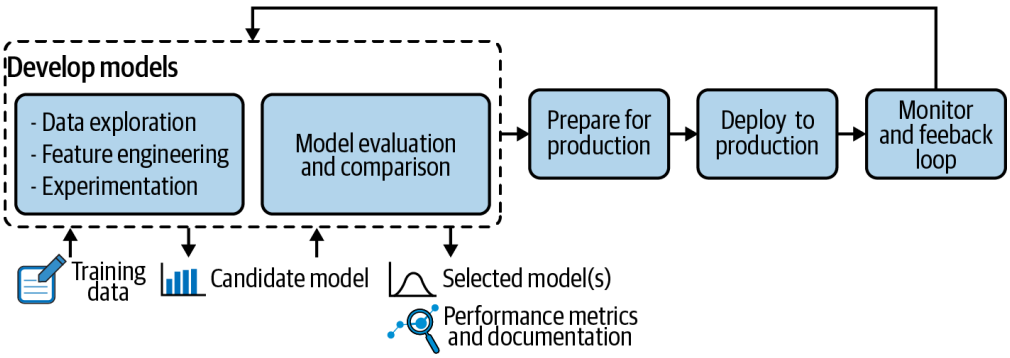

Figura 1.5: Modelo de desarrollo de un proyecto de ML. Obtenido desde [20].

En la Figura presentada se puede ver que uno de los componentes de este *pipeline* es el entrenamiento de los datos, que contempla dentro de esta etapa la exploración e ingeniería de los datos recolectados. Al igual que el ejemplo presentado del modelo *CRISP DM*, el conocimiento y manejo de datos se vuelve una pieza clave para el buen entrenamiento y encontrar buenos candidatos de modelos.

Con lo descrito, queda claro que las fases tempranas de los proyectos de ciencia de datos son generalmente acerca de entender los datos que se tienen y procesarlos. Estas etapas son muy importantes ya que contribuyen a agilizar sus procesos y a generar mejores modelos.

Como *DashAI* busca ser una herramienta de ayuda que acerque a usuarios de distintas áreas al *Machine Learning*, es importante que pueda entregar a los usuarios funcionalidades para poder preparar los conjuntos de datos que sube a la plataforma.

En el estado previo de *DashAI* esto no era posible ya que el flujo solo permite usar el conjunto de datos como se subió y no deja explorarlo ni modificarlo. Además, debido a su arquitectura, donde los conjuntos de datos ya tienen una tarea a realizar asociada, es imposible aprovechar al máximo los datos entregados por el usuario.

Específicamente, el flujo en la plataforma, previo a las implementaciones hechas en la memoria, presenta las siguientes limitaciones:

1. **La interfaz de *DashAI* no permite que el usuario conozca los datos que va a utilizar para los experimentos.**

   - Previa a la carga de los datos, el usuario debe conocer el conjunto de datos antes de subirlo, y debe saber si este es concuerda con lo requerido para que sea utilizado por los modelos que se encuentran implementados para cada *Task*, incluso si la interfaz no provee mucha información de esto.

     En caso de que el conjunto de datos subido no sirva para la *Task* elegida entonces el proceso se cae y el usuario no obtiene mayor retroalimentación de la causa del problema. Lo cual afecta a la usabilidad de la plataforma y entorpece los procesos de experimentación que *DashAI* tiene como finalidad facilitar.

   - Al subir un conjunto de datos luego de elegir la tarea no hay ninguna forma de que el usuario sepa si el conjunto de datos se ha subido bien y si este era el que contenía los datos necesarios

     Por ejemplo, si el conjunto de datos contiene unos distintos a los que el usuario esperaba, no se daría cuenta de esta situación y podría eventualmente utilizar datos erróneos para los entrenamientos, obteniendo resultados no útiles.

   - No se validan los tipos de las columnas con las que soporta la tarea elegida en ningún momento del flujo, si los tipos no corresponden el entrenamiento se cae.

   De acuerdo a las fases de los procesos comentadas con anterioridad en esta Sección, queda claro que el científico de datos se tiene que asegurar que los datos que va a utilizar sean los correctos y que estos sean válidos. Como fue detallado anteriormente, en este caso, se anula completamente la posibilidad de que el usuario si quiera explore sus datos en la misma plataforma, que en *MLOps* es una etapa básica para poder entender los datos.

2. **La plataforma, contemplando su arquitectura e interfaz, no permite que el usuario pueda aprovechar al máximo los datos subidos.**

   - Los conjuntos de datos se pueden utilizar solo para una tarea en particular luego de subirlos a la plataforma, ya que para subir uno nuevo primero se debe elegir qué tarea se quiere realizar de modo que esta queda guardada en la base de datos como un atributo del *Dataset*.

     Esto limita al usuario a hacer uso de un conjunto de datos para distintos tipos de experimentos, si quisiera realizar otra tarea tendría que subir una copia del conjunto de datos y asignarle la nueva *Task*.

   - No se puede elegir las columnas de entrada y salida para cada experimento, siempre que se entrene la última columna será la de la clase y el resto serán columnas de entrada.

     Al igual que el problema anterior si el usuario quisiera entrenar modelos haciendo uso de diferentes combinaciones de las columnas, o contiene datos con más de una columna de clase, no es posible hacerlo con el mismo conjunto de datos subido a *DashAI*.

Generalmente la etapa de recolección y preparación de datos consume la mayor parte del tiempo empleado en la creación de modelos, por lo que no poder utilizar un mismo conjunto de datos para lograr diferentes tareas y crear múltiples experimentos con distintas configuraciones supone en un retraso de las otras tareas del modelamiento.

1.4. Objetivos

Los objetivos descritos a continuación tienen la finalidad de resolver las problemáticas presentadas en la Sección anterior acerca de la subida, manejo y uso de los datos dentro la plataforma. La Sección se divide en el objetivo general de la memoria y objetivos específicos que servirán como guía para llegar al diseño de la solución.

1.4.1. Objetivo General

El objetivo general es permitir al usuario subir conjuntos de datos y comprender que tipo de datos tiene, sin tener que acoplar una *Task* al *Dataset* en ese momento. Además, dar la posibilidad de crear nuevos experimentos con el mismo conjunto de datos pero dándole distintas configuraciones. En resumen, el propósito es permitir una gestión independiente y eficiente de los datos, brindando al usuario creación de experimentos sin la necesidad de estar ligado a una tarea específica desde el inicio.

1.4.2. Objetivos Específicos

Considerando el objetivo general planteado anteriormente, se proponen los siguientes objetivos específicos:

1. **Desacoplamiento de conjuntos de datos y tareas:**
    - Permitir la subida de conjuntos de datos independientes de una tarea específica.
    - Implementar un nuevo flujo que permita asociar los datos con tareas posteriormente, en lugar de requerir la elección de una tarea al subir el conjunto de datos.
2. **Mejora en la retroalimentación del proceso de carga de datos:**
    - Desarrollar un mecanismo que proporcione retroalimentación inmediata al usuario al subir un conjunto de datos.
3. **Flexibilidad en la definición de columnas de input y output:**
    - Permitir al usuario seleccionar las columnas de input y output para cada experimento, brindando la opción de definir la columna de clase y las columnas de input de manera personalizada.
4. **Validación de tipos de columnas según la tarea seleccionada:**

- Implementar un sistema de validación de tipos de columnas que verifique la correspondencia con los requisitos de la tarea elegida en cada etapa del flujo, evitando errores en el entrenamiento debido a inconsistencias en los tipos de datos.

5. **Refactorización del código para soportar la separación de tareas y conjuntos de datos:**
   - Revisar y modificar la estructura del código existente en *DashAI* para asegurar la viabilidad técnica de los cambios propuestos.
   - Garantizar la compatibilidad y funcionalidad integral del software después de los cambios, asegurando que los usuarios puedan subir *Datasets* y entrenar modelos de manera efectiva en cada experimento, independientemente de la tarea asociada.

1.5. Metodologías de desarrollo

Para poder diseñar e implementar la solución se necesita utilizar metodologías de desarrollo que potencien el tiempo utilizado, agilizando las distintas etapas de desarrollo y mejorando continuamente los resultados obtenidos. En el proyecto actualmente se hace uso de la metodología ágil llamada *Scrum*[18]. En esta Sección se listan algunos elementos de esta y como son incorporados en *DashAI*.

1.5.1. Desarrollo incremental e iterativo

La metodología *Scrum* se enfoca en desacoplar proyectos grandes y complejos en tareas o hitos pequeños, que se desarrollan de forma independiente y en donde cada hito le añade valor al proyecto.

En *DashAI* se hace uso de un *Kanban*[11] para hacer seguimiento al avance de las tareas y así se asigna prioridad a cada una de ellas. Cada tarea se liga a una *Pull Request* cuando está en etapa de revisión lo que agiliza el proceso de lograr objetivos pequeños.

Este elemento además le añade flexibilidad y disminución de riesgos a la incorporación de nuevas ideas, ya que al separar en objetivos específicos se puede disminuir el riesgo de divergir con la rama original, y en caso de haber inconsistencias es más fácil poder arreglar los errores que surjan.

1.5.2. Reuniones periódicas

En metodologías ágiles y en particular en *Scrum* se hace uso de reuniones cortas diarias para compartir el avance en las tareas a realizar. El equipo de desarrollo del proyecto lo incorpora, en ellas cada integrante del equipo de desarrollo habla sobre su progreso y los

[11] Un *Kanban* es un método visual para gestionar proyectos donde se anota en que estado de desarrollo se encuentra la tarea a completar.

pasos a seguir durante el día, al igual que posibles problemas que esté enfrentando y en los que requiera ayuda del resto del equipo.

Además, *Scrum* contiene el elemento de iteraciones o *sprints*, que son sucesos cada 1-4 semanas, donde se dividen los procesos a realizar y se elijen los próximos pasos a seguir. *DashAI* implementa reuniones semanales donde cada memorista o tesista comparte sus avances, también se discuten ideas a implementar o problemas mayores que se presenten durante el desarrollo y que se deban evaluar con todo el equipo.

1.5.3. Discusiones acerca del diseño

Es necesario que cada proyecto se alinee con las necesidades del usuario, con la metodología incorporada es esperable que se realicen reuniones tempranas donde se comente el diseño a implementar y como esto contribuye al avance del proyecto.

Al iniciar el desarrollo de una nueva característica, en caso de que sea *front-end* se presentan *mock-ups* o maqueta del diseño de la vista que se va a implementar y esta se discute en las reuniones diarias con el resto del equipo. Durante estas discusiones pueden surgir recomendaciones o comentarios al diseño que aporten usabilidad o estilos y que son útiles para lograr la finalidad de la vista.

En el caso de que la nueva característica necesite código en el *back-end* de la aplicación también hay ocasiones que se presentan diagramas de flujo con el proceso a implementar, considerando los métodos utilizados y llamadas a la *API*. Esto también es comentado con el equipo de desarrolladores para que finalmente se pueda implementar.

1.5.4. Desarrollo y evaluación del código

El sistema de control de versiones *Git*[12] se basa en ramas del código donde se pueden postear avances por medio de *commits*. Cada nueva característica a integrar en *DashAI* se debe implementar en una nueva rama y por cada *commit* es necesario que este pase los revisores de estilo y testeo integrados en el proyecto.

Complementando lo anterior, para validar e integrar a *DashAI* cualquier avance en el desarrollo se utiliza el sistema de *Pull Requests* integrado en la plataforma de *GitHub*. Con cada nueva característica, formateo al código y/o integración de nueva herramienta, se realiza un *Pull Requests* desde la rama de trabajo a la rama principal del repositorio, la que es evaluada por ingenieros del equipo. Si los cambios son aprobados por el equipo entonces se integra a la rama principal de *develop*, sino se deben revisar los cambios sugeridos y se vuelve a evaluar.

[12]Documentación de Git: `https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control`

1.5.5. Mejoramiento continuo

Por último, es importante destacar que la metodología aplicada se centra en el concepto de mejora constante tanto a nivel individual como en el desarrollo del proyecto en sí. Cada iteración y reunión planificada se consideran como puntos cruciales para perfeccionar no solo el producto final, sino también las habilidades y procesos internos del equipo.

Estas instancias proporcionan oportunidades valiosas para llevar a cabo discusiones significativas y recibir comentarios constructivos. De esta manera, cada miembro del equipo puede aprender de las experiencias pasadas, identificar áreas de mejora y aplicar ajustes relevantes para elevar la calidad del proyecto y el rendimiento del equipo en su conjunto.

1.6. Descripción general de la solución

Para lograr los objetivos planteados se desacoplará la *Task* de los *DataLoaders*, modificando la base de datos para que la tarea especifica a realizar no sea parte de los *Datasets* sino de cada experimento. Así el usuario puede subir su conjunto de datos a la plataforma y luego al configurar los experimentos elegir qué *Task* es la que se desea implementar en ese Experimento en particular.

La solución se divide tres distintas etapas:

- **Cambios en el flujo de la carga de conjuntos de datos:** Se elimina el paso de elegir la *Task* en *New Dataset* y se añade una vista de resumen del *Dataset* que permite al usuario cambiar los tipos de las columnas que fueron colocados por defecto.
- **Cambios en el flujo de la configuración de experimentos:** Se agrega el paso de elegir la *Task* y uno donde el usuario puede elegir que columnas quiere de entrada y cuales para salida, además de cambiar los *splits* de entrenamiento, testeo y validación. Para luego validar las columnas elegidas con la *Task* asociada y poder guardar y configurar distintos Experimentos con el mismo conjunto de datos pero con distinta configuración.
- **Refactorización:** Para hacer efectivos los cambios en el flujo de la aplicación se debe cambiar la definición de múltiples métodos en el *back-end*, ya que utilizaban de alguna forma la definición de columnas que se cambia. Por lo que se mueve la validación de columnas al Experimento, se elimina la *metadata* de los *Datsets* con las columnas, y se modifican los Modelos existentes para que puedan utilizar los datos guardados en *Experiments* al momento de entrenar.

Cada etapa se encuentra detallada y justificada en la Sección 3 que aborda la solución en su totalidad, explicando como estas ayudan a lograr los objetivos y resolver los problemas planteados anteriormente.

1.7. Estructura del documento

Este informe cuenta con 5 capítulos que contextualizan y exponen el trabajo de la memorista.

El Capítulo 1 habla del contexto en el que es creado el proyecto de *DashAI* y, específicamente, qué problema presente en la plataforma es el que aborda la memorista. Además, define los objetivos y presenta un resumen de la solución implementada para alcanzarlos.

El Capítulo 2 extiende este contexto listando *softwares* y soluciones existentes que se asimilan a *DashAI* y/o que buscan resolver las mismas problemáticas. También habla de las herramientas que utiliza el equipo de *DashAI* para solucionar el problema que buscan abordar detallando las tecnologías incluidas en el proyecto.

El Capítulo 3 detalla las funcionalidades desarrolladas por la memorista. Para esto primero explica el estado previo de la plataforma con fines comparativos y luego el diseño del nuevo flujo ideado y su posterior implementación.

El Capítulo 4 aborda la posterior validación de los cambios implementados, incluyendo como se testean las implementaciones y como, desde la perspectiva de un usuario, se sigue el nuevo flujo en la plataforma.

Finalmente, el Capítulo 5 incluye las reflexiones y conclusiones que deja el trabajo realizado, junto con futuras mejoras, modificaciones y nuevas etapas a incorporar.

Capítulo 2

Estado del Arte

El Capítulo a continuación tiene como objetivo mostrar aplicaciones y *frameworks* ya existentes que buscan solucionar problemas similares o relacionados al de esta memoria, comparándolos con la situación actual de *DashAI*. También contextualiza acerca de qué librerías y herramientas son las utilizadas en el desarrollo de la plataforma y que aportan al entendimiento de la solución.

2.1. Soluciones existentes

DashAI está inspirado en otros *softwares* similares orientados a modelosde *Machine Learning* que permiten a los usuarios trabajar con ellos sin la necesidad de programar. En el ámbito del problema que se busca solucionar acerca de la nula exploración de los conjuntos de datos que se suben a la interfaz de *DashAI*, existen alternativas que ofrecen preparar y transformar los conjuntos de datos para entrenarlos y que contienen distintas funcionalidades para modificar los datos y su estructura pero que no están en el mismo contexto de *DashAI*, donde se busca generar experimentos con distintos modelos y métricas.

2.1.1. *RStudio* y *RapidMiner*

RStudio[1] es un *software* tipo IDE que permite trabajar con conjuntos de datos con lenguaje *R*[2] y que está principalmente orientado a la estadística. En *RStudio* se pueden preparar los datos y generar visualizaciones gracias a distintas librerías gracias a su característica de código abierto.

En la interfaz, al cargar un archivo es posible obtener vista previa, elegir parámetros y configurarlos, tales como la separación, codificación, valores nulos, entre otros, como se detalla en la Figura 2.1. Esta funcionalidad permite al usuario de la aplicación saber de antemano

[1]Documentación de *RStudio*: `https://posit.co/download/rstudio-desktop/`
[2]Documentación del lenguaje de programación *R*: `https://www.r-project.org/other-docs.html`

como se comporta el conjunto de datos y configurarlo para que su manipulación futura sea más fácil.

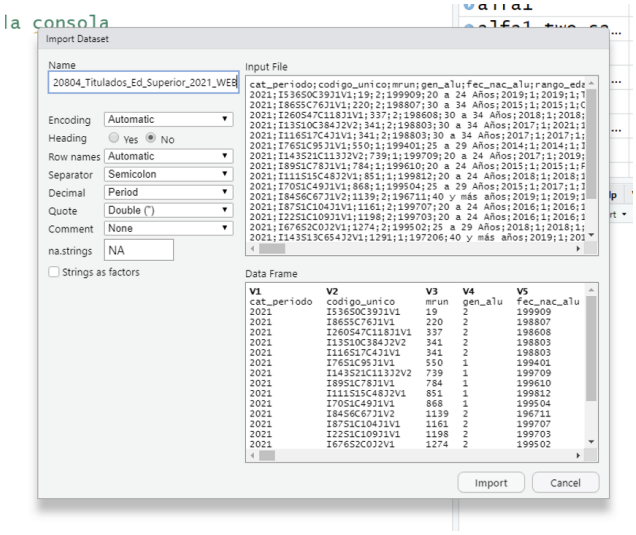

Figura 2.1: Captura de la vista de *RStudio* al configurar un conjunto de datos.

RapidMiner es otra aplicación que sirve para generar gráficos y prototipos con la finalidad de analizar los conjuntos de datos. Al igual de *RStudio*, *RapidMiner* presenta una vista de la data tabular que se puede transformar eliminando columnas, además de una ventana que permite conocer el tipo y estadísticas de cada una de ellas, como se aprecia en la Figura 2.3.

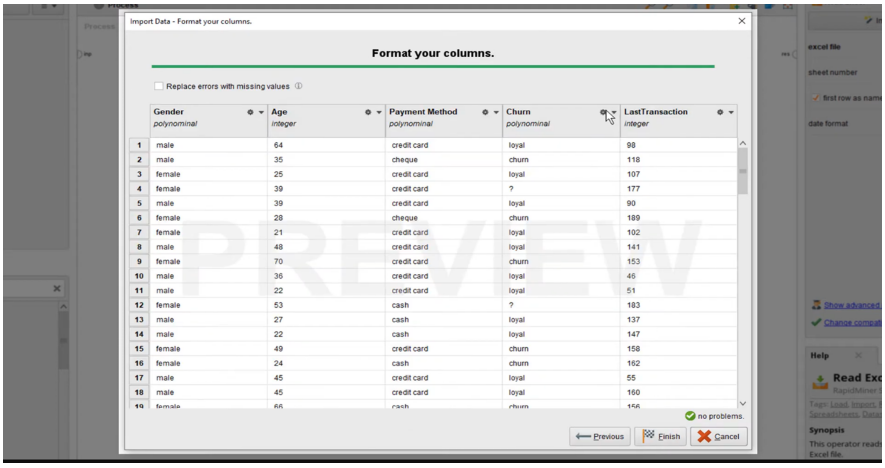

Figura 2.2: Captura de *RapidMiner* con la vista para modificar las columnas.

