UCUENCA

Universidad de Cuenca

Facultad de Ingeniería

Carrera de Ingeniería de Sistemas

Aprovisionamiento de recursos para la ejecución de experimentos basados en Machine Learning a través de las guías MLops

Trabajo de titulación previo a la obtención del título de Ingeniero de Sistemas

Autor:

David Marcelo Peñafiel Mora

Daniel Andrés Seaman Mora

Director:

Víctor Hugo Saquicela Galarza

ORCID: 0000-0002-2438-9220

Cuenca, Ecuador

2023-05-25

Resumen

Dentro de la investigación científica, más allá del conocimiento específico que se requiere para realizar la investigación, se necesita un poco de conocimiento dentro de otras áreas: Machine Learning, estadística, programación o bases de datos por mencionar algunas. Por otra parte, los flujos manuales de Machine Learning llegan a presentar diversos problemas técnicos, debido a la variedad de herramientas y profesionales que se involucran en estos procesos, especialmente al tomar en cuenta el ciclo de vida del desarrollo de Machine Learning. Esto implica que al realizar experimentación, existan problemáticas desde la preparación de los recursos y servicios necesarios para llevar a cabo este tipo de procedimientos. Debido a esto, áreas como MLOps tratan de reducir la deuda técnica en el desarrollo de aplicaciones basadas en Machine Learning, esto se realiza fomentando la automatización de procesos mediante lo que se conoce como pipelines MLOps. Bajo este contexto, en el presente trabajo se plantea la creación de un prototipo de software que facilite el aprovisionamiento de las herramientas necesarias para la creación, configuración, ejecución y versionamiento de experimentos científicos basados en Machine Learning mediante la utilización de técnicas y metodologías MLOps.

Palabras clave: aprovisionamiento, MLOps, pipelines, experimento

Abstract

Within scientific research, beyond the specific knowledge required to conduct research, some knowledge is needed in other areas such as machine learning, statistics, programming, or databases, to name a few. On the other hand, manual machine learning workflows can present various technical issues due to the variety of tools and professionals involved in these processes, especially when considering the machine learning development lifecycle. This implies that during experimentation, there may be issues starting with the preparation of necessary resources and services to carry out such procedures. Due to this, areas such as MLOps seek to reduce technical debt in the development of machine learning-based applications by promoting process automation through what is known as MLOps pipelines. In this context, this paper proposes the creation of a software prototype that facilitates the provisioning of necessary tools for the creation, configuration, execution, and versioning of scientific experiments based on machine learning using MLOps techniques and methodologies.

*Keywords:*Provisioning, MLOps, pipelines, experiments

Índice de contenidos

Índice de figuras

Índice de tablas

Agradecimientos

Quisiéramos agradecer a nuestros padres, por su incansable soporte en todo aspecto durante la carrera universitaria, sin ellos nada de esto hubiera sido posible. Además, deseamos expresar nuestra eterna gratitud por el director de nuestra tesis, Victor Saquicela, por todas sus enseñanzas y consejos que fueron aplicados a plenitud durante el desarrollo de este trabajo.
Marcelo y Daniel

1. Introducción

En la actualidad, la utilización de aplicaciones basadas en Machine Learning se ha convertido en un tema muy relevante. Generalmente el Machine Learning puede llegar a ser una pequeña parte de un ecosistema más grande de software, e incluir este tipo de tecnología dentro de un proyecto genera complejidad que antes no se conocía, o no se tomaba en consideración a la hora del desarrollo de software. Este problema fue expuesto por primera vez en un manifiesto de Google en el año 2015 (Sculley et al., 2015), en donde se mencionan problemáticas como dependencias en las entradas de los modelos o inestabilidad en los datos.

Dentro de este contexto, los flujos manuales de Machine Learning pueden llegar a presentar problemas técnicos, puesto que usualmente estos sistemas se relacionan con más productos en los que se involucran diferentes profesionales como desarrolladores, científicos de datos, entre otros. Adicionalmente, el desarrollo de un sistema de Machine Learning tiene un ciclo de vida muy diferente en cuestión de herramientas, recursos y tiempo con respecto a otros sistemas (Zhou et al., 2020). Es por esto que, áreas como MLOps buscan reducir la deuda técnica en el desarrollo de sistemas basados en Machine Learning promoviendo la automatización de los pasos involucrados en el proceso de desarrollo (Ruf et al., 2021). La automatización de cada una de las actividades se realiza mediante los llamados pipelines MLOps, los cuales se los puede considerar como un conjunto de elementos de procesamiento de datos conectados en serie donde la salida de uno es la entrada de otro, a través de la utilización de los conceptos de grafos acíclicos.