Una vista que permita al usuario conocer su conjunto de datos antes de ser guardado y utilizado para experimentos no está disponible actualmente en *DashAI*, cada vez que un usuario quiere usar un set de datos con los *DataLoader* de *JSON* y *CSV*, estos tienen que estar previamente conocidos y en caso de introducir datos que no sean compatibles con el *DataLoader* escogido este no es aceptado. Por lo que si un usuario quisiera poder ver y transformar sus datos tiene que usar otros *softwares* previamente.

Cabe destacar que las aplicaciones mencionadas están concentradas en estadística a diferencia de *DashAI* que está enfocado en *Machine Learning* y su experimentación con modelos.

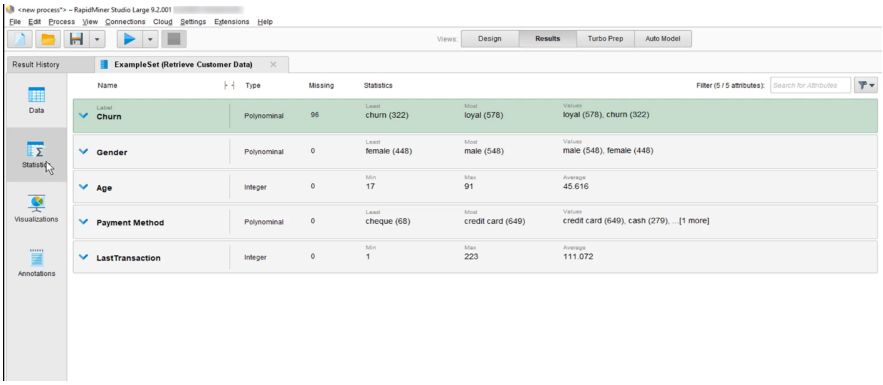

Figura 2.3: Captura de *RapidMiner* con el resumen de las columnas.

Pero para ambos se requiere este tipo de módulos para comprender los datos que se están utilizando.

2.1.2. *OpenRefine*

OpenRefine es una aplicación *software* libre actualmente desarrollado por Google que sirve para limpiar y transformar datos, a este proceso se le conoce como *Data Wrangling*[14] y/o *Data Cleaner*[15].

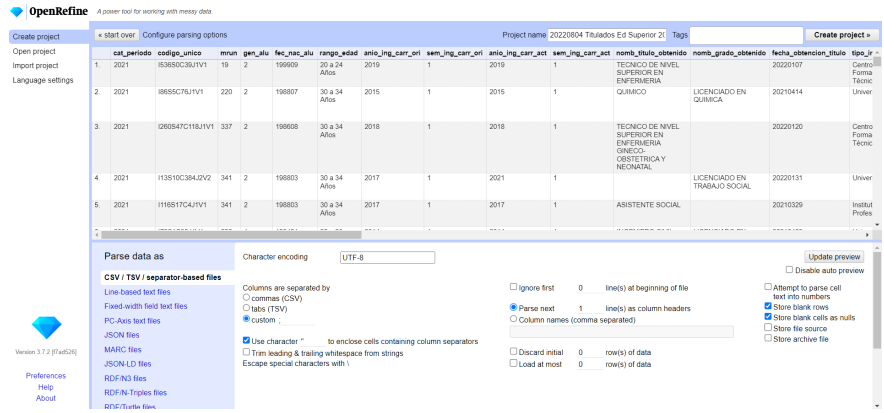

Figura 2.4: Captura de *OpenRefine* que contiene la vista al cargar un nuevo conjunto de datos.

En la aplicación se puede subir un archivo *CSV*, tener una vista de todas las columnas de la data tabular, editar columnas y sus nombres, borrar valores nulos, entre otros. A diferencia de *RStudio* este módulo permite realizar distintas transformaciones en los conjuntos de datos, como modificar los valores *string* y cambiar los tipos como muestra la Figura 2.5 y renombrar, eliminar o mover las columnas, como muestra se la Figura 2.6.

Antes de la realización de la memoria, *DashAI* no contiene ninguna vista previa de los *Datasets*. Los objetivos que se requieren implementar contienen funcionalidades que *OpenRefine* ya implementa, pero en *DashAI* se entregaría la ventaja de que estaría todo en un mismo *software*, tanto el entrenamiento como la preparación de los conjuntos de datos a utilizar.

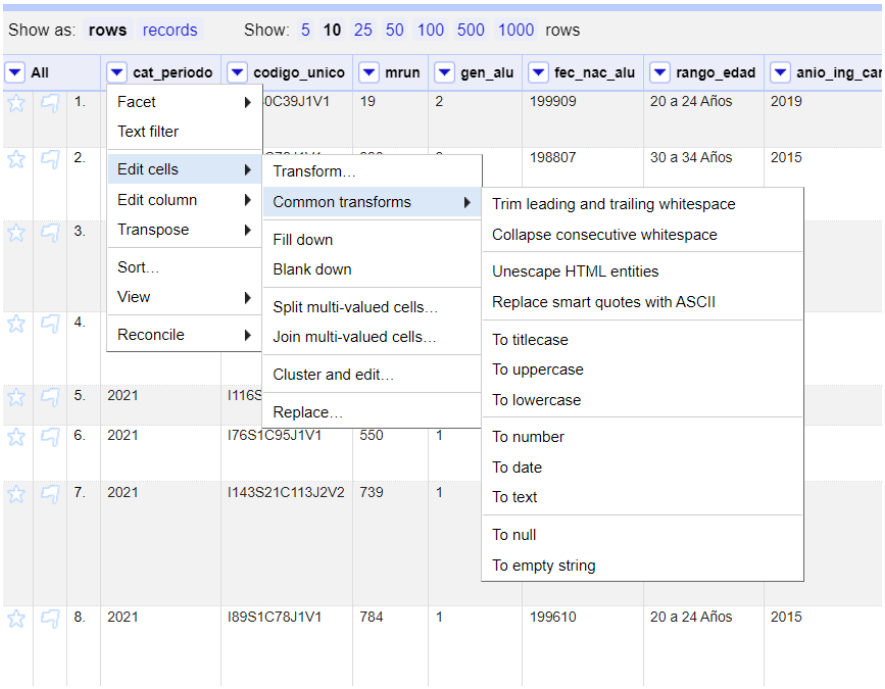

Figura 2.5: Captura de *OpenRefine* con opciones de edición.

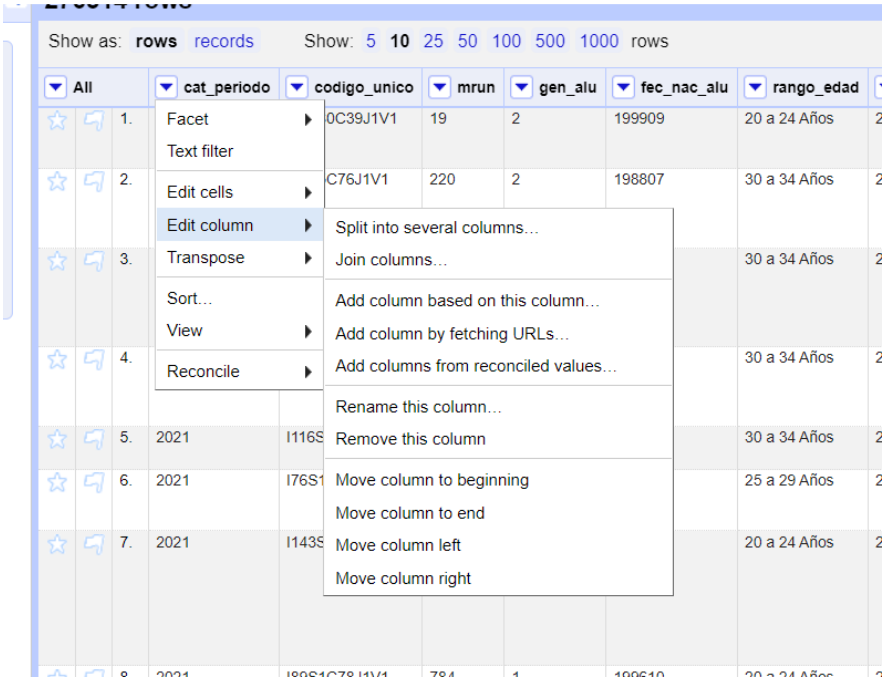

Figura 2.6: Captura de *OpenRefine* con opciones de transformación.

2.1.3. *WEKA*

WEKA es una aplicación desarrollada en la Universidad de Waikato para la minería de datos descrita en JAVA, que contiene distintos módulos para la preparación y clasificación de datos. La plataforma contiene distintas formas de visualizar la data tabular para que el usuario logre entender por medio de estadísticas su conjunto de datos.

Como se observa en la Figura 2.7, *WEKA* genera gráficos para ver las frecuencias de cada categoría o los clústers que se generan como en la Figura 2.8.

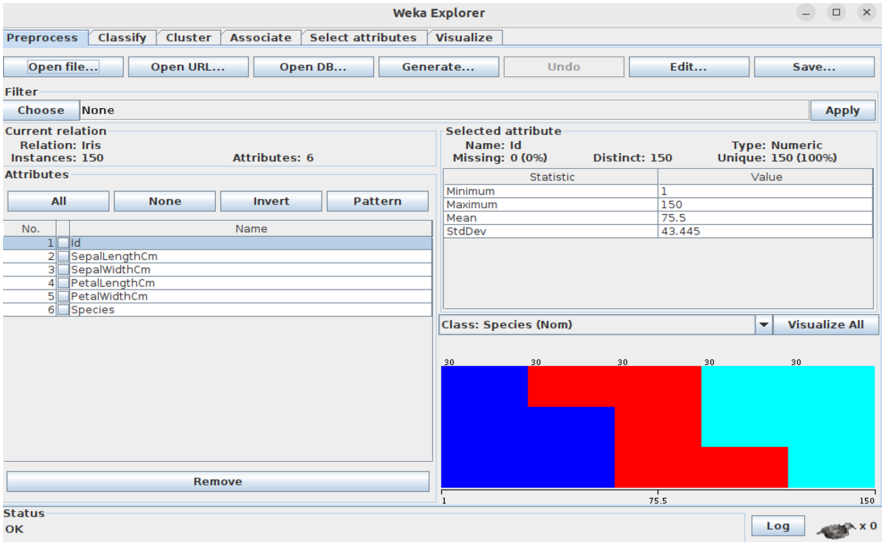

Figura 2.7: Captura de *WEKA*.

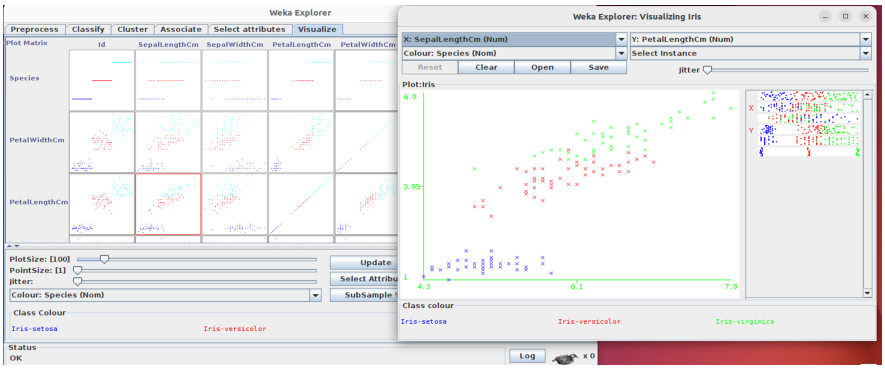

Figura 2.8: Captura de *WEKA*.

Debido a que la aplicación está en vigencia hace años, a día de hoy presenta algunas limitaciones frente al avanzado desarrollo de la Inteligencia Artificial. *WEKA* está programada en *JAVA* que era el lenguaje usado para *Machine Learning* en ese tiempo debido a su capacidad para manejar grandes conjuntos de datos. Actualmente esto limita las implementaciones ya que las principales herramientas para el aprendizaje de máquinas no son compatibles y están implementadas con *Python*.

2.1.4. *Vertex AI*

Vertex AI[3] es una herramienta de Google Cloud que sirve para entrenar y probar modelos de aprendizaje automático. Está enfocada en Inteligencia Artificial y contiene un módulo de visualización previa de los datos, como se observa en la Figura 2.10, al subir un conjunto de datos de imágenes se muestran ejemplos de ellas.

[3]Documentación de *Vertex AI*: `https://cloud.google.com/vertex-ai?hl=es-419`

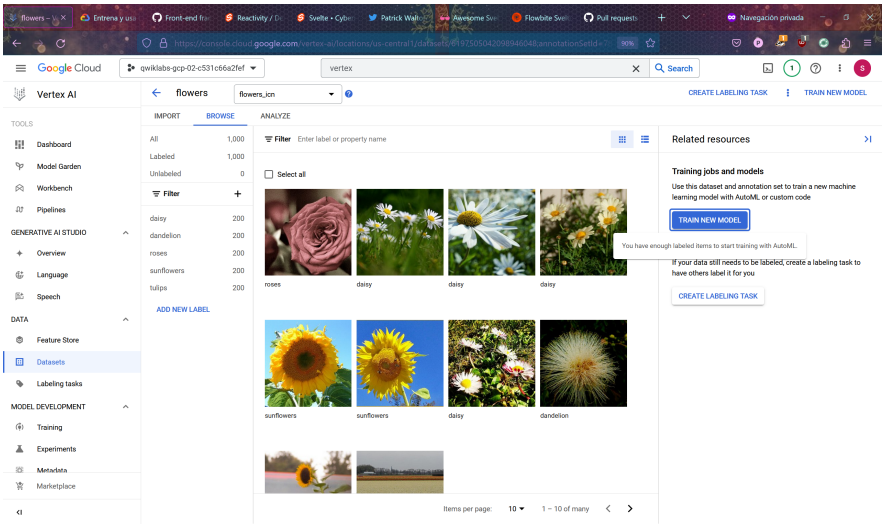

Figura 2.9: Captura de *VertexAI* con vista previa de imágenes aleatorias.

En resumen, se pueden obtener las siguientes comparaciones entre los *softwares* presentados y el estado en que se encuentra actualmente *DashAI*:

Característica del Software	RStudio	RapidMiner	OpenRefine	WEKA	Vertex AI	DashAI
Open-Source	Sí	Sí	Sí	Sí	No	Sí
Contiene herramientas de pre-visualización y/o pre-procesamiento de datos	Sí	Sí	Sí	Sí	Sí	No
Centrado en entrenar modelos de Machine Learning	No	No	No	Sí	Sí	Sí
Modelos en el estado del arte	-	-	-	No	Sí	Sí
Entrenamiento sin codificar	-	-	-	Sí	Sí	Sí

Figura 2.10: Tabla comparativa entre *softwares* existentes y *DashAI*.

Si bien las integraciones que se realicen a *DashAI* en esta memoria estarán basadas en la exploración de datos que entregan *softwares* y herramientas como las mencionados anteriormente, se espera que la manipulación y transformaciones que entregue *DashAI* en conjunto con la extensibilidad de la plataforma permita que se mantenga relevante en el tiempo, adaptándose a los cambios en el ámbito de la Inteligencia Artificial.

2.2. Tecnologías consideradas

Esta Sección habla de los lenguajes de programación en los que se basa el código de *DashAI* y las distintas herramientas y *frameworks* que se utilizan para lograr que la plataforma sea robusta y eficiente.

2.2.1. Desarrollo de software

El desarrollo de un *software* se divide en dos principales lados. El *front-end* o lado del cliente, que contiene los componentes que integran la interfaz de usuario. El *back-end* o parte trasera, que se encarga de la lógica y procesamiento de la información con la que se alimenta el *front-end*. Ambos se desarrollan con distintos *frameworks* y herramientas que se detallan a continuación.

Back-end

El *back-end* del proyecto utiliza el lenguaje de programación *Python*, que se conecta al *front-end* mediante una *API* integrada con el *framework FastAPI*, que destaca por su rendimiento y eficiencia, permitiendo realizar consultas rápidas y precisas. Esta *API* fue implementada con anterioridad por un egresado parte del equipo de *DashAI*, y su trabajo en detalle se encuentra en su memoria[22].

FastAPI hace uso de *Pydantic*[4] para la validación y serialización de datos. Esta combinación proporciona un sistema de tipado estático que facilita la identificación temprana de errores y fortalece la integridad del código, asegurando su robustez, ya que los tipos se validan en tiempo de desarrollo.

En resumen, la *API* contiene una arquitectura *RESTful*[16]. Es decir, entre otros, que contiene las siguientes características:

- Se hace uso explícito de los métodos HTTP, donde cada método se utiliza para un único propósito. Esto contempla un método `GET` para obtener recursos desde el *back-end*, método `POST` para crear nuevos recursos, un método `PUT` para actualizar un recurso y un método `DELETE` para borrarlos.
- Cada petición hecha a la *API* no depende de otra hecha anteriormente.
- Se utilizan *Uniform Resource Identifier* o *URIS* para diferenciar cada *end-point*, estos son similares a estructuras de carpetas donde cada *end-point* se localiza con un nombre.

Para el manejo de la base de datos se utiliza *SQLAlchemy*[5], una herramienta de mapeo objeto-relacional (*Object-Relational Mapping, ORM*). Este *ORM* simplifica la gestión de la base de datos sin necesariamente tener un conocimiento de *SQL*. En este proyecto, se utiliza una base de datos *SQLite*[6] para almacenar y administrar los datos de manera eficiente.

Front-end

En el *front-end* se utiliza el *framework React* que utiliza sintaxis *JSX*, esta sintaxis es una extensión del lenguaje *Javascript* y sirve para crear interfaces de usuario de forma eficiente

[4]Documentación de Pydantic: `https://docs.pydantic.dev/latest/`
[5]Documentación de SQAlchemy: `https://www.sqlalchemy.org/`
[6]Documentación de SQLite: `https://www.sqlite.org/index.html`

y legible, y que además facilita la creación de componentes.

También, se hace uso de sintaxis *Typescript* para restringir los tipos de datos de entradas y parámetros en las conexiones con la *API*. Esto agrega un nivel adicional de robustez al código al proporcionar información detallada sobre los tipos de datos esperados, lo que facilita la detección de errores en tiempo de compilación, similar al uso de *Pydantic* en el *back-end*.

2.2.2. Bibliotecas científicas de *Machine Learning*

Complementando las herramientas de desarrollo de *software*, existen librerías relacionadas a Inteligencia Artificial que son fundamentales en el proyecto de *DashAI*. *DashAI* es una plataforma creada para manipular conjuntos de datos y utilizarlos en el entrenamiento de modelos de Inteligencia Artificial sin que el usuario tenga necesidad de conocer su programación en profundidad. Por esto, se utilizan herramientas y modelos implementados en el *back-end* de la aplicación.

Hugging Face Datasets

Datasets es una librería desarrollada por la empresa *Hugging Face*, que permite acceder, manipular y procesar conjuntos de datos para tareas o *Tasks* de Audio, Visión Computacional y Procesamiento de Lenguaje Natural (NLP). Esta librería permite procesar los datos para los entrenamientos dejándolos en el formato de tablas de *Apache Arrow* que son altamente flexibles y eficientes.

DashAI extiende los *Datasets* de *Hugging Face* con la clase de *DashAI Datasets* permitiendo compatibilidad con las distintas tareas y modelos.

Modelos de *Scikit-Learn*

Scikit-Learn[7] es una librería de *Python* que contiene algoritmos de clasificación, regresión, clustering, reducción dimensional, entre otras funcionalidades. Es una librería de código abierto y ofrece buena integración con otras librerías usadas en *Machine Learning* como *Pandas*, librería ampliamente usada en *data science*. Es por estos atributos que *DashAI* implementa sus modelos haciendo uso de esta librería.

[7]Documentación de *Scikit-Learn*: `https://scikit-learn.org/stable/`

Capítulo 3

Solución

El siguiente Capítulo está dividido en cuatro partes que explican como se llega a la solución de los problemas presentes en *DashAI*. Primero explica como se encontraba el flujo previo a la memoria y porque esto causaba los problemas definidos para el usuario. Luego habla de como se diseño la solución para lograr los objetivos planteados y su posterior implementación. Para concluir en como se refactorizó el código de la aplicación para lograr la integración de la solución por completo.

3.1. Flujo previo

La Sección actual tiene como objetivo presentar cual era el flujo en *DashAI* previo al trabajo de la memorista. Primero muestra los pasos a seguir por el usuario tanto en la carga de *Datasets* como en la creación de nuevos experimentos y luego detalla el estado de la *API* que conecta todo este flujo con el *back-end*. Lo mostrado en esta Sección está desarrollado en su totalidad por desarrolladores, memoristas y tesistas del equipo de *DashAI*.

3.1.1. Carga de datos

Al abrir *DashAI* el usuario ve un tablero con las tres secciones que tiene ya implementada la plataforma, como se puede apreciar en la Figura 3.1. Desde este menú se puede acceder a las siguientes vistas, las que se comentarán a profundidad:

- **Datasets:** Vista donde el usuario puede subir y configurar sus conjuntos de datos.
- **Experiments:** Vista donde el usuario puede crear y conocer en que estado están sus experimentos, si ya se entrenaron.
- **Results:** Vista donde el usuario puede ver y comparar los resultados de los distintos modelos configurados por cada experimento creado y entrenado.

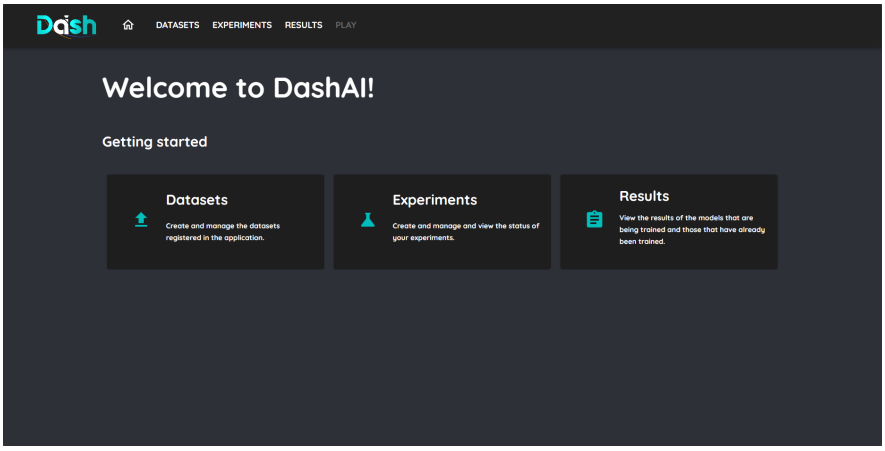

Figura 3.1: Captura de pantalla con la vista principal de *DashAI*.

Al ir a la vista de *Datasets* el usuario ve una lista de todos los conjuntos de datos que ha subido a la plataforma. esta lista contiene la ID, nombre, la *Task* asociada y en que fecha se agregó y modificó cada conjunto de datos. En caso de que el usuario quiera subir nuevos datos, puede presionar el botón *"New Dataset"* que se encuentra en la esquina superior derecha de la tabla.

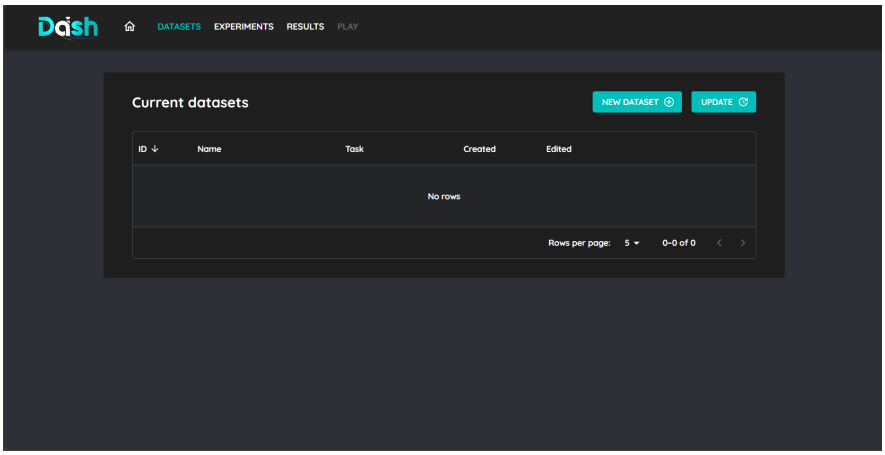

Figura 3.2: Captura de pantalla de la vista *Datasets*.

Al presionar el botón para agregar un nuevo conjunto de datos, se abre una ventana emergente donde el usuario puede ir navegando por los pasos que necesita completar para subir y configurar un nuevo *Dataset*. Los pasos son los siguientes:

1. **Select Task:** Se elige y asocia una *Task* al *Dataset*.
2. **Select a way to upload:** Se selecciona con que *DataLoader* cargar el conjunto de datos.
3. **Configure and upload your dataset:** Se sube y configura el conjunto de datos.

En detalle, el primer paso que se muestra en la Figura 3.3, cuenta con la lista de las *Task* que están disponibles en *DashAI*, donde el usuario tiene que elegir que *Task* quiere

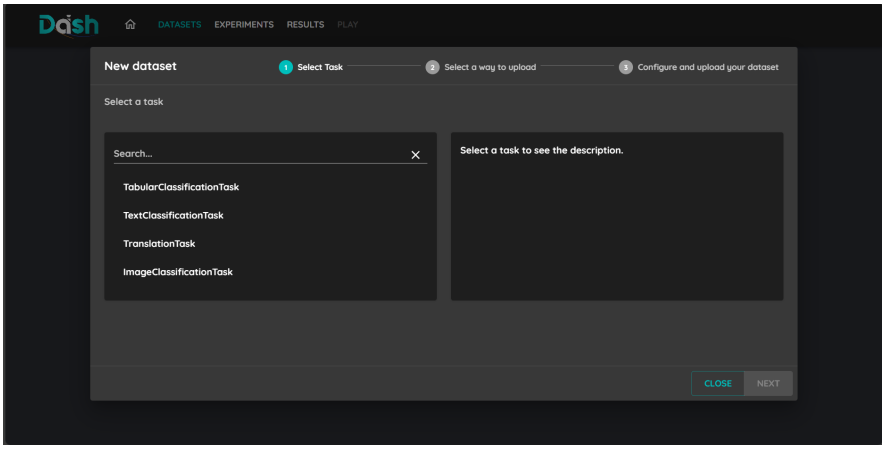

Figura 3.3: Captura de pantalla del paso *"Select Task"*.

realizar con el *Dataset* que va a subir. Este paso del flujo genera parte del primer problema comentado en la Sección 1.3, ya que el usuario debe elegir una única *Task* a realizar y esto no le permite aprovechar el mismo conjunto de datos para distintos fines.

Avanzando al segundo paso, el usuario debe elegir qué *DataLoader* utilizar, tal como se muestra en la Figura 3.4. La lista contiene las opciones con las formas de cargar el conjunto de datos que están disponibles para la *Task* anteriormente seleccionada. Al elegir el *DataLoader* se puede avanzar al último paso.

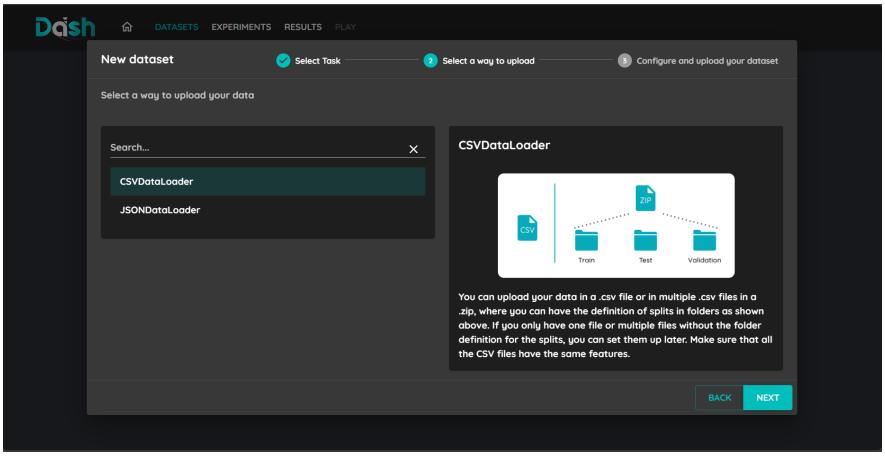

Figura 3.4: Captura de pantalla del paso *"Select a way to upload"*.

Finalmente, luego de seleccionar *Task* y *DataLoader*, el usuario llega al paso mostrado en la Figura 3.4 donde puede subir y configurar un *Dataset*. Este paso muestra una sección donde el usuario puede subir o traer un archivo de con su conjunto de datos y configurar parámetros relacionados.

Uno de los parámetros a configurar por el usuario son los *splits*, que representan las porciones del conjunto de datos subido que se quiere utilizar para entrenamiento, validación y testeo de los modelos. Esta es la única etapa en la que el usuario puede elegir cómo dividir sus datos, en caso de que quiera utilizar otras particiones debe volver a subir el conjunto de

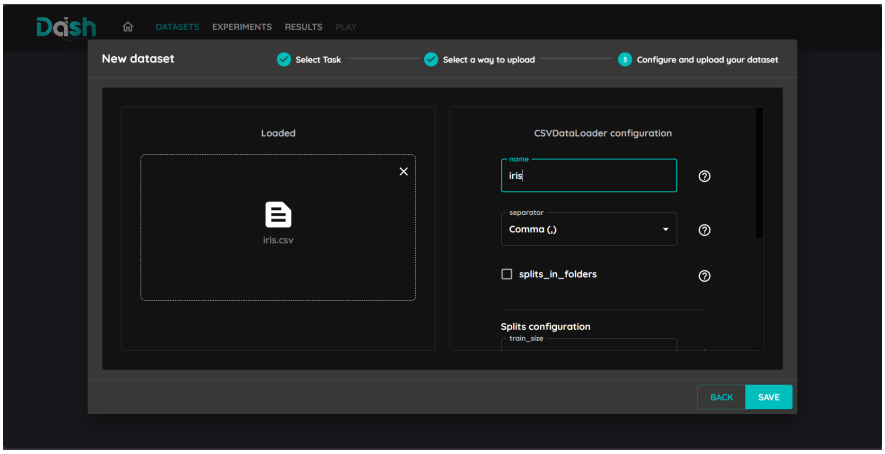

Figura 3.5: Captura de pantalla del paso *Configure ans upload your dataset.*

datos pero con una configuración de *splits* diferente.

Anteriormente, se habló de que una etapa importante en la ciencia de datos es poder preparar los conjuntos de datos y que en muchos casos se debían configurar los modelos de distintas formas para poder buscar con cual se genera el mejor rendimiento. Esta característica aplica a configurar como se dividirá el conjunto de datos, por lo que elegir solo una forma de dividir los conjuntos de entrenamiento, testeo y validación entorpece los procesos de entrenamiento.

Al elegir nombre y subir el archivo, el usuario ya puede terminar de configurar su nuevo *Dataset DashAI* cerrando la ventana, donde se vuelve a mostrar el listado con el conjunto de datos recién subido ya disponible para manejar y utilizar en sus experimentos como se muestra en la imagen de la Figura 3.6.

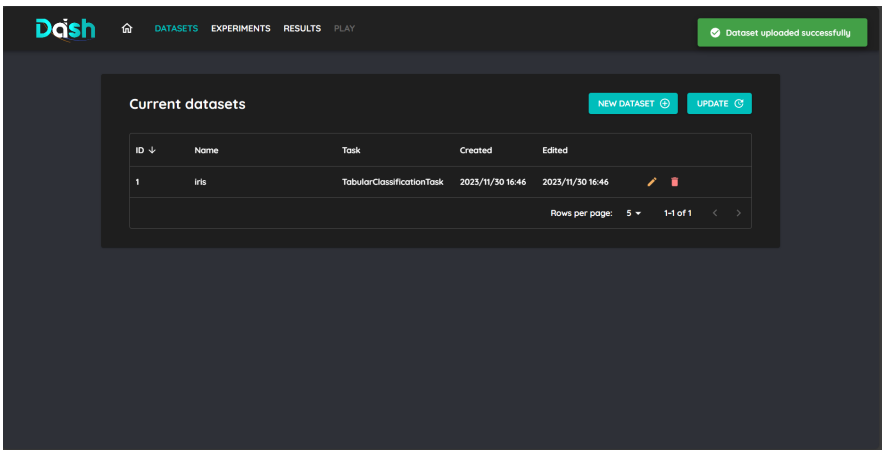

Figura 3.6: Captura de pantalla de la vista de *Datasets* luego de subir nuevos datos.

Como se puede apreciar, en ninguna parte de este flujo de tres pasos se le entrega al usuario alguna herramienta para verificar el contenido de su conjunto de datos, lo que está relacionado con el primer problema comentado, ya que la interfaz no muestra si los datos utilizados son los necesarios y esperados por la persona que está utilizando la plataforma.

3.1.2. Creación de experimentos

Para crear un nuevo experimento y hacer uso de sus conjuntos de datos, el usuario debe dirigirse a la ventana de *Experiments*, aquí al igual que en *Datasets* hay una lista de los experimentos creados que contiene el nombre, la *Task* y el *Dataset* asociados y las fechas de creación y modificación del experimento, como se muestra en la Figura 3.7. Para agregar un nuevo experimento el usuario puede presionar el botón *"New Experiment"* ubicado en la esquina superior derecha de la tabla.

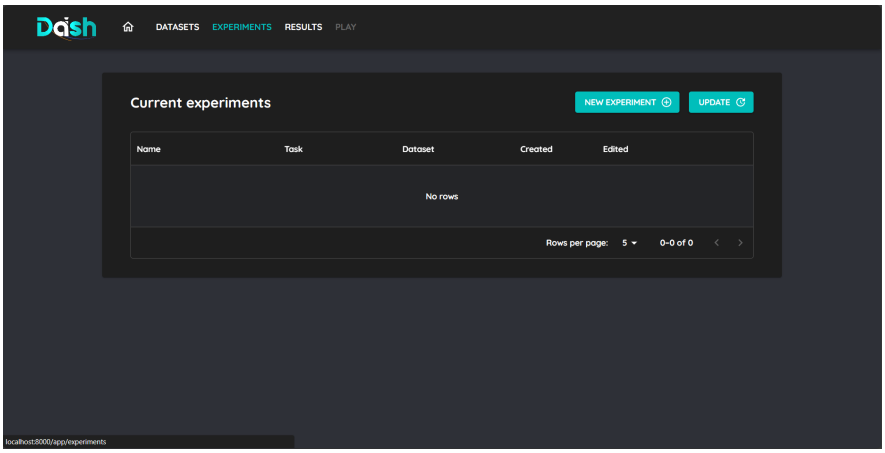

Figura 3.7: Captura de pantalla de la vista *Experiments*.

Luego de presionar el botón, se abre una nueva ventana emergente donde el usuario puede ir navegando por los diferentes pasos que se necesitan para crear un nuevo experimento. Los pasos son los siguientes:

1. **Set name and task:** Se nombra el experimento y se elige la *Task*.
2. **Select dataset:** Se elige que *Dataset* utilizar.
3. **Configure models:** Se seleccionan y configuran los modelos de entrenamiento.

En la Figura 3.8 se muestra el primer paso de la creación de experimentos, allí hay un campo donde el usuario puede ingresar el nombre del experimento y una lista donde puede elegir la *Task* que quiere realizar de una lista que contiene todas las *Task* disponibles en *DashAI*.

Al seleccionar la *Task* y avanzar al segundo paso, mostrado en la Figura 3.9, el usuario debe elegir que conjunto de datos utilizar, aquí se listan solo los conjuntos de datos que en su carga fueron configurados con la *Task* elegida en el paso anterior.

Por ejemplo, en la Figura 3.8, el usuario está en el primer paso y selecciona la *Task* *TabularClassificationTask*, mientras que en la Figura 3.9 el usuario está en el paso 2 y solo se muestra el *Dataset* de nombre *'iris'*, esto porque en al cargar este conjunto de datos se le asoció esa *Task* en específico. En el caso de que existiera otro conjunto de datos con otra *Task* asociada este no se mostraría en el listado.

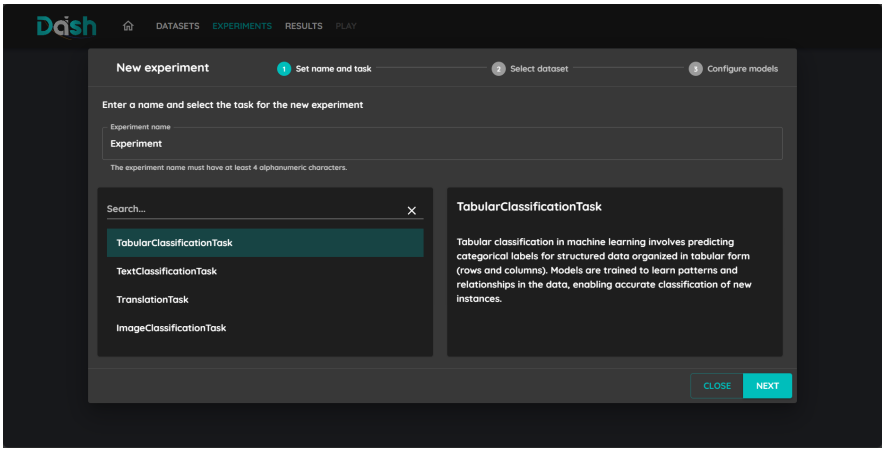

Figura 3.8: Captura de pantalla del paso *"Set name and Task"*.

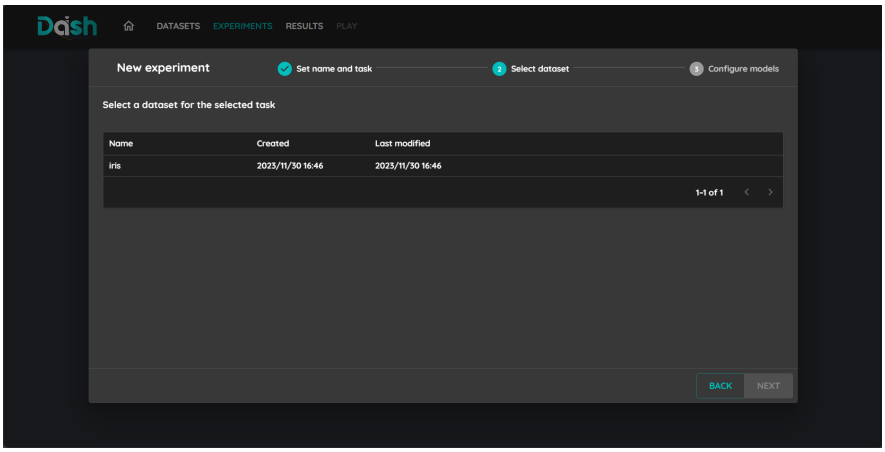

Figura 3.9: Captura de pantalla del paso *"Select dataset"*.