Esto implica que la experimentación para los científicos de datos puede llegar a ser compleja en el sentido de que para cada experimento se llegan a necesitar varias herramientas diferentes. Esto deriva en la dificultad de la instalación y configuración básica de todas estas herramientas para las cuales, en la mayoría de los casos, se usan de manera poco frecuente o durante cortos periodos de tiempo. Además, una vez acabada la investigación o experimentación, varios de estos servicios se mantienen en ejecución consumiendo recursos computacionales de forma innecesaria. Así también, durante procesos de investigación los flujos de Machine Learning que se manejan, por lo general se realizan de forma manual aumentando la complejidad y posibilidades de error humano en el proceso de experimentación.

Si bien existen herramientas que permiten crear pipelines MLOps que automatizan un proceso de Machine Learning, estas herramientas son poco amigables con el usuario (Zhou et al., 2020). Adicionalmente, estas herramientas asumen que se tiene toda una infraestructura con todos los servicios activos (Ruf et al., 2021), es decir que, no ofrecen un aprovisionamiento de utilidades o servicios necesarios para el usuario, esto implica que antes de iniciar con la creación de los pipelines se requiere de bastante tiempo y trabajo para que el usuario se haga con los recursos necesarios, lo cual para un usuario poco familiarizado con estas tecnologías, podría tomar mucho más tiempo que el proceso de experimentación como tal.

Basado en el contexto expuesto, en este trabajo se plantea la creación de un prototipo de software que facilite el aprovisionamiento de las herramientas necesarias para la creación, configuración, ejecución y versionamiento de experimentos científicos basados en Machine Learning mediante la utilización de metodologías MLOps, permitiendo que el usuario no requiera instalar ni configurar explícitamente las herramientas necesarias, sino que esta herramienta pueda automatizar dicho proceso y de esta forma facilitar la experimentación.

Objetivo General:

Crear un prototipo de plataforma para el aprovisionamiento de recursos que permita la experimentación científica de Machine Learning basados en las guías MLOps.

Objetivos específicos:

- Automatizar la configuración y despliegue de servicios a utilizar en un experimento de machine learning mediante Docker.
- Automatizar las fases del proceso MLOps seleccionadas en la revisión bibliográfica.
- Verificar la reproducibilidad de los experimentos mediante la comparación de resultados

El procedimiento para lograr estos objetivos se estructura de la siguiente manera: En primer lugar, en la sección 2 se presentan los conceptos teóricos necesarios para el entendimiento del proyecto, además de una revisión del estado actual de los recursos utilizados para el desarrollo de Machine Learning, concretamente aquellos empleados dentro del espacio del MLOps. En la sección 3 se presenta un escenario donde se describe la problemática de un usuario común al realizar una investigación, este escenario justifica el desarrollo de la herramienta. Continuando, en la sección 4 se describen aspectos técnicos del desarrollo del sistema, como la interfaz de usuario y la API. En la sección 5 se describe un caso de uso en donde se replica un experimento realizado de forma manual con la herramienta desarrollada y se comparan sus

resultados. Finalmente se presentan las conclusiones del proyecto junto a posibles trabajos futuros sobre este tema en la sección 6.

Al final del proyecto, se espera generar un prototipo de plataforma que provea los servicios necesarios y permita la experimentación científica de Machine Learning basándose en las metodologías MLOps.

2. Antecedentes y Trabajos Relacionados

En esta sección se introducen los principales conceptos teóricos necesarios para el entendimiento y el desarrollo del proyecto, así como un análisis del estado del arte.

2.1. Tecnología de contenedores

La tecnología de contenedores es una forma de virtualización más ligera que las tecnologías tradicionales como las máquinas virtuales, dando la posibilidad de despliegue y ejecución de aplicaciones en distintos tipos de plataformas (Casalicchio and Iannucci, 2020). Si bien existen varias tecnologías de contenedores como LXD (Containers, 2023) o Podman (Podman, 2023), la herramienta más utilizada actualmente es Docker (Docker, 2023).