Lo que, nuevamente, limita a solo utilizar ciertos conjuntos de datos y si se quisiera crear un experimento con una *Task* distinta, tendría que subirse el conjunto de datos nuevamente, volviendo a presentarse el segundo problema, que establece que no se permite aprovechar un conjunto de datos para distintas *Tasks*.

Después de elegir la *Task* y conjunto de datos el usuario puede elegir y configurar modelos disponibles, como se muestra en la Figura 3.5 donde el usuario agregó un modelo *"SVC"* y está agregando otro de nombre *"KNeighborsClassifier"*.

Finalmente, luego de configurar los modelos para el experimento, este se crea y se cierra la ventana emergente. Al cerrar se vuelve a mostrar la lista de experimentos ya actualizada como se aprecia en la Figura 3.6.

Las dos secciones anteriores detallan como el usuario sube un *Dataset* para una *Task* en especifico y al crear un nuevo experimento los conjuntos de datos que puede elegir son los que ya están acopladas a ella. Este es el flujo que es cambiado en el trabajo de la memorista y que se detallará en la Sección 3.2 de Diseño.

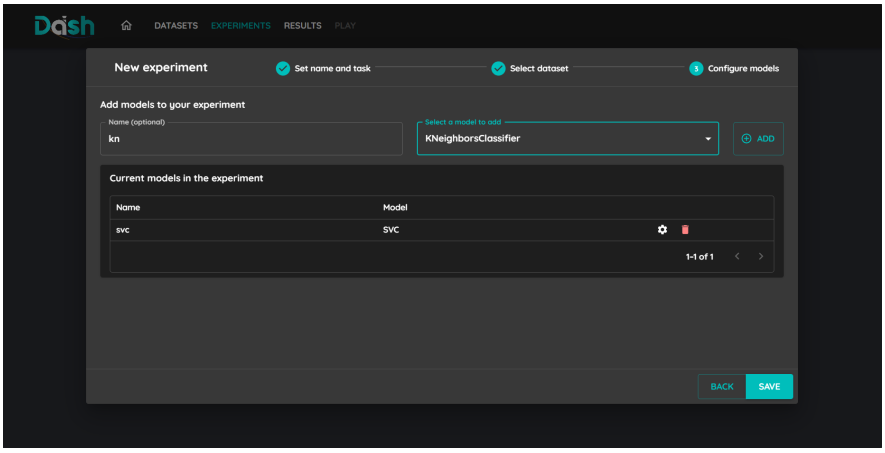

Figura 3.10: Captura de pantalla del paso *Çonfigure models"*.

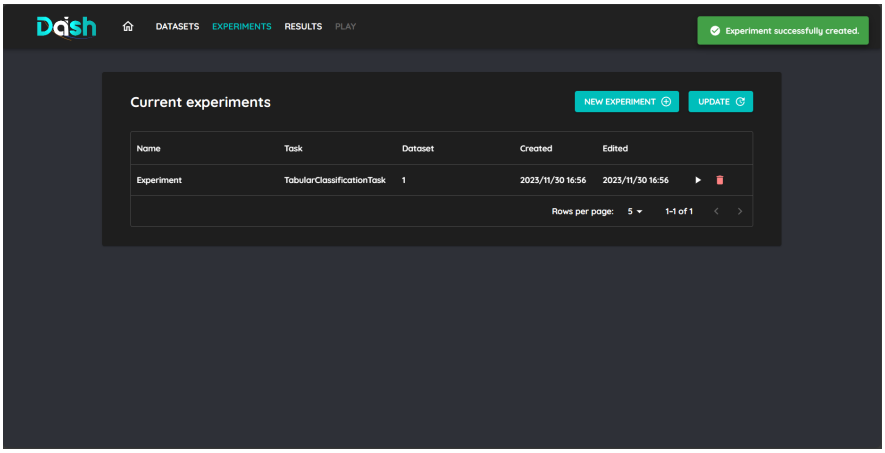

Figura 3.11: Captura de pantalla de la vista de *Experiments* luego de la creación de un nuevo experimento.

3.1.3. API

La *API* de *DashAI* contiene distintos *end-points* que alimentan al *front-end* con información a través de la API, estos son:

- **Components:** Este elemento sirve para informar al *front-end* acerca de componentes que contiene *DashAI*, ya sea información sobre las *Task* que están disponibles, los modelos, las métricas, entre otros. Para hacer peticiones con la *API* a este *end-point* se utiliza el *URI* `/component/`.
- **Datasets:** Este elemento entrega y recibe lo necesario para administrar y configurar los elementos de la entidad de los conjuntos de datos del usuario en la base de datos. Para hacer peticiones con la *API* a este *end-point* se utiliza el *URI* `/dataset/`.
- **Experiments:** Este elemento entrega y recibe lo necesario para administrar y configurar los distintos elementos de la entidad de experimentos de la base de datos creados por el usuario. Para hacer peticiones con la *API* a este *end-point* se utiliza el *URI* `/experiment/`.

- **Jobs:** Ente elemento entrega y recibe información sobre las colas de trabajos que serán ejecutados. Para hacer peticiones a este *end-point* con la *API* se utiliza el *URI* `/job/`.
- **Runs:** Este elemento entrega y recibe los metadatos de los modelos de *Machine Learning* seleccionados por el usuario en un experimento. Para hacer peticiones a este *end-point* con la *API* se utiliza el *URI* `/run/`.

Cada uno de estos *end-points* contiene sus propios métodos HTTP que son manejados mediante una interfaz *CRUD* (sigla en inglés de *Create-Read-Update-Delete*). Por lo que para que el *front-end* haga requerimientos por medio de ellos se hacen llamas HTTP con la *URI* en específico. Por ejemplo en el caso de los *Datasets*, utilizando el método HTTP `GET` se pueden leer todos los conjuntos de datos almacenados y subidos por el usuario o uno en específico con al proporcionar la ID en los parámetros.

Esta es la forma en que la *API* está definida. Este punto es importante ya que el trabajo de la memorista contempla tanto la modificación como añadir *end-points* que se puedan consultar por medio de ella. Los *end-points* como se presentan en esta Sección eran suficientes para abastecer al *front-end* de la información que necesitaba, pero el nuevo diseño contempla mayor interacción con el *back-end* por lo que la solución requiere de estas nuevas modificaciones que se detallarán en la Sección de Implementación 3.3.1.

3.2. Diseño de la solución

En esta Sección primero se detalla el diseño a implementar por la memorista, cuyo trabajo se realizó presentando *mock-ups* previos hechos en la plataforma *Figma*, que posteriormente eran discutidos con el equipo de *DashAI*, quienes sugerían cambios en términos de usabilidad y estilo. También se muestra como estas nuevas vistas afectan el flujo de subida de conjuntos de datos y creación de experimentos tanto en el *front-end* como en el *back-end*.

3.2.1. Diseño de nuevo flujo de la carga de datos

Para lograr el Objetivo 2 acerca de mejorar la retroalimentación al usuario en el proceso de carga de datos, el nuevo flujo debe contener mayor información sobre el conjunto de datos, por lo que en primera instancia se propone una nueva vista en una ventana emergente accesible desde cada conjunto de datos de la tabla de la página de *'Datasets'*. El modal contiene una tabla con los nombres de las columnas, los tipos y muestras de los valores que tiene cada una.

Con esta tabla el usuario podría conocer que tipos de datos existen en su dataset y al mismo tiempo asegurarse de que el conjunto de datos es el que esperaba subir a la plataforma. Esto evita que tenga que hacer uso de otra plataforma o herramienta fuera de *DashAI* para revisar el contenido de sus datos y al mismo tiempo ayuda a que el proceso de entrenamiento se realice a ciegas.

A lo anterior, también se suman los cambios para que el usuario no tenga que ligar una *Task* al conjunto de datos que esta subiendo desde la interfaz. Es decir, se elimina el paso

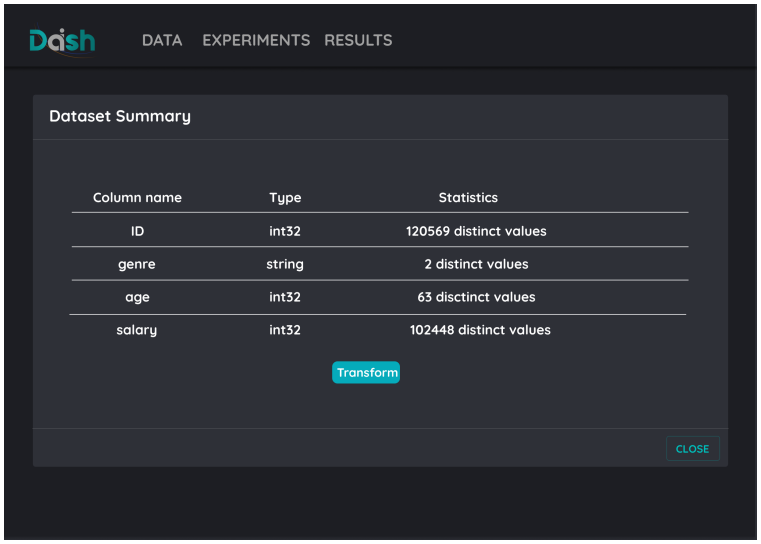

Figura 3.12: Mock-up del resumen de un conjunto de datos con los tipos de cada columna.

previo de elección de *Task* al subir un nuevo dataset, dejando como primer paso la elección de *DataLoader* en lugar de ella. Esto contribuye al logro parcial del Objetivo 1 que habla del desacoplamiento de los datos a las *Tasks*, ya que el usuario ya no necesita elegir la *Task* previo al experimento y dejarla enlazada al conjunto de datos.

Además, a continuación de los pasos de elección de *DataLoader* y de la subida del conjunto de datos se incluye un nuevo paso de configuración del conjunto de datos que incluye una vista previa y la posibilidad de cambiar los tipos por rango de los índices de las columnas, es decir, el usuario puede introducir rangos de columnas que quiera que tengan cierto tipo de dato, el cual estará observando en la vista previa.

Este paso al igual que la vista emergente con el resumen busca lograr el objetivo de la mayor retroalimentación, con esto el usuario y además podrá interactuar con el conjunto de datos en caso de que no esté conforme con el valor asignado por defecto, cambiando el tipo siempre y cuando este sea un cambio válido.

Como fue mencionado en la Sección de Metodologías de desarrollo, Sub-sección de Discusiones acerca del diseño 1.5.3, estos *mock-ups* son presentados al equipo *DashAI* en las reuniones diarias y semanales, en estas instancias se discuten y validan los componentes y contenido de la vista hasta llegar a un diseño adecuado. Así, luego de iteraciones a la propuesta de la nueva interfaz del flujo de subida de un nuevo conjunto de datos, se llega al diseño final que contiene los siguientes pasos a seguir para el usuario:

1. **Select a way to upload:** Este paso es el mismo que contenía el anterior flujo y permite al usuario elegir el *DataLoader* a utilizar para subir el conjunto de datos.
2. **Configure and upload your dataset:** Este paso también ya estaba implementado en el flujo y permite al usuario subir y configurar algunos parámetros del conjunto de datos.
3. **Dataset Summary:** Nuevo paso que permite al usuario ver una tabla con los nombres

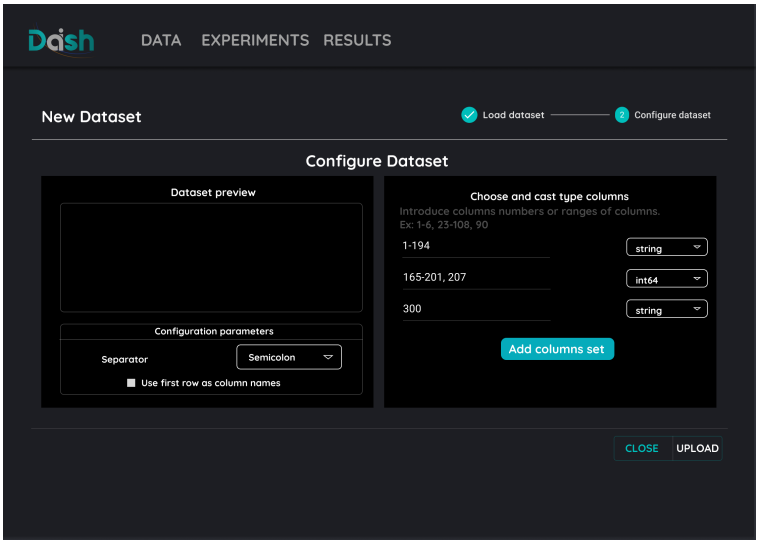

Figura 3.13: Mock-up preliminar de la vista previa y configuración del conjunto de datos subido.

de las columnas, un ejemplo de dato y los tipos de las columnas, otorgando la posibilidad de cambiar el tipo colocado automáticamente por *Hugging Face*.

Como se puede apreciar, el tercer paso del flujo nuevo no es una vista previa y elección de tipos, sino más bien un resumen que deja como opcional el cambio de los tipos que se suben por defecto. Este cambio se debe a que en el *mock-up* propuesto, la vista previa queda con poco espacio, lo que no es óptimo para conjunto de datos con múltiples columnas, además de no quedar claro que el cambio de tipos es opcional. Por lo que se decide dejar solo una muestra de dato por columna y los tipos opcionales, permitiendo que el usuario conozca qué contiene el conjunto de datos subido y que pueda interactuar con los tipos de una forma más sencilla, solo si lo requiere.

Los cambios propuestos con los nuevos pasos para la carga de datos requieren de llamadas a la *API* para subir el *Dataset* y para obtener información de este, por lo que contemplando la lógica de *front-end* y *back-end* se consigue el flujo final presentado en la Figura 3.14.

Con esta solución, cuyos cambios asociados a la implementación y las definiciones de cada método se explicarán en profundidad en la Sección 3.3.1, se elimina el problema que presentaba *DashAI* donde los conjuntos de datos solo se podían usar para una *Task*, ya que esta no es parte de los atributos del dataset. Además, ya no es necesario que el usuario conozca desde antes su conjunto de datos, ya que la misma plataforma le otorga retroalimentación sobre el mismo al subirlo y este puede decidir si finalmente quiere utilizar ese conjunto de datos para la *Task* en especifico en el futuro.

3.2.2. Diseño del nuevo flujo de la creación de experimentos

Complementando lo anterior, para lograr el objetivo de desacoplar los *Datasets* de la *Task* en su totalidad, el paso de elección de conjunto de datos para la *Task* en el flujo para crear un

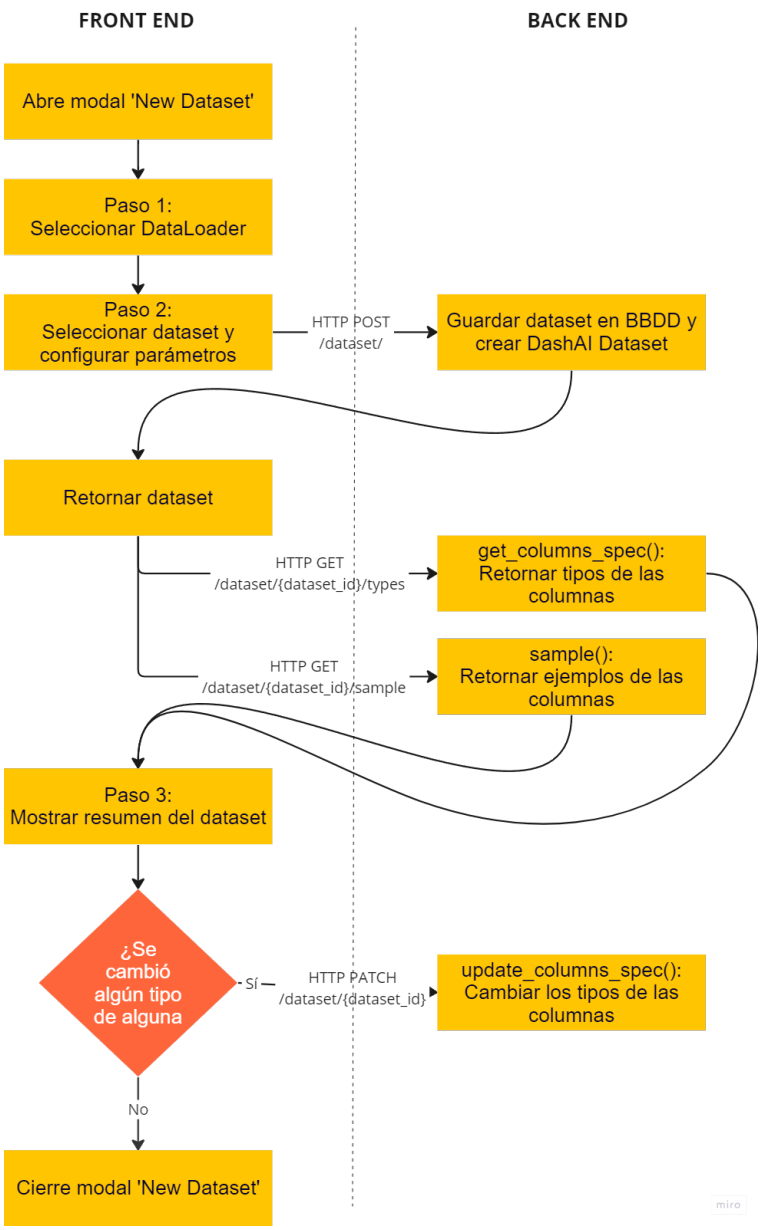

Figura 3.14: Diagrama de flujo implementado al subir un nuevo conjunto de datos.

nuevo experimento ahora lista la totalidad de conjunto de datos. Antes esta lista mostraba solo los *Datasets* que tenían la *Task* seleccionada ligada a él como atributo, como ahora estos no tienen una *Task* asociada la lista muestra todos los *Datasets* que están subidos en la plataforma.

A los pasos de selección de *Task* y *Dataset* se agrega un nuevo paso que contribuye al logro del Objetivo 3, dando mayor flexibilidad al usuario y logrando que pueda usar el mismo conjunto de datos con diferentes configuraciones en distintos experimentos. Esto es posible porque en el nuevo paso del flujo el usuario puede ingresar los siguientes datos:

Primero, en la parte superior de la vista, esta la configuración de la separación del conjunto de datos por columnas, que se utilizan al correr el experimento permitiendo que el usuario use distintas combinaciones de columnas:

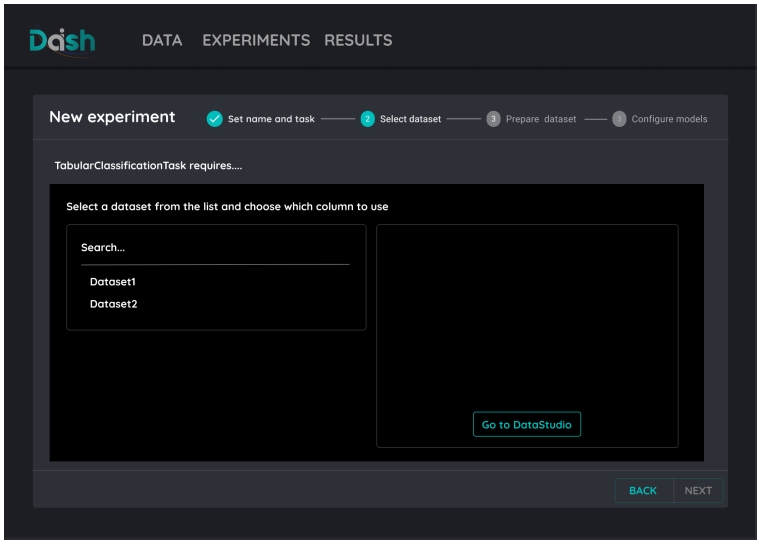

Figura 3.15: Mock-up de la lista de conjuntos de datos completa.

- Columnas de entrada: Son las columnas que entregan la información en los modelos.

  En este campo se deben ingresar los índices de las columnas individualmente o por rangos separados por coma. Por defecto se muestra el rango de índices de todas las columnas menos la última.

- Columnas de salida: Son las columnas que tiene las clases o información de salida en los modelos.

  En este campo se deben ingresar los índices de las columnas individualmente o por rangos separados por coma. Por defecto se muestra el índice de la última columna.

Luego, en la parte inferior de la vista, está la configuración de la separación del *Dataset* por particiones, donde el usuario debe elegir que porción de las filas del conjunto de datos quiere utilizar para entrenamiento, validación y testeo, aquí se entregan dos opciones al usuario:

- Dividir el *Dataset* por porcentaje de filas: Si se elije esta forma de dividir el conjunto de datos, en los campos de entrenamiento, validación y testeo se debe ingresar cada porcentaje respectivo y deben sumar 100 %. Por defecto se muestran los valores estándar o comúnmente usados.

- Dividir el *Dataset* manualmente: Si se elije esta forma de dividir el conjunto de datos, en los campos de entrenamiento, validación y testeo se deben ingresar los índices de las filas individualmente o por rangos separados por una coma en cada campo respectivamente.

Luego de elegir las configuraciones, la interfaz muestra si las columnas elegidas son válidas y cumplen con los requerimientos de la *Task* seleccionada. Logrando el Objetivo 4 de lograr mayor usabilidad por medio de la validación de columnas elegidas para cada *Task*.

Con este diseño el usuario puede ingresar distintas configuraciones de *splits* y columnas en distintos experimentos pero con el mismo conjunto de datos, independiente de las particiones

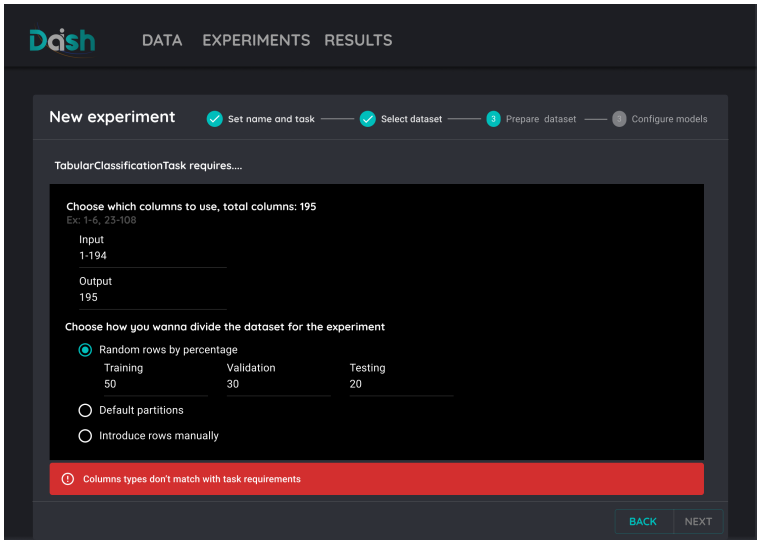

Figura 3.16: Mock-up del nuevo paso para preparar el conjunto de datos.

que fueron configuradas al subirlo, ayudando a que pueda aprovechar en mayor medida los datos recolectados y subidos.

Al igual que en la etapa de diseño del flujo de *Datasets*, los *mock-ups* presentados se someten a discusión y validación por parte del equipo de *DashAI*, esto conlleva a que se itere sobre el diseño hasta que contenga los componentes necesarios para cubrir la nueva funcionalidad. Finalmente, con los pasos detallados anteriormente, el nuevo flujo de la creación de nuevos experimentos contempla los siguientes pasos a seguir por el usuario:

1. **Set name and task**: Paso ya implementado donde se define el nombre del nuevo experimento y se elije la *Task* a realizar.
2. **Select dataset:** Paso modificado donde se puede seleccionar el conjunto de datos a utilizar desde una tabla con todos los conjuntos de datos subidos.
3. **Prepare dataset:** Nuevo paso que permite elegir columnas de entrada y salida y que particiones de las filas se quiere utilizar para cada etapa.
4. **Configure models:** Paso existente donde se eligen los modelos para el experimento y se configuran.

Los cambios propuestos para la creación de nuevos experimentos requieren de llamadas a la *API* y de definiciones de nuevos métodos que validen las configuraciones ingresadas por el usuario. Contemplando la lógica necesaria en *front-end* y *back-end* se consigue el siguiente flujo final presentado en la Figura 3.17.

Con esta solución, cuya implementación donde se detallan las definiciones de cada método añadido al código está en la Sección 3.3.2, se elimina el problema que causaba la no validación de las columnas pues ahora el entrenamiento se puede hacer con la *Task* seleccionada ya que se procura de que el conjunto de datos y las columnas seleccionadas sean las apropiadas para ella.

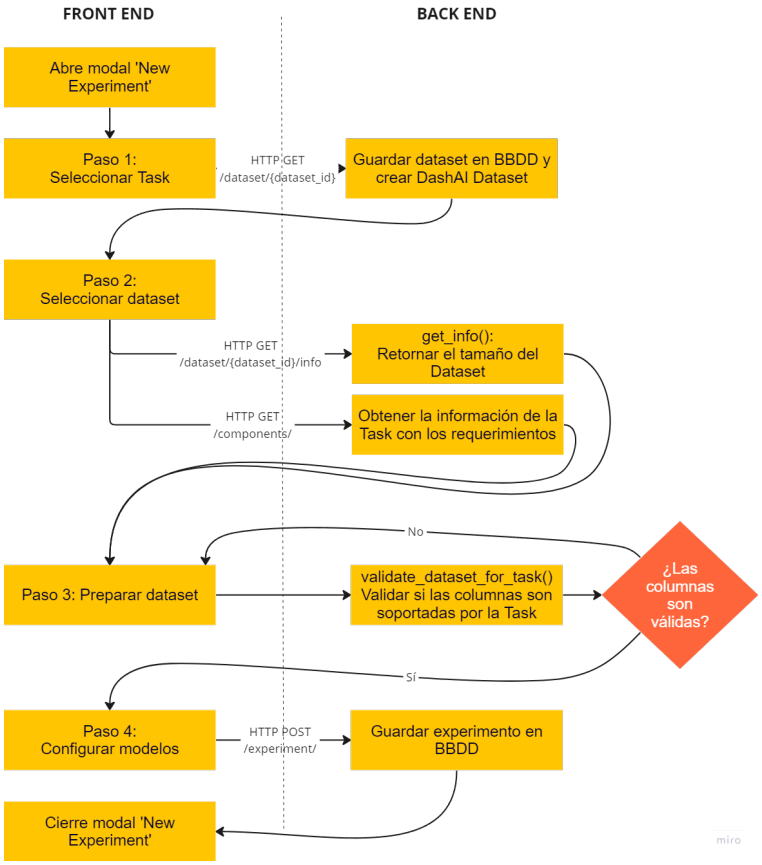

Figura 3.17: Diagrama de flujo implementado al crear un nuevo experimento.

3.2.3. Modificaciones al modelo de datos

La Sección a continuación muestra los cambios que se necesita aplicar al modelo de datos existente para lograr el diseño propuesto en la Sección 3.2 que contemplan nuevos flujos dentro de la plataforma.

En concreto, el modelo anterior guarda los *feature_names* en el modelo *Datasets*, esto correspondía a las columnas de entrada, es decir las columnas del *Dataset* que no eran usadas como salida. Este atributo ya no es necesario por lo que se elimina. En las siguientes figuras se presenta una comparación de la entidad *Dataset* antes y después de las modificaciones.

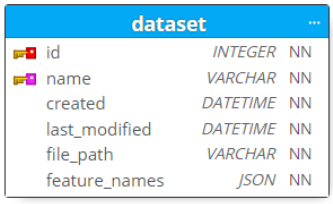

Figura 3.18: Entidad *Dataset* del antiguo modelo de datos.

Con las modificaciones propuestas el modelo actualizado para *Dataset*, pasa de ser del presente en la Figura 3.18 al de la Figura 3.19.

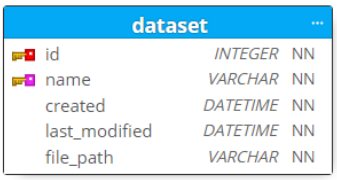

Figura 3.19: Entidad *Dataset* del modelo de datos actualizado.

Además de lo anterior, se cambia el modelo de *Experiment* para que pudiera almacenar las columnas a usar y los valores de *split*. Es decir, se agregaron los campos de columnas de entrada, que guarda los nombres de las columnas que sirven como información de entrada a los experimentos, columnas de salida, que corresponden a los valores que se debe predecir por los modelos y finalmente las de *split* que corresponde a en que fracción del conjunto de datos se quiere usar para cada etapa del experimento, ya sea entrenamiento, validación o testeo.

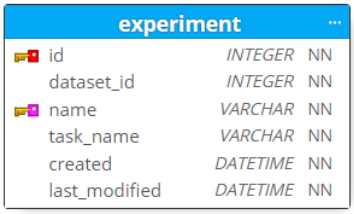

Figura 3.20: Entidad *Experiment* del antiguo modelo de datos.

Con las modificaciones propuestas queda el modelo de la Figura 3.21 donde se agregan los siguientes atributos:

- `input_columns`: Lista de las columnas de entrada.
- `ouput_columns`: Lista de las columnas de salida.
- `splits`: JSON que almacena la configuración de particiones para el dataset.

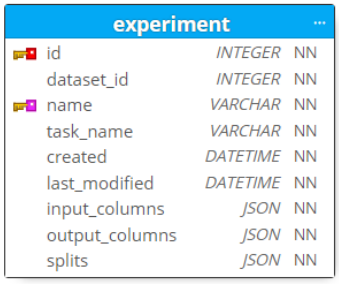

Figura 3.21: Entidad *Experiment* del modelo de datos actualizado.

Las modificaciones detalladas en los párrafos anteriores proponen que el nuevo modelo de datos sea el especificado en la Figura 3.22.

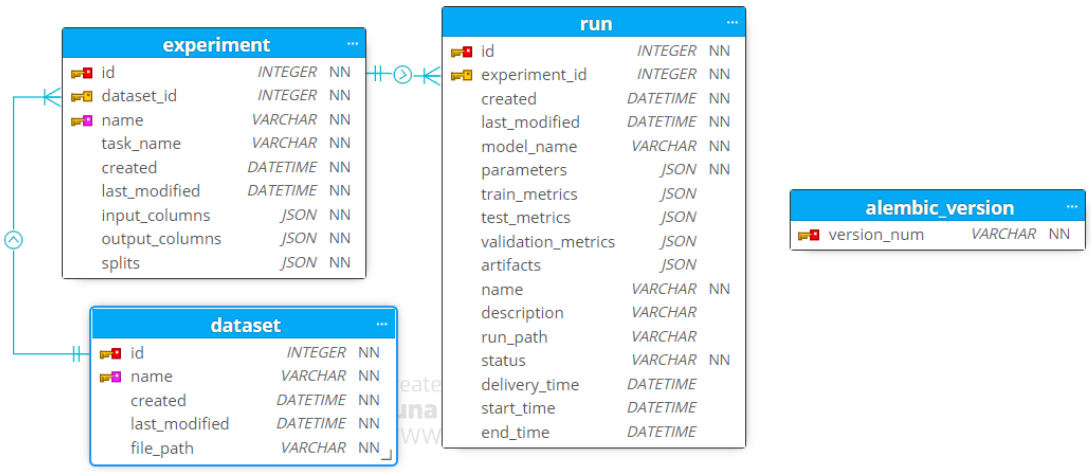

Figura 3.22: Modelo actual base de datos de *DashAI*.

Lo anterior se hace con el fin de que cada experimento en particular pueda tener su propia configuración de como utilizar el conjunto de datos seleccionado, el modelo anterior indicaba que cada experimento que se asocie al *Dataset* debía utilizar el conjunto de datos en su totalidad y con la misma división de particiones. Si el usuario quería utilizar una configuración distinta debía subir el conjunto de datos nuevamente a la plataforma.

3.3. Implementación de la solución

Esta Sección explica como se implementó cada uno de los cambios detallados en la Sección de Diseño 3.2, contemplando el desarrollo de nuevas vistas en el *front-end* y la modificación de los *end-points* de la *API* ya existente, agregando métodos que integren los cambios necesarios para poder lograr el nuevo flujo. Además de especificar otros cambios de refactorización del código que se implementaron para que los experimentos puedan ser ejecutados luego de los cambios de flujos descritos.

3.3.1. Implementación flujo carga de datos

Front-end

Para implementar el nuevo paso de *'Dataset summary'* se agrega una nueva vista que contiene una tabla paginada, implementada con componentes de la librería *MUI* y componentes configurables implementados previamente en la plataforma. Esta tabla obtiene la información de cada columna, lo que incluye nombre, ejemplo, tipo de columna y tipo de datos. Esto se obtiene desde el *back-end* por medio de la *API* con el método HTTP `GET`. Esta vista se puede apreciar en la Figura 3.23.

En detalle, la tabla contiene un *drop-down* donde se puede seleccionar un nuevo tipo de

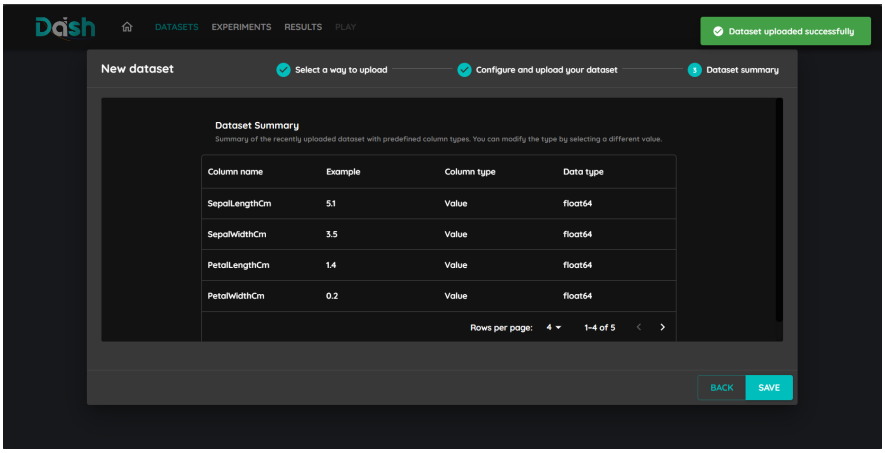

Figura 3.23: Captura de pantalla del paso de resumen del dataset.

columna y dato, en caso de que el usuario no esté de acuerdo con los tipos automáticos al ver el ejemplo.

La columna *Column type* contiene un *drop-down* donde se puede elegir entre los valores *Value* y *ClassLabel*, que corresponden a los tipos de las columnas que se pueden elegir. Mientras que las columnas de tipo *Value* representan valores escalares de distintos tipos, las columnas de tipo *ClassLabel* son las que tienen clases por lo que son generalmente usadas para tareas de clasificación.

A la columna anterior, le sigue la columna *Data type* que contiene una lista de todos los posibles tipos de datos que se le puede poner a una columna de tipo Value. Estos valores son todos los soportados por los *Datasets* de *HuggingFace*[1]. Un ejemplo de esto se puede apreciar en la Figura 3.24, donde el usuario está cambiado la columna *'SepalLenghtCm'* de tipo *float64* a tipo *string*.

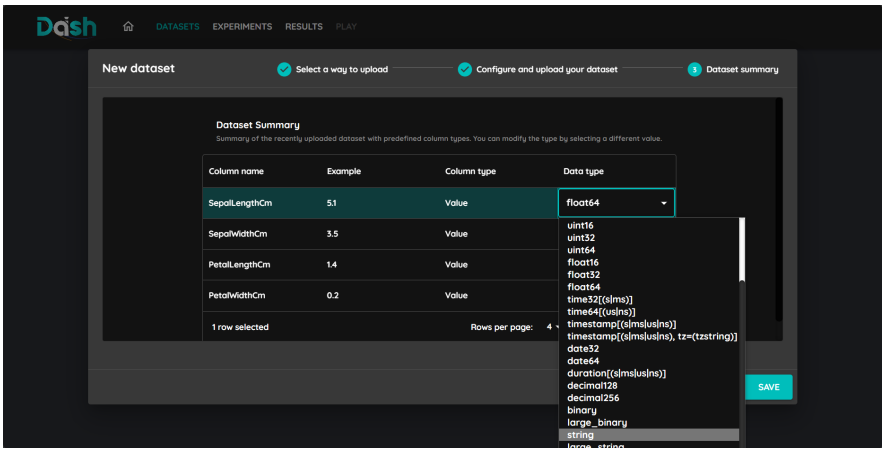

Figura 3.24: Captura de pantalla del paso de resumen del *Dataset* cambiando el tipo de la columna.

Existen otras modificaciones en la interfaz que fueron necesarias para tener coherencia con

[1]Valores soportados por la librería *Datasets*: `https://huggingface.co/docs/datasets-server/data_types`

el cambio en el flujo, por ejemplo, ya no se permite cambiar el tipo de la *Task* ni ver la *Task* en la tabla de conjuntos de datos ya que este atributo ya no existe. Esto se logró borrando las celdas de la lista de *Datasets*, como se puede apreciar en la Figura 3.25 y eliminando el atributo del nombre de *Task* en los *end-points* para crear y actualizar un conjunto de datos.

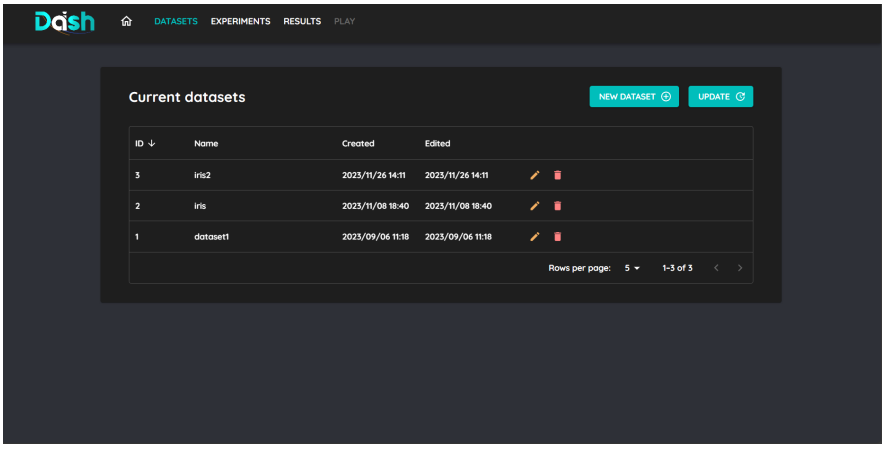

Figura 3.25: Captura de pantalla de la tabla de *Datasets* actualizada.

Back-end

Para hacer efectivos los cambios se modificaron los *end-points* de *Datasets* que ya estaban implementados en la plataforma, esto porque según el diseño planteado es necesario un *end-point* que entregue información además de actualizar los existentes con la eliminación del atributo del nombre de la *Task*. Específicamente, los *end-points* creados y modificados para la *URI* `/dataset/` son los siguientes:

1. **Método HTTP** `GET`
   - `/dataset/dataset_id/types`: Nuevo método que recibe la ID del *Dataset*, lo trae desde la base de datos y retorna los tipos de cada columna.