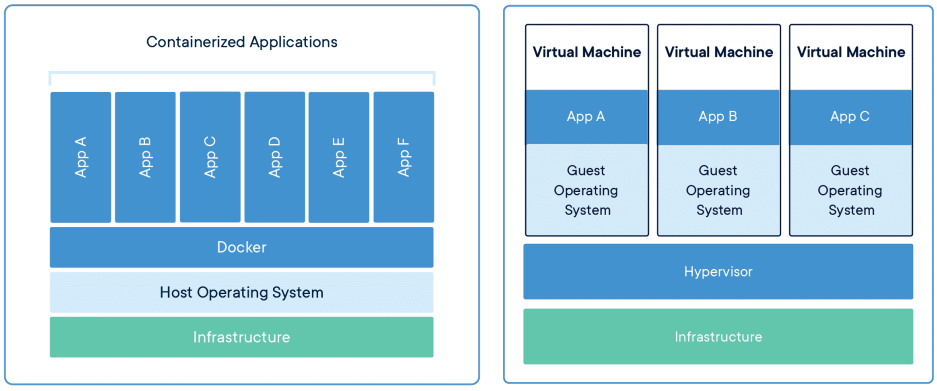
Los contenedores pueden ser comparados con tecnologías de máquinas virtuales, en el sentido de que ambas opciones tratan de ejecutar diferentes aplicaciones en ambientes aislados hasta cierto nivel. En la tabla 2.1 se muestra una comparación entre Docker y máquinas virtuales extraída de (Potdar et al., 2020). Adicionalmente en la Fig 2.1 tomada directamente de la documentación oficial (Docker, 2023) se muestra de forma gráfica la diferencia principal entre los contenedores y las máquinas virtuales, en donde estas últimas abstraen hardware mediante un *Hypervisor*[1], mientras que los contenedores corren sobre un motor común, de forma que el kernel del sistema operativo es compartido entre cada contenedor.

Los contenedores ofrecen varias ventajas frente a las máquinas virtuales, como una menor utilización de recursos, facilidad de administración y ofrecen menores tiempos de arranque comparados con las máquinas virtuales. Estas características han agilizado en gran medida el desarrollo y despliegue de aplicaciones, puesto que resulta sencillo generar un ambiente completo con todas las dependencias necesarias para diferentes tipos de proyectos, razón por la cual se han vuelto especialmente útiles para aplicaciones relacionadas con el Machine Learning en donde el manejo de dependencias y preparar ambientes de desarrollo llega a ser complejo.

[1] Un *Hypervisor* es un software que crea y ejecuta máquinas virtuales (RedHat, 2023).

	Máquinas Virtuales	Contenedores de Docker
Nivel de aislamiento de proceso	Hardware	Sistema Operativo
Sistema Operativo	Separado	Compartido
Tiempo de arranque	Largo	Corto
Utilización de recursos	Mayor	Menor
Imágenes Preconstruidas	Difícil de encontrar y administra	Disponibles para *"home servers"*
Personalizar Imágenes Preconstruidas	Difíciles de construir	Fáciles de construir
Tamaño	Mayor debido a que contiene todo un sistema operativo	Menor únicamente con Docker sobre el sistema operativo anfitrión
Movilidad	Fácil de mover entre anfitriones	Destruido y reconstruido en lugar de moverlo
Tiempo de creación	Mayor tiempo	Toma segundos

Table 2.1: Máquina Virtual contra Contenedor de Docker (Potdar et al., 2020)



(a) Arquitectura de Docker (b) Arquitectura Máquina Virtual

Fig 2.1: Comparación de Arquitecturas de Docker y Máquinas Virtuales (Docker, 2023)

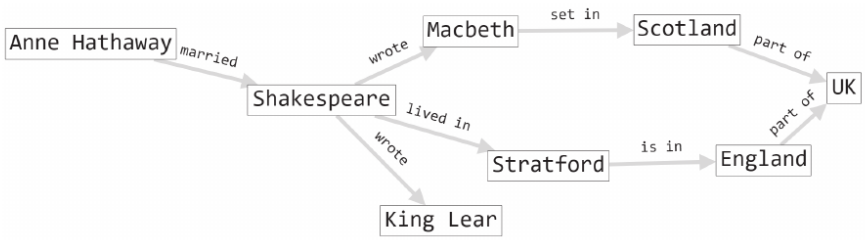


Fig 2.2: Ejemplo de representación en grafo (Allemang et al., 2020)

2.2. Ontologías

Las ontologías son el componente principal de la web semántica y tiene la función de proveer una especificación formal y explícita de la conceptualización de un dominio (Taye, 2010). Tomando como referencia a (Taye, 2010), estas representaciones están compuestas por cuatro elementos fundamentales; el *concepto*, el cual es un grupo abstracto de elementos de un dominio que representan entidades con propiedades comunes; las *instancias* dentro de una ontología representan elementos específicos de un *concepto*; una *relación* expresa relaciones entre dos conceptos específicos de un dominio; finalmente los *axiomas* son restricciones utilizadas generalmente para mantener la consistencia de una ontología.