     El método puede fallar en caso de no encontrar el *Dataset* o de no establecer conexión con la base de datos.
   - `/dataset/dataset_id/sample`: Nuevo método que recibe la ID del *Dataset*, lo trae desde la base de datos y retorna una muestra de 10 filas desde la partición de entrenamiento.

     El método puede fallar en caso de no encontrar el *Dataset* o de no establecer conexión con la base de datos.
2. **Método HTTP** `POST`
   - `/dataset/`: Este método recibe los parámetros del *DataLoader*, la *URL* y el archivo con el *Dataset* y lo añade a la base de datos, creando los archivos necesarios localmente y creando el dataset como *Dataset DashAI*.

     Se modificó ya que antes guardaba las columnas de entrada y salida como parte del *Dataset DashAI*, lo cual ya no es necesario porque no se utilizan.

El método puede fallar en caso de no poder leer el archivo o no poder hacer conexión con la base de datos.

3. **Método HTTP** `PATCH`

   - `/dataset/dataset_id`: Este método recibe los parámetros, ya sea un nuevo nombre para el *Dataset* o un diccionario que contenga los nuevos tipos de las columnas a modificar. Este último parámetro fue añadido para recibir los cambios de tipos con la *API* por medio de este *end-point*.

     El método puede fallar en caso de no poder cargar el *Dataset* o no poder hacer conexión con la base de datos.

Al ponerse en contacto el *front-end* con el *back-end* por medio de los *end-points* de la API, estos responden a los requerimientos usando nuevos métodos implementados en la clase de *Dataset* de *DashAI*. Estas funciones acceden a los conjuntos de datos y entregan o cambian su información.

Los cambios para el flujo de carga de datos descrito permiten que el usuario suba un nuevo *Dataset DashAI*, lo explore y cambie los datos si es necesario, lo cual fue integrado por la memorista en su totalidad.

3.3.2. Implementación flujo creación de experimentos

Front-end

Para lograr este nuevo flujo y cumplir con los objetivos primero se modificó la lista de elección de conjunto de datos. Antes esta lista mostraba solo los *Datasets* que tenían la *Task* seleccionada ligada a él como atributo, como ahora los datos no tienen una *Task* asignada, la lista muestra todos los que están subidos en la plataforma en su totalidad. Desde el *back-end* se consiguen los conjuntos de datos por medio del método HTTP `GET` y se muestran como se aprecia en la Figura 3.26.

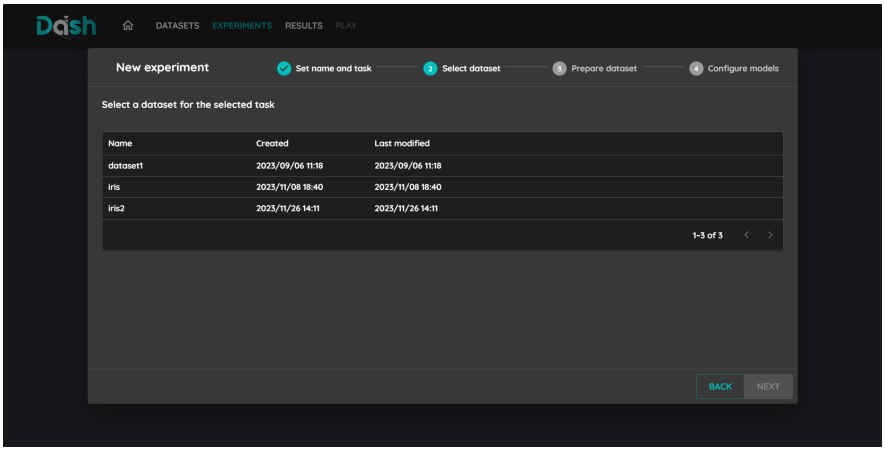

Figura 3.26: Captura de pantalla de la lista de conjuntos de datos completa.

Luego de elegir los datos que se quieren utilizar en el experimento, en la ventana se muestra una vista con el nuevo paso de *Prepare dataset*, que muestra los campos de columnas de entrada y salida y las de *split* (explicadas en la Sección 3.2.2) y que están implementados con campos de texto de la librería *MUI*.

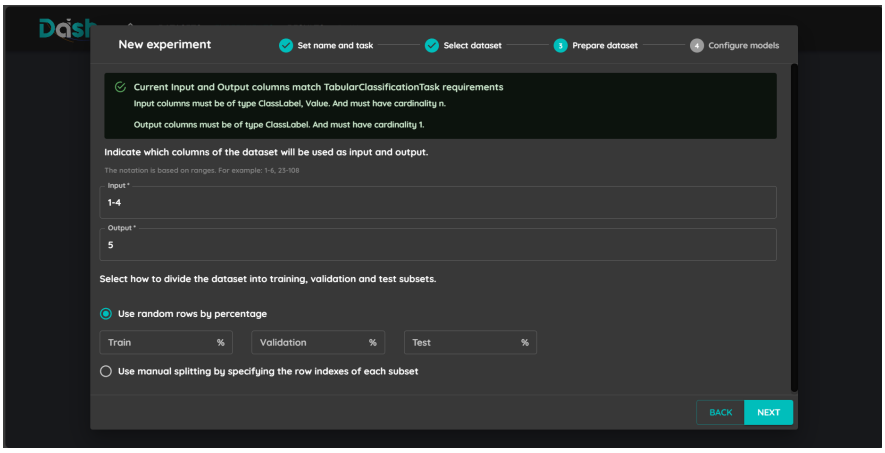

Figura 3.27: Captura de pantalla del nuevo paso para preparar el conjunto de datos.

Cada vez que el usuario introduce un nuevo valor a los campos de los rangos de columnas, se traduce este valor para actualizar la lista con todas las columnas dentro de los rangos. Dentro de la lógica se manejan distintos errores que pueden surgir con los valores que se introducen: el primero en caso de no cumplir con la sintaxis exigida, otro en caso de que algún índice ingresado sea mayor al total de columnas y otro en caso de que los rangos tengan el primer número mayor que el segundo.

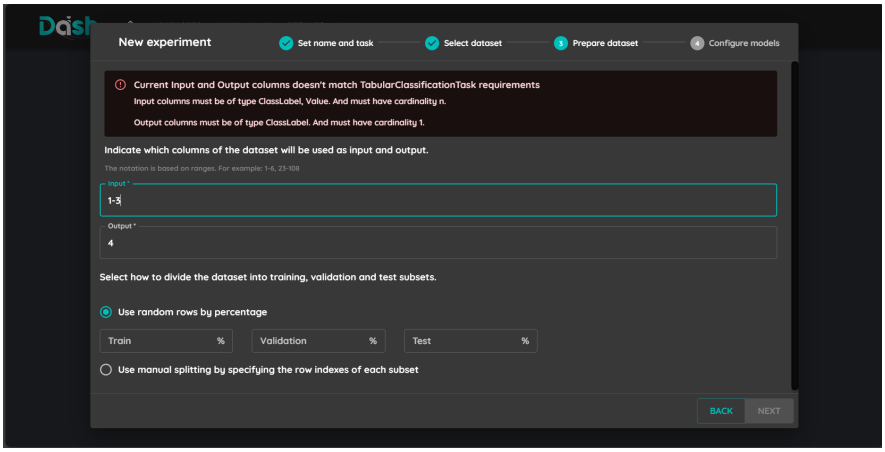

Figura 3.28: Captura de pantalla del nuevo paso para preparar el conjunto de datos.

Además, los campos para separar el *Dataset* en particiones de entrenamiento, testeo y validación también manejan errores en caso de ingresar valores que sumen distinto al 100 % ya que se debe utilizar todo el *Dataset* en el experimento.

Back-end

Para hacer efectivos los cambios, como las columnas de entrada y salida ya no son parte de los atributos de los conjuntos de datos, se agregan estos a los experimentos. Cabe destacar que es necesario que estos sean accesibles ya que estos valores son utilizados luego por los modelos. Por lo que se modificó el *end-point* de *Experiments* que ya estaba implementado en la plataforma. Las modificaciones a los *end-points* para la *URI* `/experiment/` son las siguientes:

1. **Método HTTP** `POST`
   - `/experiment/`: Este método recibe los parámetros de un nuevo experimento y lo añade a la base de datos.

     Se modificó para que pudiera recibir tanto las columnas de entrada y salida como los *splits*.

     El método puede fallar en caso de no poder encontrar el *Dataset* asociado o no poder hacer conexión con la base de datos.

Para implementar estos cambios en su totalidad, en el *back-end* se agregan métodos a la clase de *Dataset DashAI* que permiten el mapeo de índices de columnas y retorna una lista con los nombres de acuerdo a los índices entregados.

Los cambios en el flujo de creación de nuevos experimentos que fueron descritos en esta Sección permiten que el usuario suba cree un experimento, eligiendo y configurando como quiere utilizar el *Dataset*.

3.4. Refactor

La siguiente Sección contempla las distintas refactorizaciones al código de la plataforma producto de las modificaciones de los atributos de *Dataset* que fueron hechas por la memorista y parte del equipo de desarrolladores de la plataforma.

3.4.1. *DataLoaders*

Los *DataLoader* constituyen la forma en que se cargan diferentes tipos de archivos como conjuntos de datos. Estos contienen funciones que dividen el conjunto de datos en particiones, donde cada partición se convierte en un *Dataset* dentro de *DashAI* y todas las paticiones están almacenadas en un *DatasetDict*. Originalmente, la clase *DashAI* se instanciaba con atributos de columnas, los cuales fueron eliminados de su definición. Esto implica que cada partición ya no contendrá metadatos que identifiquen las columnas de entrada y salida.

La clase *DashAI* se encarga de almacenar los *Datasets* extendidos derivados de los conjuntos de datos de *HuggingFace*. Esta extensión implicaba la inclusión de la separación de

columnas de entrada y salida, guardándolas como metadatos en un archivo local. Sin embargo, esta nueva solución elimina esa funcionalidad, lo que requirió una refactorización del código en los métodos de esta clase que se encargan de guardar y cargar los conjuntos de datos.

Además, los métodos que anteriormente requerían las columnas de entrada y salida, como el de validación, ahora reciben estos datos como parámetros. Es decir, al invocar el método, se pasan estos parámetros, los que ahora se encuentran en la base de datos para cada experimento en lugar de estar guardados como metadatos en los archivos locales del *Dataset*.

En detalle, los métodos que sufrieron modificaciones para las clases de *DataLoader* y *Dataset* de *DashAI* son:

1. `to_dashai_dataset()`: Método que recibe un *DatasetDict* con el *Dataset* dividido en particiones y las columnas de entrada y salida, luego convierte cada componente del *DatasetDict* en un *Dataset* de *DashAI*.

   Es modificado ya que ya no recibe las columnas de entrada y salida.

2. `load_dataset()`: Método que recibe una ruta de conjunto de datos y carga el *Dataset DashAI* con su información y datos, entregando una copia con las columnas de entrada y salida como parte de la información.

   Se modifica ya que las columnas de entrada y salida ya no son parte del conjunto de Datos.

3. `validate_inputs_outputs()`: Este método recibe un *DatasetDict*, una lista de las columnas de entrada y una lista con las columnas de salida validando si las columnas existen en el *Dataset*.

   El método arroja error en caso de que la suma de la cantidad de columnas de entrada y salidas sea mayor a la cantidad de columnas totales o en caso de que las listas entregadas contengan columnas que no existen.

   Este método fue modificado ya que estaba implementado para que la suma de la cantidad de entrada y salida pueda ser diferente al total de columnas, lo cual ya no es necesario en el nuevo flujo donde se puede elegir distintos rangos de columnas para los experimentos.

3.4.2. *Tasks*

Las *Task* contienen métodos que validan tanto las columnas de entrada como las de salida, asegurándose de que los tipos de datos sean los adecuados y que las cardinalidades sean correctas para los modelos respectivos. Además, estos métodos se encargan de preparar el *Dataset* al seleccionar la columna con clase de salida. Dado que los valores de las columnas ya no se encuentran en el *Dataset* de *DashAI*, se ha agregado un parámetro para incluir estas columnas. Ahora, estas columnas deben ser entregadas desde los datos que almacena el experimento cada vez que sean requeridas.

Los métodos de las *Tasks* que se vieron modificados en la refactorización son:

1. `validate_dataset_for_task()`: Este método es parte de la clase de *Task* base, recibe el *Dataset* a utilizar y revisa si las columnas de entrada y salida son permitidos por la *Task* elegida.

   Fue modificada para recibir las listas de columnas de entrada y salida que ya no son parte de los atributos del *Dataset* para que pueda seguir funcionando en su totalidad.

   El método puede lanzar errores si las columnas de entrada o salida tienen un tipo de dato no soportado por la *Task*, o si la cardinalidad de las listas no está permitida por la *Task*.

3.4.3. *Runner* y modelos

El *Runner* es el encargado de procesar todos los pasos necesarios para poder entrenar los modelos de cada experimento, ya sea la preparación y validación del *Dataset* como la evaluación de métricas posterior al entrenamiento. Utiliza los modelos elegidos y configurados por el usuario en la creación del experimento y a estos modelos se les debe entregar el conjunto de datos a utilizar para entrenamiento, validación y testeo.

El entrenamiento se realiza mediante la función `fit()` que contiene cada modelo, y que prepara los conjunto de datos para ser utilizados, dejándolos en el formato que permite cada modelo, Por ejemplo, la función `fit` del modelo de *SKLearn* recibe el conjunto de datos, lo pasa a un pandas y lo divide en dos conjuntos diferentes, uno con las columnas de entrada y otro con las columnas de salida. Así utiliza estos para el entrenamiento, como se muestra en el flujo de la Figura 3.29.

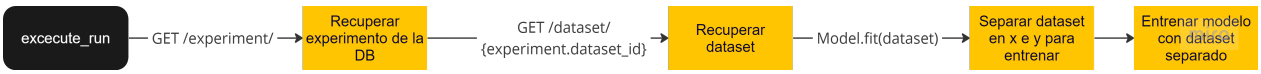

Figura 3.29: Captura de pantalla del flujo antiguo para entrenar.

Al igual que los otros componentes implicados en la refactorización, esto genera un problema ya que las columnas de entrada y salida ya no están almacenadas en los *Datasets*. Y cambiar esto requiere cambiar la firma de todos los modelo y métricas que contienen los métodos de `fit`.

La solución es dividir en conjunto de datos en dos antes de entregárselo a los métodos y luego de esto cada modelo o métrica puede manejarlo con sus componentes por separado. Para resolverlo se usa `select_columns()` de *HuggingFace*, que sirve para crear una copia del conjunto de datos solo con las columnas que se le asignan. Con esta solución el nuevo flujo del *Runner* a grandes rasgos es el presentado en la Figura 3.30.

Figura 3.30: Captura de pantalla del nuevo flujo para entrenar.

Los cambios detallados en esta Sección son claves para lograr que un usuario pueda correr el entrenamiento de sus modelos y posteriormente ver los resultados de las métricas. Sin esta refactorización las implementaciones de la Sección de Implementación de la solución 3.3.1 no hubieran sido suficientes para lograr los objetivos de desacoplamiento.

Capítulo 4

Evaluación

El siguiente Capítulo detalla como se evalúa la implementación tanto en el *back-end* con las pruebas de los métodos y en la interfaz con una simulación de un caso representativo de un usuario de *DashAI*.

4.1. Testing

Los cambios tanto de la implementación como de la refactorización se prueban por medio de tests unitarios. Los test unitarios evalúan cada función con distintas entradas, ejecutándola y verificando que los resultados de estas sean los esperados.

En el proyecto de *DashAI* estas pruebas se realizan gracias a la librería *Pytest*, que describe el estilo que debe tener cada test en lenguaje *Python*.

Se añaden test para los métodos creados en la clase *DashAIDataset*

1. Verificar que el método que divide el conjunto de datos en dos, uno con las columnas de entrada y otro con las columnas de salida para las distintas particiones de entrenamiento, testeo y validación tengan las dimensiones correctas.
2. Verificar que el método que cambia los tipos de la columna actualice efectivamente guarde los cambios asegurándose de que cada tipo de las columnas sea el mismo que se entregó en el arreglo de columnas.
3. Asegurar que al método que cambia los tipos de las columnas no pueda modificar los tipos que no se puedan cambiar por el que estaba por defecto, es decir, se asegura de que al mandar un tipo incorrecto la respuesta del método sea un error tipo *ValueError*.
4. Verificar que se le puedan pasar una o más columnas con el tipo *ClassLabel* al método que cambia los tipos de las columnas entregadas, asegurándose de que los tipos queden actualizados en el conjunto de datos que retorna el método.

5. Asegurar que el método que entrega los tipos de cada columna entregue un diccionario con las llaves siendo cada nombre de las columnas del dataset y que no contenga más elementos que esto.
6. Por último, se tuvo que refactorizar los test que incluían las columnas de entrada y salida como parte de los atributos del *Dataset* de *DashAI*.

Además, producto de la refactorización, se tuvo que refactorizar los test que incluían las columnas de entrada y salida como parte de los atributos del *Dataset* de *DashAI*. Los componentes cuyos test de *Pytest* fueron modificados por la memorista son *API*, *Tasks*, *DataLoaders*, *Models*.

Las pruebas descritas y cambios realizados comprueban que el nuevo flujo pueda realizarse sin problemas y que al usuario no le surjan errores de tipo *ValueError* o *TypeError* al correr cada experimento por medio de las *Runs* gracias al *Runner*.

4.2. Simulación usuario

Luego de probar y testear los métodos y componentes que fueron creados y modificados, se realiza una simulación de un caso representativo de un usuario donde se pueda apreciar como puede interactuar con la plataforma en su estado actual.

En esta Sección se lista la simulación utilizando la metodología *User Journey*[7], donde se específica el usuario y su contexto, la situación y las expectativas que tiene, las distintas fases que tiene que pasar para lograr y, por último, las oportunidades que surgen luego de la simulación. Con fines de simplificar la simulación es que se omiten las curvas de sentimiento y pensamientos.

4.2.1. Contexto

- **Usuario:** Francisca, *data scientist*.
- **Situación:** Francisca tiene un conjunto de datos que no conoce y le gustaría realizar experimentos con el en *DashAI*.
- **Expectativas:** Podrá subir el conjunto de datos, ver su contenido y crear dos experimentos de clasificación con distintas configuraciones de entrenamiento, validación y test.

4.2.2. Fases

Los pasos que sigue Francisca para poder lograr sus expectativas son los siguientes:

1. **Abre DashAI y se dirige a la sección de *Datasets*:**

El usuario abre *DashAI*, vista que se muestra en la Figura 4.1 y se dirige a *Datasets* para poder subir un nuevo conjunto de datos, tal como se muestra en la Figura 4.2.

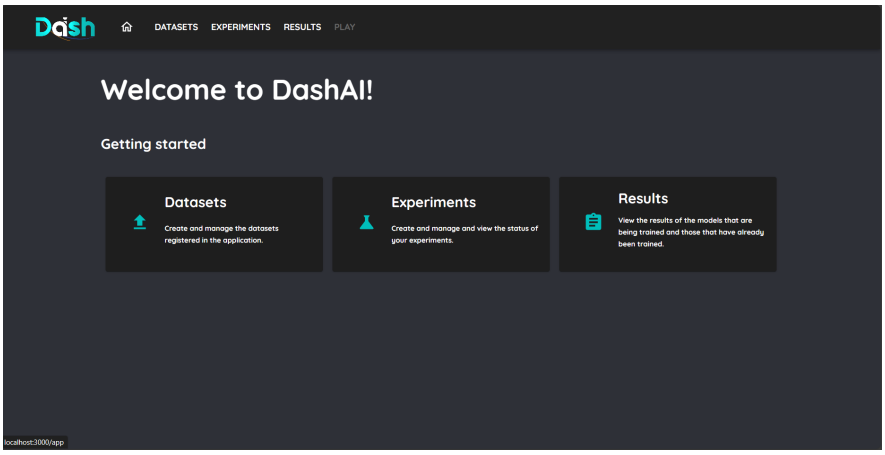

Figura 4.1: Captura de pantalla de la vista principal de *DashAI*.