Para representar una ontología se utiliza el estándar aprobado por W3C mediante el lenguaje formal RDF (Resource Description Framework) (Hitzler et al., 2009). RDF hace uso de tripletas, las cuales, están compuestas por un sujeto, un predicado y un objeto que permiten representar relaciones entre distintas entidades (Allemang et al., 2020). Con estos elementos es posible representar las ontologías a modo de grafos dirigidos tal y como se muestra en la Fig 2.2 recuperado de (Allemang et al., 2020). En dicha figura se pueden observar distintas relaciones expresadas como tripletas que representan conocimiento específico de un dominio, en este caso en particular, información sobre el poeta Shakespeare. Por ejemplo, de dicho grafo se puede extraer información como que *Anne Hathaway* se casó con *Shakespeare*, *Shakespeare* escribió *Macbeth, King Lear* y vivió en *Stratford*.

Si bien RDF permite representar datos en forma de grafo, no permite expresar un esquema como tal. Debido a esto surge RDFS (RDF Schema) como una extensión de RDF que habilita características como la inferencia sobre los datos (Allemang et al., 2020). Esta extensión permite dar semántica a los datos dando la posibilidad de asignar un significado a todos los elementos de la ontología (Hitzler et al., 2009).

Mediante RDFS se pueden describir diferentes *conceptos* e incluso jerarquías entre ellos tal y como se ilustra en la Fig 2.3 tomada de (Allemang et al., 2020). En este ejemplo se puede ob-

servar el *concepto Ropa de Hombre*, el cual es una superclase del concepto *Camisas*. Gracias a esta representación se puede realizar inferencias sobre los datos, como por ejemplo que, la *instancia Biker T* al ser de tipo *Camiseta* que es una subclase del concepto de *Camisas* y a su vez es una subclase del concepto *Ropa de Hombre*, por lo tanto, se puede concluir que la instancia *Biker T* pertenece al grupo de *Ropa de Hombre*. Todas estas definiciones son implementadas mediante el lenguaje OWL (*Web Ontology Language*) el cual fue diseñado específicamente para la representación de conocimiento (W3C, 2012) y es el lenguaje más usado para estos propósitos.

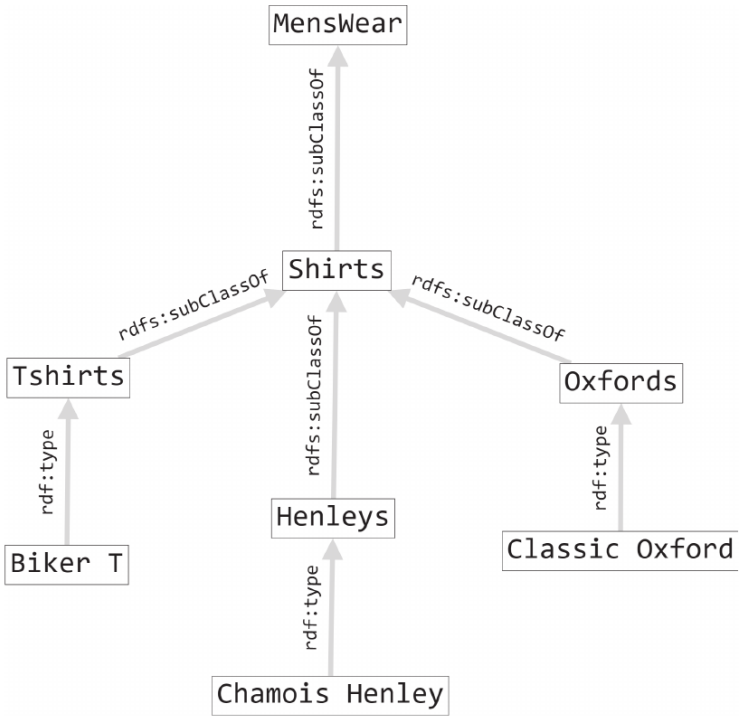


Fig 2.3: Ejemplo de RDFS (Allemang et al., 2020)

2.2.1. Ontologías para la representación de experimentos basados en Machine Learning

El uso de ontologías para representar experimentos se ha investigado durante un largo periodo y varias propuestas para representar procesos de Machine Learning han surgido con el paso del tiempo. Una buena síntesis se provee en (Sinha et al., 2022) donde se realiza una revisión sobre ontologías de Machine Learning, tomando como base: el propósito, el tipo de ontología, los enfoques de diseño y otros factores para el análisis de las propuestas recolectadas.
Dentro de la revisión presentada en (Sinha et al., 2022) se tomaron publicaciones que cumplen los siguientes criterios; publicaciones en inglés, que traten con el problema de representación

semántica de conceptos describiendo el dominio de Machine Learning, que provean la ontología expresado mediante el lenguaje OWL u otros lenguajes formales. Esta revisión toma como fecha límite en la recolección de información el mes de febrero del año 2021.

A partir de este estudio se rescatan 3 ontologías que pueden ser usadas para la representación de experimentos; La primera es MLTO (Hwang et al., 2018) la cual trata de describir un método de agrupación, utilizado para modelar el aprendizaje autónomo para la ejecución automática de procesos; Expose (Vanschoren and Soldatova, 2010) la cual trata de representar la experimentación de Machine Learning utilizando un modelo formal; Finalmente se destaca la ontología propuesta en (Tianxing et al., 2021) la cual consiste en un modelo jerárquico basado en la metodología CrispDM (Wirth and Hipp, 2000).

En el caso de MLTO la ontología puede ser utilizada para la descripción de procesos automáticos, no obstante en este estudio se optó por el lenguaje UML y no se provee alguna forma de archivo que permita reutilizar dicha ontología. Por otro lado Expose provee el archivo en OWL para la reutilización de la ontología, y aprovecha ontologías previamente desarrolladas como OntoDM (Panov et al., 2008) y EXPO (Soldatova and King, 2006), lo que permite representar de forma eficaz el proceso de experimentación. Si bien esta ontología produce una buena representación de experimentos basados en Machine Learning, la ontología no se basa en ninguna metodología para la realización de Machine Learning. Finalmente se tiene a la ontología expuesta en (Tianxing et al., 2021), en esta investigación también se reutilizan ontologías previas como OntoDM. Esta ontología provee los archivos en OWL que facilita su uso, y sobre todo, se basa en la metodología CrispDM, de forma que hace una diferenciación precisa entre las diferentes fases de un proceso de Machine Learning, permitiendo representar un experimento mediante una metodología ya probada y establecida. Debido a esto la propuesta de (Tianxing et al., 2021) se convierte en la opción más robusta.

2.3. Airflow

Apache Airflow es una plataforma de código abierto que permite la creación de *workflows*, los cuales son representados como un grafo acíclico dirigido o DAG (Directed Acyclic Graph) (Airflow, 2023). Un ejemplo de la representación de *workflows* como un DAG extraído de la documentación oficial se ilustra en la Fig 2.4. En este grafo cada nodo representa una actividad o tarea a realizar dentro del flujo de trabajo y cada vértice representa dependencias entre las distintas actividades. Airflow no está diseñado para un dominio en concreto por lo que puede ser utilizado para automatizar distintas tareas como procesamientos de datos en batch, generación de reportes automáticos o, como en el caso de este proyecto, la ejecu-

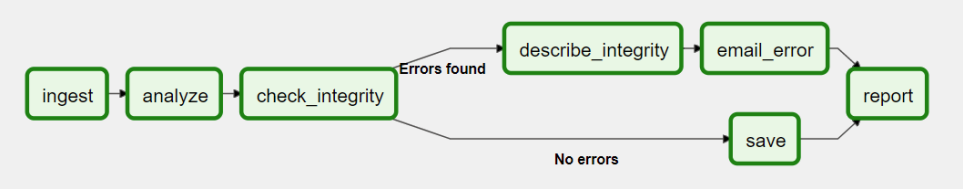


Fig 2.4: Ejemplo de *"workflow"* representado como DAG (Airflow, 2023).

ción de experimentos. Cabe recalcar que si bien esta herramienta es altamente utilizada para procesamiento de datos, Airflow se centra en la orquestación de los diferentes componentes que están involucrados en un flujo (de Ruiter and Harenslak, 2021).