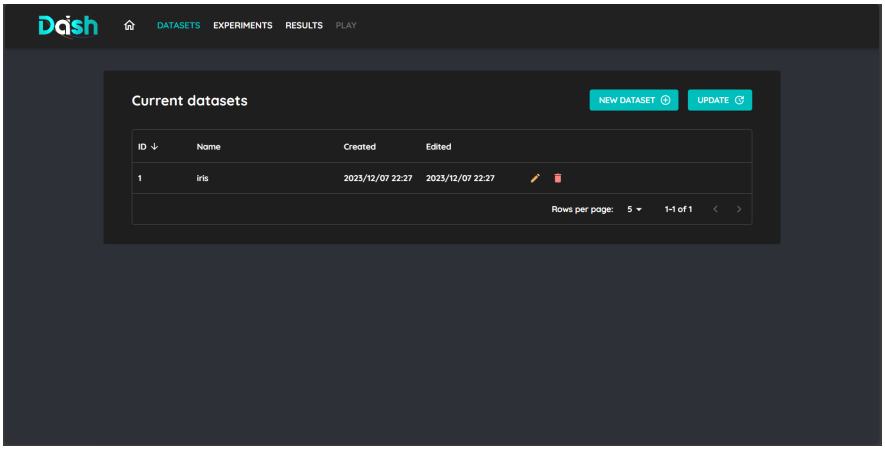

Figura 4.2: Captura de pantalla de la vista de *Datasets*.

2. **Sube su conjunto de datos siguiendo los pasos:**

   Luego el usuario abre la ventana emergente con los pasos para subir un conjunto de datos. Para esto primero elige la forma en la que quiere subir su conjunto de datos, como su conjunto de datos es un archivo *CSV* elige la opción *CSVDataLoader* como se aprecia en la imagen de la Figura 4.3.

   Luego de elegir como cargar el *Dataset* llega al segundo paso, donde sube su archivo desde sus archivos locales y lo configura, proporcionando el nombre *'nuevo_dataset'*, que se ve en la Figura 4.4.

3. **Llega a la vista de resumen y examina el contenido del conjunto de datos:**

   Luego de subir el conjunto de datos, llega al paso de resumen, aquí ve que el conjunto de datos que subió tiene 5 columnas y que los tipos automáticos dejaron las columnas de *'SepalLengthCm'*, *'SepalWidthCm'*, *'PetalLengthCm'* y *'PetalWidthCm'* con datos de tipo *double* o *float64*, y la columna de *'Specie'* como *string*. Donde además, puede

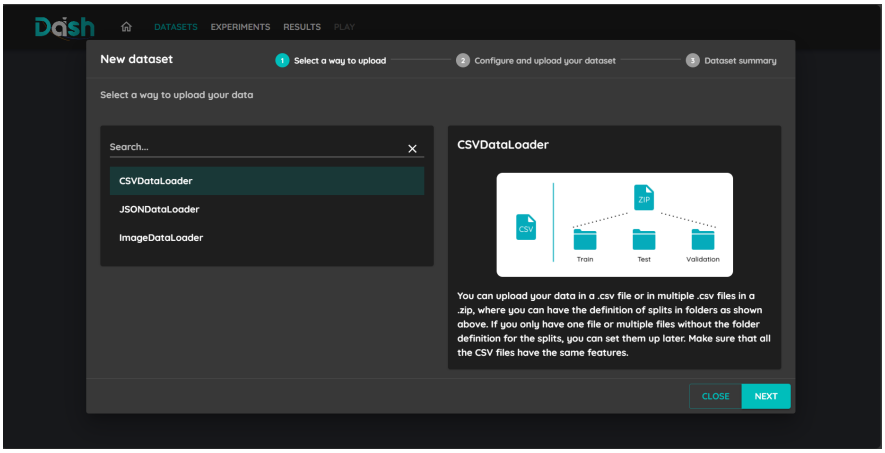

Figura 4.3: Captura de pantalla de la ventana emergente de *New Dataset* en el primer paso.

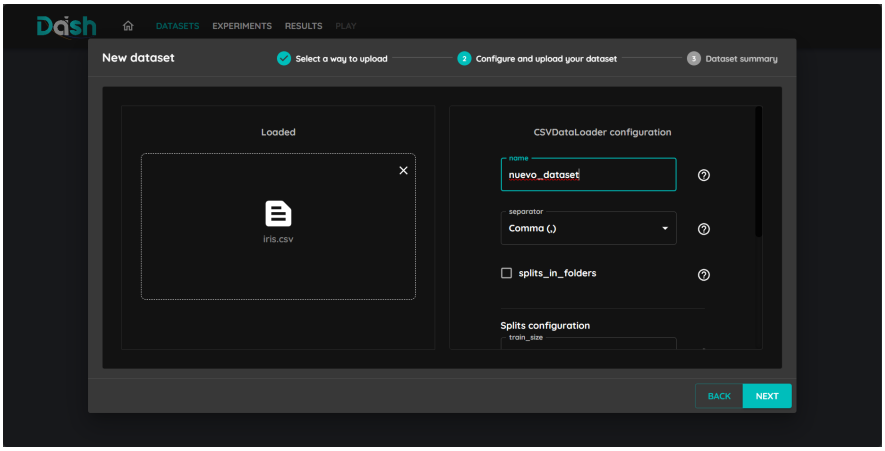

Figura 4.4: Captura de pantalla de la vista del segundo paso.

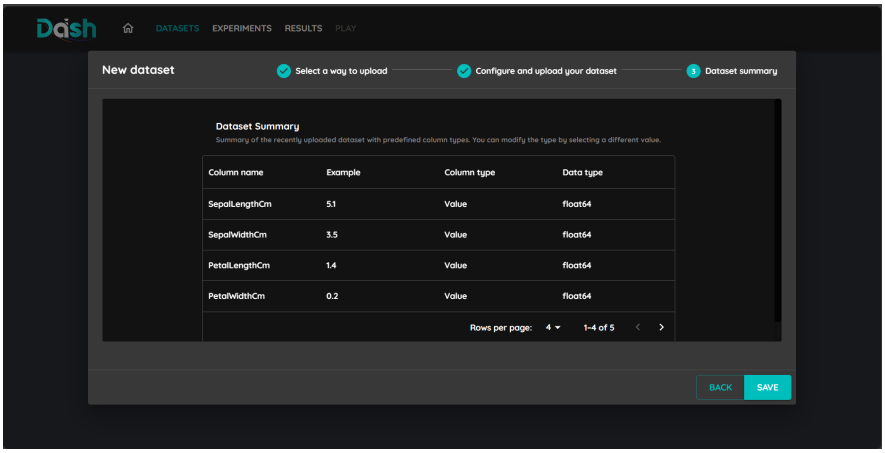

Figura 4.5: Captura de pantalla de la vista del tercer paso.

ver un ejemplo de valor de cada una. Esta tabla de resumen se muestra en la imagen de la Figura 4.5.

4. **Termina el proceso de carga y configuración del *Dataset*:**

A continuación, se termina el proceso de carga y configuración de nuevo *Dataset* y el usuario ve nuevamente la tabla, esta vez con la reciente agregación.

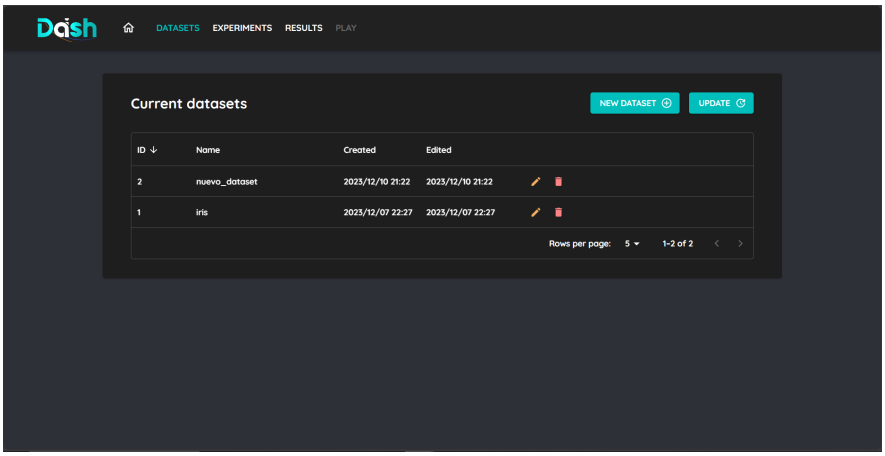

Figura 4.6: Captura de pantalla de pantalla de *Datasets* con el nuevo conjunto de datos.

5. **Se dirige a la sección de *Experiments*:**
   Ahora, el usuario quiere crear experimentos con el conjunto de datos recién subido, por lo que se dirige a *Experiments* y abre la ventana emergente para crear uno nuevo.

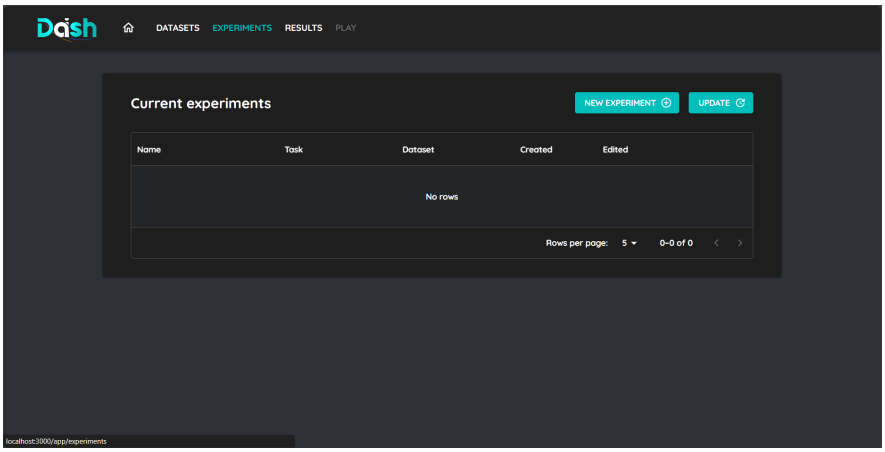

Figura 4.7: Captura de pantalla con la vista de *Experiments*.

   En el primer paso define el nombre de este primer experimento como *'nuevo_experimento'* y elige *TabularClassificationTask* como la tarea que quiere realizar.

   Luego de esto llega al segundo paso donde se despliegan todos los conjuntos de datos y elige su recién subido conjunto de datos *'nuevo_dataset'* para ser usado.

6. **Llega al paso de preparación del *Dataset* y elige como dividir el conjunto:**
   El usuario llega al paso de elegir y quiere usar columnas con tipos o cardinalidades no permitidas, la interfaz le muestra que los tipos que está ingresando no son permitidos por la *Task* y un texto que le informa qué configuraciones debería estar ingresando, como se puede apreciar en la Figura 4.10.

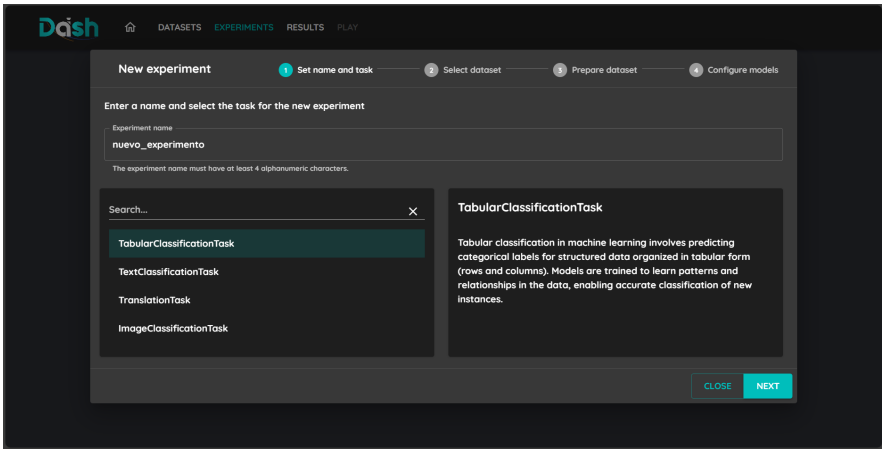

Figura 4.8: Captura de pantalla de la ventana emergente de *New Experiment* con el primer paso.

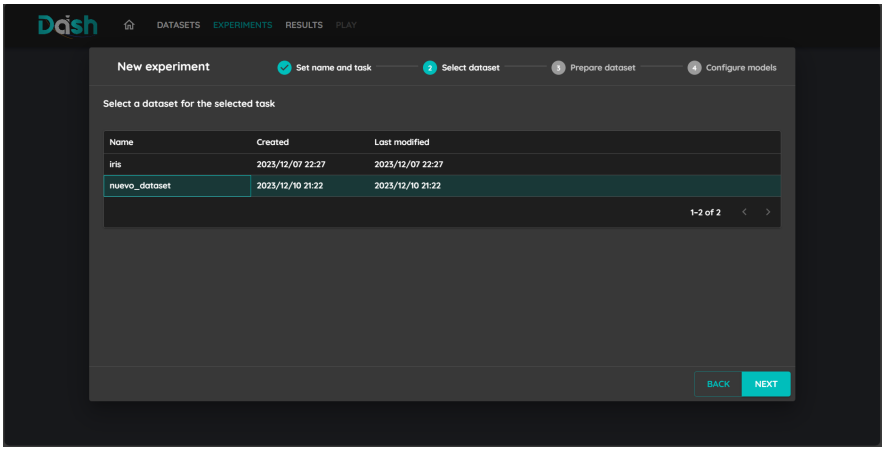

Figura 4.9: Captura de pantalla de la vista del segundo paso.

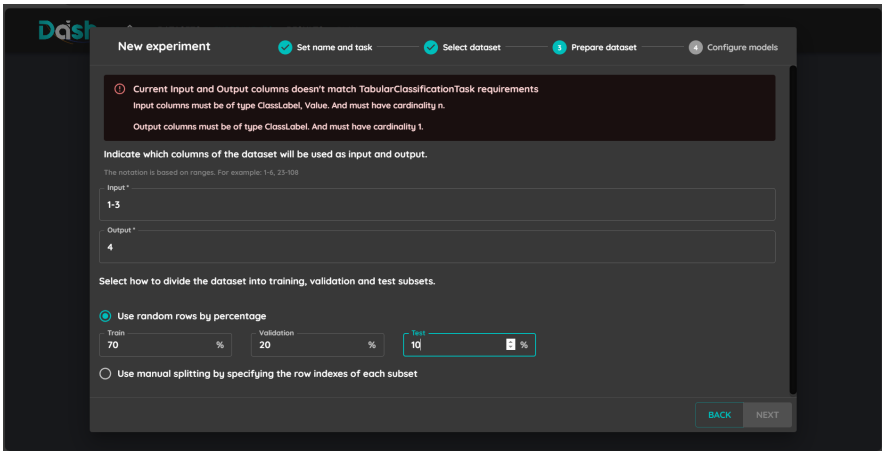

Figura 4.10: Captura de pantalla de la vista del tercer paso sin validación.

Finalmente ingresa columnas de tipos y cardinalidades permitidas como se le anuncia en la información de la sección superior de la ventana emergente (que se aprecia en la Figura 4.11) y puede avanzar al siguiente y último paso.

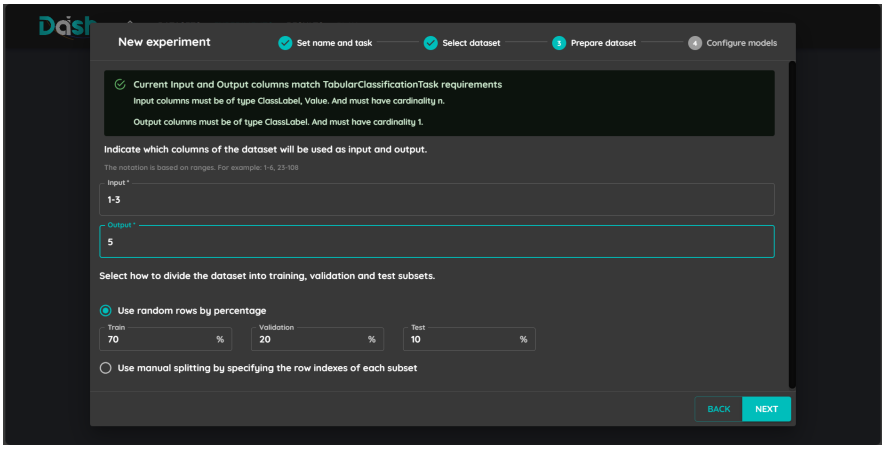

Figura 4.11: Captura de pantalla de la vista del tercer paso.

7. **Configura los modelos y termina la creación del primer experimento:**
   Ahora el usuario avanza al paso donde puede configurar sus modelos, en este paso decide agregar al experimento el modelo de clasificación *SVC*, como se puede ver en la Figura 4.12.

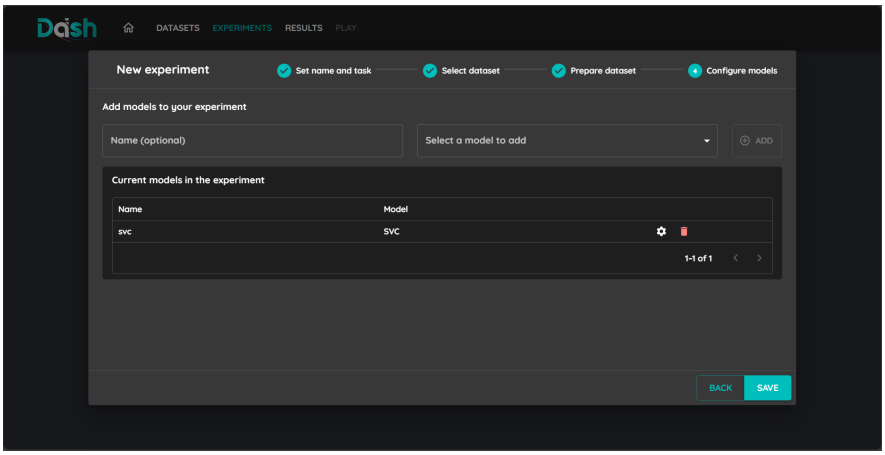

Figura 4.12: Captura de pantalla de la vista del cuarto paso.

   Por último, termina de crear el experimento con éxito y, como se aprecia en la Figura 4.13, el usuario ahora visualiza su nuevo experimento *'new_experiment'* en la lista de Experimentos donde se puede ver el estado de este.

8. **Sigue nuevamente los pasos de creación de experimentos pero divide el conjunto de datos de distinta forma:**
   Ahora, el usuario quiere crear un nuevo experimento con splits diferentes, para esto abre nuevamente la ventana *New Experiment* y realiza los mismos pasos anteriores(el primer paso se muestra en la Figura 4.14 y el segundo en la Figura 4.15), pero al llegar al paso *Prepare Dataset*, ingresa porcentajes diferentes a los anteriores para las particiones de *training*, *validation* y *testing*.

   Como se puede ver en la Figura 4.11 en el primer experimento creado el usuario elige los *splits* con particiones de 70 %-20 %-10 %, mientras que en el nuevo Experimento en la Figura 4.16, el usuario elige *splits* separando las particiones en 60 %-20 %-20 %.