2.3.1. Descripción General de Arquitectura

Airflow tiene varios componentes que entran en juego al momento de ejecutar un *"workflow"*, estos componentes son ilustrados en la Fig 2.5 extraída de (de Ruiter and Harenslak, 2021). El primero es el *scheduler* el cual se encarga de iniciar procesos programados y enviarlos al siguiente componente, el *executor* que se encarga de manejar la ejecución de cada acción en un proceso, este puede convivir en el mismo espacio que el *scheduler* y encargar la ejecución de una tarea en específico a un *worker* para computación distribuida. Adicional, se tiene un servidor web que presenta al usuario una interfaz gráfica que permite administrar de forma sencilla los flujos disponibles. También se dispone de una base de datos de metadatos utilizado por el *scheduler*, el *executor* y cualquier *worker* que se tenga a disposición. Finalmente se tiene una carpeta en donde residen los archivos de definición de los DAGs.

2.3.2. DAGs en Airflow

Cada pipeline, workflow, o DAG, está compuesto por varias tareas y sus respectivas dependencias. Como se mencionó anteriormente, esto dentro del DAG es representado como los nodos y los vértices del grafo respectivamente. Las acciones se encargan de cada parte del *workflow*, ya sea para la obtención de datos, procesos de análisis, iniciar acciones en otros sistemas, entre otras funcionalidades, y son ejecutadas dependiendo del orden y las dependencias definidas en el DAG (Airflow, 2023). Este modo de representar las acciones y sus dependencias presenta ventajas como la clara separación de tareas independientes que pueden ser ejecutadas de forma paralela, de manera que se aprovechan mejor los recursos, además de que se pueden definir nuevas tareas de forma incremental sin interferir con aquellas tareas ya existentes.

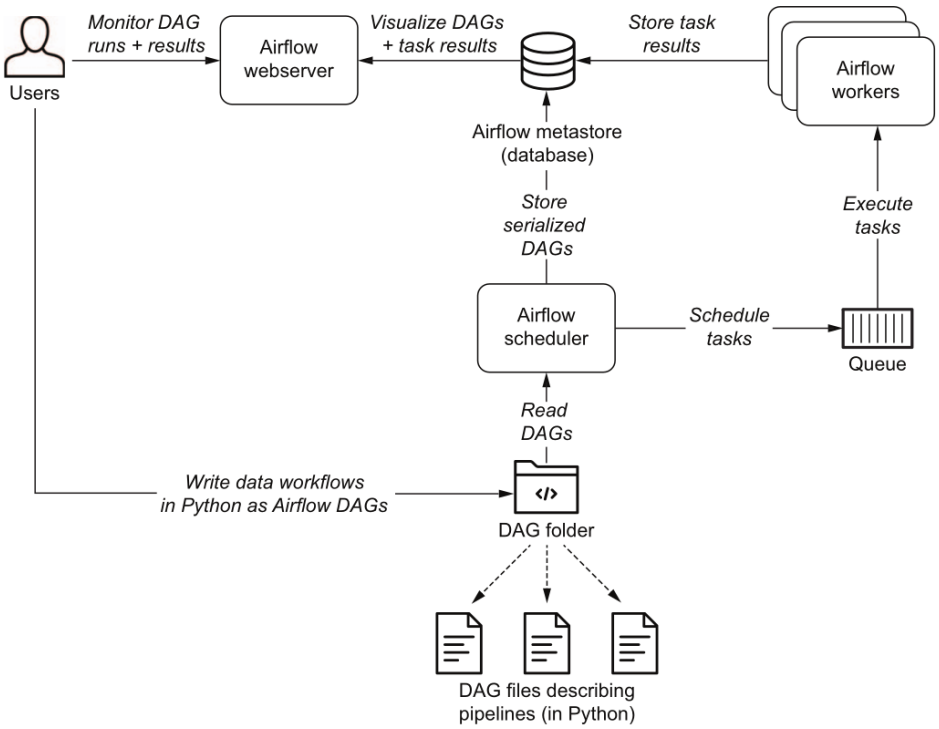


Fig 2.5: *Vista general de la arquitectura de Airflow (de Ruiter and Harenslak, 2021)*

La definición de cada DAG se realiza mediante código de Python, este tipo de definición resulta flexible puesto que Airflow es agnóstico a la ejecución de las tareas definidas en cada paso. (Airflow, 2023). Sin embargo, tal y como lo dice (de Ruiter and Harenslak, 2021), esto puede representar un reto tanto para leer o para probar las diferentes tareas. Adicional, cada script de Python permite definir metadatos de los DAGs, de manera que Airflow puede saber cómo y cuando el DAG en cuestión debe ser ejecutado.

Una vez se tiene los DAGs definidos, el proceso para ejecutarlos resulta relativamente sencillo y es ilustrado en la Fig 2.6 extraída de (de Ruiter and Harenslak, 2021). Se puede observar como a partir del script de Python que define el DAG, el *scheduler* genera el DAG junto a sus dependencias y produce los cronogramas correspondientes según los metadatos descritos en el script. Posteriormente, el *executor* envía las diferentes tareas a los distintos *workers*, los cuales ejecutan cada nodo del DAG y envían los resultados a la base de datos. Finalmente, el servidor web monitorea el progreso y los resultados de la ejecución para que el usuario pueda hacer seguimiento mediante la interfaz web.

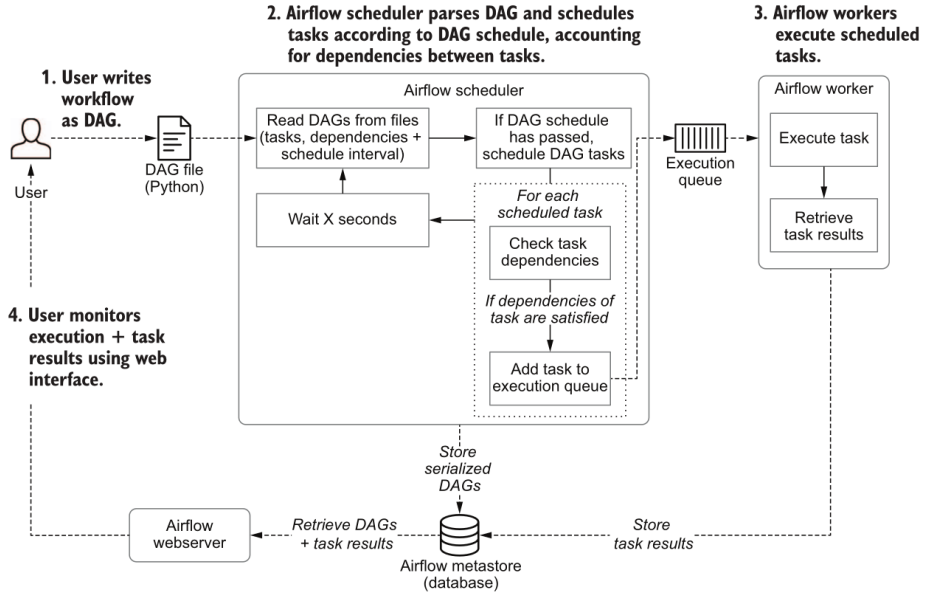


Fig 2.6: Esquema de desarrollo de un DAG (de Ruiter and Harenslak, 2021)

2.4. DevOps

Con el surgimiento de las metodologías ágiles para el desarrollo de software, metodologías más clásicas como la “cascada” quedaron obsoletas. Este tipo de metodologías ofrecieron varias ventajas, como cambios inmediatos a los requerimientos de forma más sencilla en caso de ser necesario. Dentro de DevOps entran dos grupos que trabajan en equipo para lograr un mismo objetivo, por un lado, el equipo de desarrollo se encarga de diseñar y escribir el programa, nuevas funcionalidades y cambios, mientras que el equipo de operaciones realiza las acciones necesarias para entregarlo al cliente, en una fecha determinada (Ruf et al., 2021). Por lo tanto ”*DevOps es un movimiento cultural, una forma de pensar*” (Salvucci, 2021), donde lo que se busca es que el equipo de desarrollo y el de operaciones trabajen juntos, con el objetivo de que las fases de las metodologías sean implementadas de una forma más veloz, donde las nuevas funcionalidades lleguen de manera más rápida (o inmediata) al cliente (Ruf et al., 2021). Según (Felipe and Maya, 2016) esto se logra con el uso en conjunto de dos importantes prácticas, CI (Continuous Integration o Integración Continua) y CD (Continuous Delivery o Entrega Continua). CI es una práctica donde varios miembros de un equipo integran el código muy frecuentemente (Shahin et al., 2017); mientras que el CD busca asegurar que la aplicación esté lista para producción luego de realizarse las pruebas correspondientes (Shahin et al., 2017). El uso de estas dos prácticas tiene como objetivo hacer que el producto final llegue al usuario final lo más rápido posible.