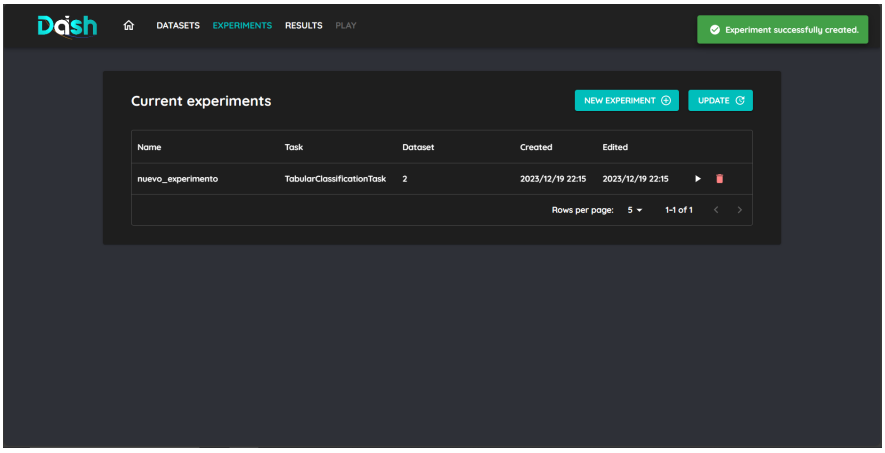

Figura 4.13: Captura de pantalla de la vista *Experiments* con el nuevo experimento creado.

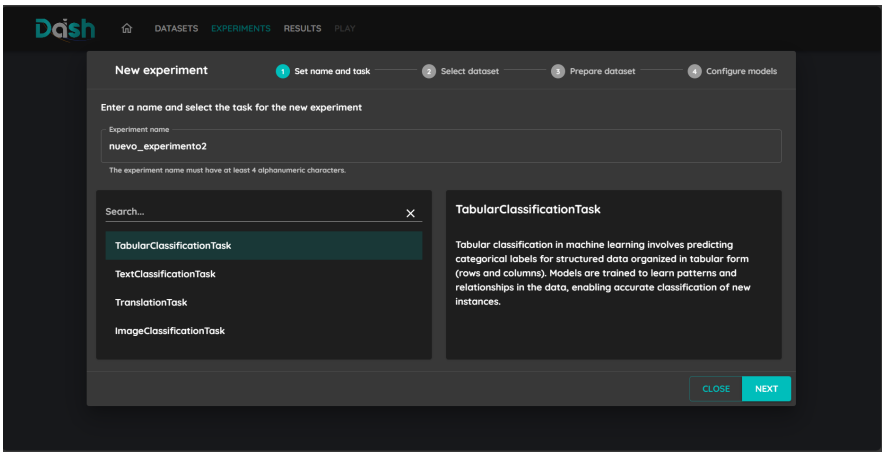

Figura 4.14: Captura de pantalla de la vista del primer paso.

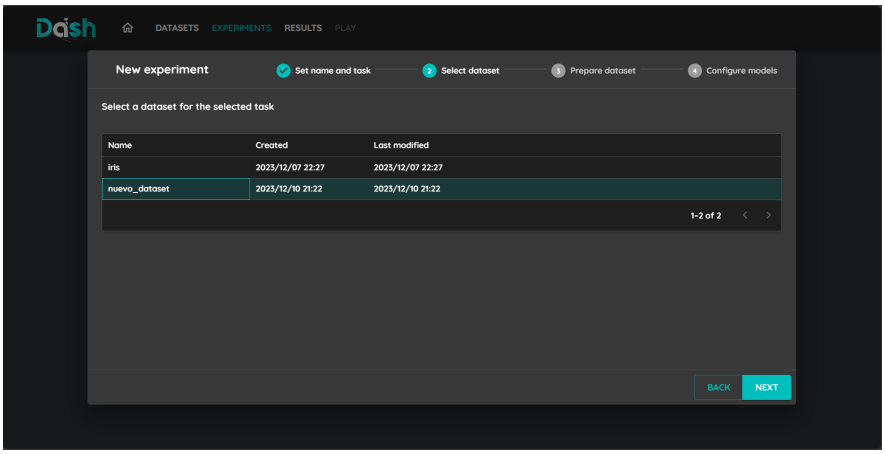

Figura 4.15: Captura de pantalla de la vista del segundo paso.

9. **Finaliza el proceso de creación de los dos experimentos:**

   Nuevamente, el usuario avanza al paso final que se muestra en la Figura 4.17, agregando nuevos modelos, en este caso Francisca decide agregar un modelo diferente *KNeighborsClassifier*.

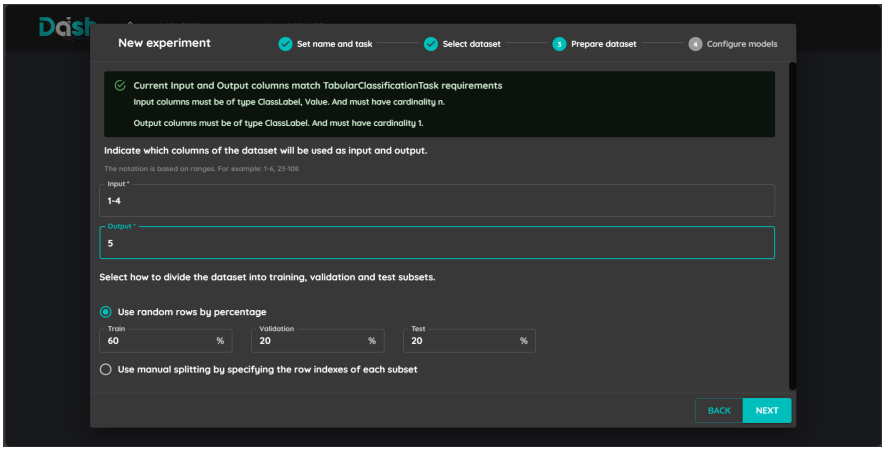

Figura 4.16: Captura de pantalla de la vista del tercer paso.

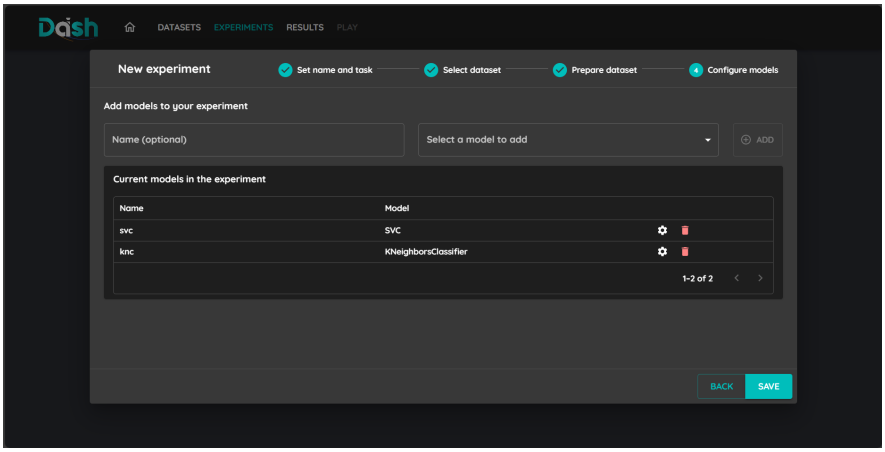

Figura 4.17: Captura de pantalla de la vista del cuarto paso.

Así termina el proceso de crear su segundo experimento, visualizando la lista actualizada con sus experimentos recién agregados y finalizando el proceso de crear dos nuevos experimentos con diferentes configuraciones pero con el mismo conjunto de datos, aprovechando de mejor manera los datos que ha recolectado y subido.

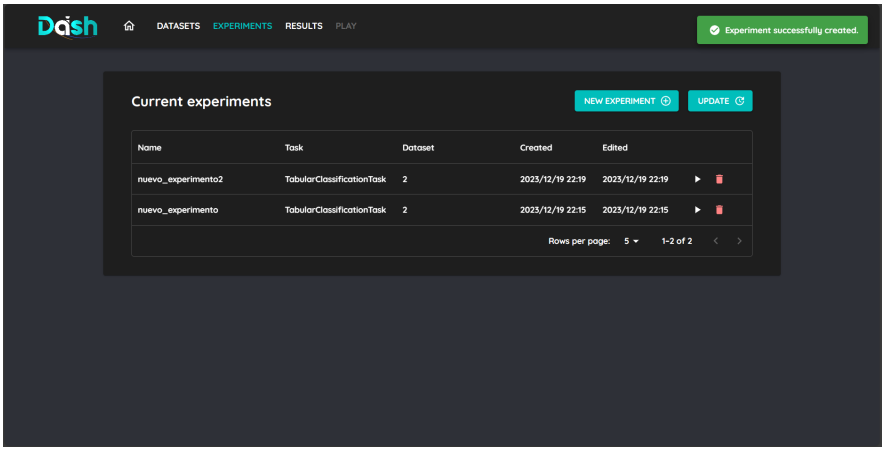

Figura 4.18: Captura de pantalla de la vista de *Experiments* con los dos experimentos creados.

4.2.3. Oportunidades

Luego de analizar las fases seguidas por el usuario surgen distintos puntos de mejora que aportarían valor al nuevo flujo implementado. Entre los cambios que se pueden realizar para entregar una mejor experiencia al usuario se encuentran los siguientes:

1. Indicar qué tipos de datos son compatibles para cambiar en el paso de resumen del *Dataset*.
2. Mejorar en la retroalimentación de los errores, específicamente en los relacionados con el cambio en el tipo de datos luego de subir los conjuntos y en la elección de las columnas al crear el experimento.
3. Agregar mayor información tanto sobre el uso de las columnas de entrada y salida como de los *splits* para que el usuario pueda comprender en qué le beneficia elegir estas configuraciones.
4. Entregar mayor información sobre que columnas, incluyendo su nombre, se están eligiendo para entrenar, luego de la elección de los rangos.
5. Proporcionar nuevas formas de explorar el conjunto de datos incluyendo estadísticas de los valores.

Capítulo 5

Conclusión

El Capítulo a continuación hablará primero del aporte al proyecto que dejan los resultados de la memoria. Luego establece las reflexiones de la memorista, incluyendo los desafíos y aprendizajes durante el trabajo realizado y, finalmente, detallará las mejoras que se pueden integrar a la plataforma para completar la solución de los problemas encontrados durante el tiempo que llevó implementar los objetivos propuestos. En resumen estas secciones buscan concluir los avances que integran esta memoria y proponer ideas para mejorar el software que se encuentra en continuo desarrollo.

5.1. Contribuciones

Las funcionalidades nuevas a las que puede acceder el usuario luego del desarrollo de la memoria son:

1. El usuario puede usar el mismo conjunto de datos para distintas *Task*.
2. El usuario puede examinar el contenido del conjunto de datos luego de subirlo.
3. El usuario puede cambiar los tipos de las columnas que fueron puestas por defecto.
4. El usuario puede crear experimentos con el mismo conjunto de datos dividido en diferentes configuraciones de columnas de entrada y/o salida.
5. El usuario puede crear experimentos con el mismo conjunto de datos dividido en diferentes conjuntos de entrenamiento, test y validación.

Las nuevas características desarrolladas aportan al objetivo general de desacoplar los conjuntos de datos a la tarea por realizar y dar la posibilidad de crear experimentos diferentes de forma autónoma.

5.2. Retrospectiva

El diseño e implementación del nuevo flujo dentro de la aplicación de *DashAI* fue una experiencia llena de desafíos, los continuos cambios relacionado al enfoque de la memoria requirieron obtener una mirada más profunda al problema existente en el *software* para resolver lo más elemental que incluían la separación de los componentes de *Datasets*, *Tasks* y *Experiments*.

Como reflexión personal, si se tuviera la oportunidad de comenzar el trabajo desde cero agilizaría los procesos de diseño, tanto las discusiones con los mock-ups como con los prototipos, ya que estos cambiaron el enfoque primordial de la memoria. Si se hubieran hecho los ciclos más rápido se podría haber ahorrado tiempo en las etapas de diseño y usarlo en mejorar la implementación de la solución o en centrar el proyecto en usabilidad del usuario.

También, se deja pendiente la realización de una validación con usuarios de los cambios realizados, esto podría entregar retroalimentación en torno a la usabilidad de los nuevos componentes y fijar los pasos a seguir para entregar una experiencia más completa y parecida a un entrenamiento que puede realizar un data scientist.

Además, se concluye que el trabajo de desacoplamiento de los conjuntos de datos con el posterior experimento, que era le objetivo general, se logró. El usuario ya no tiene que elegir una *Task* previamente sino al momento de crear cada experimento y puede hacer mayor y mejor uso de los datos ingresados a *DashAI*.

Finalmente, el desarrollo fue una experiencia muy satisfactoria para la memorista, dado que trabajar con un equipo amplio de ingenieros y memoristas que sugerían constantemente mejoras, ayudó a dejar un código más robusto y con mejor usabilidad para el usuario.

5.3. Trabajo futuro

Es competente reconocer que a pesar de que se han cumplido los objetivos planteados, también es necesario incluir mayores componentes y herramientas que ayuden a explorar, conocer y transformar los datos que el usuario sube a la aplicación y que podrían mejorar su experiencia, estas son:

- **Integración de un módulo completo para la transformación de datos:** En los proyectos de ML es importante que el data scientist pueda preparar o preprocesar los datos, esto contempla transformaciones como *data augmentation*, escalamiento, limpieza de datos nulos o blancos. Actualmente esta incorporación fue iniciada por estudiantes de trabajo dirigido que incorporaron transformaciones básicas para el módulo *Data Converter*, que aún se encuentra en desarrollo.
- **Integración de un módulo de exploración de conjuntos de datos:** Además del módulo anterior, se debe implementar un módulo que pueda entregar mayor información de sus datos al usuario, la exploración debe contener estadísticas y gráficos que le den una idea de que valores son los que contiene cada columna.

- **Acceso al resumen del conjunto de datos desde distintos puntos del flujo:** El flujo implementado contempla un paso de *Data Summary* que entrega información del contenido del conjunto de datos que ha subido y, al finalizar esta memoria, este es el único momento en el que el usuario puede acceder a este resumen. Acceder a él desde otros puntos del flujo, como la tabla de *Datasets* o al momento de preparar el dataset al crear un experimento, permitiría que el usuario recuerde el contenido cada vez que utilizará el conjunto de datos, dándole un mejor experiencia.
- **Mejoramiento en la retroalimentación de errores:** En específico en el flujo de creación de nuevos experimentos, el nuevo paso implementado de preparación del *Dataset* escogido, la elección de columnas puede fracasar por distintos motivos, ya sea el tipo de las columnas que está eligiendo no son válidos o la cantidas de columnas tanto de entrada o salida no es la correcta. El mensaje mostrado actualmente no describe con exactitud cuál es la causa del problema y, pese a mostrar los requerimientos de la *Task*, mostrar un mensaje más descriptivo del error ayudaría al usuario a identificar el problema con mayor rapidez.

Estas implementaciones, ayudarán a erradicar por completo los problemas descritos en la Sección de Problema 1.3, donde se detalla que la preparación de datos es una etapa importante en todo proyecto de Machine Learning.

Por otro lado, se hace necesario validar estos cambios con los usuarios finales, ya que su feedback es valioso para poder mejorar e integrar herramientas que sean útiles para ellos y sus proyectos. Es importante destacar que el problema presentado en esta memoria surgió de los comentarios de los mismo usuarios que probaron de forma temprana *DashAI* y que solicitaron la integración de herramientas que los ayuden a explorar los conjuntos de datos, por lo que sus comentarios son importantes para poder validar los nuevos *features*.

La validación con usuarios se puede realizar mediante test *System Usability Scale (SUS)* [3], que consiste en diez ítem de preguntas con cinco opciones de respuestas. Este test tiene los beneficios de que es fácil de administrar a los participantes y puede ser usado con pocos encuestados y aún así tener resultados confiables.

Con los componentes de trabajo futuro descritos se podrá mejorar en gran medida la usabilidad de la plataforma entorno a la exploración y uso de los conjuntos de datos subidos por los usuarios, lo que añade valor al software *DashAI*.

Bibliografía

[1] Sridhar Alla and Suman Kalyan Adari. Beginning MLOps with MLFlow. *Begin. MLOps with MLFlow*, 2021.

[2] Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*. Springer, 2006.

[3] John Brooke. SUS: A quick and dirty usability scale. *Usability Eval. Ind.*, 189, 11 1995.

[4] K. R. Chowdhary. *Natural Language Processing*, pages 603–649. Springer India, New Delhi, 2020.

[5] T. Dasu and T. Johnson. *Exploratory Data Mining and Data Cleaning*. Wiley Series in Probability and Statistics. Wiley, 2003.

[6] Rihab Feki. MLOps 01: An Introduction to MLOps levels and its life-cycle. `https://rihab-feki.medium.com/mlops-01-an-introduction-to-mlops-levels-its-life-cycle-7e9eb9a1f296`. Accessed: 2023-12-15.

[7] Sarah Gibbons. Journey Mapping 101. `https://www.nngroup.com/articles/journey-mapping-101/`, 2018. Accessed: 2023-12-19.

[8] N. Gift and A. Deza. *Practical MLOps*. O'Reilly Media, 2021.

[9] Kieran Healy. *Data visualization: a practical introduction*. Princeton University Press, 2018.

[10] Nick Hotz. What is CRISP DM? `https://www.datascience-pm.com/crisp-dm-2/`. Accessed: 2023-12-15.

[11] Steve Krug et al. Don't make me think, Revisited. *A Common Sense Approach to Web and Mobile Usability*, 2014.

[12] Tamara Munzner. *Visualization analysis and design*. CRC press, 2014.

[13] Jakob Nielsen. How Many Test Users in a Usability Study? `https://www.nngroup.com/articles/how-many-test-users/`, 2012. Accessed: 2023-04-21.

[14] Malini Patil and Basavaraj Hiremath. A Systematic Study of Data Wrangling. *International Journal of Information Technology and Computer Science*, 10:32–39, 01 2018.

[15] Erhard Rahm, Hong Hai Do, et al. Data cleaning: Problems and current approaches. *IEEE Data Eng. Bull.*, 23(4):3–13, 2000.

[16] Alex Rodriguez. RESTful Web Services. *IBM developerWorks*, 2008.

[17] Stuart J Russell. *Artificial intelligence a modern approach*. Pearson Education, Inc., 2010.

[18] Sutherland J Scrum. The art of doing twice the work in half the time. *United States of America: Crown Business*, 2014.

[19] AI Tabassam. MLOps: A Step Forward to Enterprise Machine Learning. *arXiv preprint arXiv:2305.19298*, 2023.

[20] M. Treveil, N. Omont, C. Stenac, K. Lefevre, D. Phan, J. Zentici, A. Lavoillotte, M. Miyazaki, and L. Heidmann. *Introducing MLOps*. O'Reilly Media, 2020.

[21] Tom Tullis and Bill Albert. *Measuring the User Experience: Collecting, Analyzing, and Presenting Usability Metrics*. Morgan Kaufmann Publishers, 2013.

[22] Rodrigo Urrea. Diseño e implementación de backend para framework DashAI. *Universidad de Chile*, 2023.

[23] Matthew O Ward, Georges Grinstein, and Daniel Keim. *Interactive data visualization: foundations, techniques, and applications*. CRC press, 2010.

[24] Ian H Witten and Eibe Frank. Data mining: practical machine learning tools and techniques with Java implementations. *Acm Sigmod Record*, 31(1):76–77, 2002.