En la Fig 2.7 propuesta por (Salvucci, 2021) se puede ver las etapas que abarca el ciclo de vida

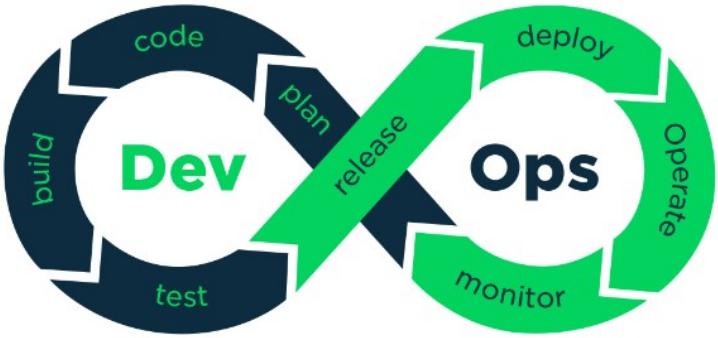


Fig 2.7: Ciclo de vida de DevOps

de DevOps las cuales son; Planificación,Codificación, Construcción, Pruebas, Lanzamiento, Despliegue, Operaciones y Monitoreo.

2.5. MLOps

Según (Ruf et al., 2021) "*MLOPS se puede pensar como una intersección entre el Machine Learning, la ingeniería de Datos y DevOps*". En otras palabras: es integrar y aplicar las prácticas DevOps a proyectos de Software que contengan componentes de Machine Learning. Construir un modelo de Machine Learning es una tarea que por lo general toma varios pasos: Los datos necesitan ser recopilados, analizados y preprocesados, y el modelo necesita ser entrenado y validado. Estos pasos son incrementales, realizados comúnmente de forma manual por un científico de datos. En MLOps además de CI y CD, se incluye el CT (Continuous Training o Entrenamiento Continuo), dado que en la gran mayoría de los casos existirá la necesidad de reentrenar un modelo de Machine Learning, no una sino probablemente varias veces (Ruf et al., 2021). El equipo de Operaciones se encargará de *desplegar, monitorear y administrar el modelo en producción* (Ruf et al., 2021).
Según (Salvucci, 2021), MLOps tiene varios niveles de automatización los cuales son:

- **Nivel 0:** Proceso Manual (Sin automatización)
- **Nivel 1:** Automatización del Pipeline de Machine Learning
- **Nivel 2:** Automatización de CI / CD (Automatiza el CI/CD de cada parte del sistema)

Teniendo en cuenta que este proyecto no toma en cuenta fases de integración con sistemas en producción, únicamente se abarcará la automatización de nivel 1, automatización del Pipeline de Machine Learning.

2.5.1. Ciclo de vida de MLOps

El ciclo de vida de Machine Learning puede ser definido de varias formas como se establece en (Salvucci, 2021) donde indica que el ciclo de vida de un proyecto de Machine Learning puede variar de acuerdo al dominio en que se encuentre el problema, no obstante también se establece que todos los proyectos de Machine Learning comparten etapas comunes que se presentan a continuación: Extracción de datos, exploración de datos, ingeniería de atributos, construcción del modelo, validación del modelo, despliegue del modelo, monitoreo y finalmente el feedback. No obstante no todos los proyectos cumplen con todo el ciclo de vida por lo que dichas etapas se suelen tomar como una guía.

Un ejemplo en donde se evidencian, de forma parcial, las etapas mencionadas es en (Hewage and Meedeniya, 2022) en donde el ciclo de vida de Machine Learning parte definiendo que el dato es lo que determina toda la efectividad de un modelo de Machine Learning, debido a esto, es importante que un dataset sea preprocesado de forma que aquellos atributos que no aportan información sean limpiados y eliminados. Esto último puede entrar dentro de las etapas de preprocesamiento de datos e ingeniería de atributos. Una vez procesados los datos a utilizar, se puede pasar con el tuneo de hiper-parámetros antes del proceso de entrenamiento. Adicionalmente, la validación y pruebas del modelo son importantes para mantener un registro del rendimiento del modelo. Finalmente, una vez el modelo cumple con el rendimiento deseado, dicho modelo es desplegado donde CD (Continuous Delivery) es aplicado.

En (Kreuzberger et al., 2022) se presenta un proceso de MLOps el cual se divide en diferentes etapas como: Inicialización de proyecto MLOps, donde se analiza el problema a resolver, se define la arquitectura y tecnología a usar y se busca entender los datos iniciales y conectar los datos para un primer análisis; Luego, se tiene el pipeline de ingeniería de características donde se limpia los datos, se los transforma, se eliminan atributos innecesarios y se crean nuevos según se requiera. Finalmente se tiene el pipeline de experimentación, donde se tiene el entrenamiento, la validación y la exportación del modelo. En este último paso, el proceso de exportar el modelo puede ser realizado aplicando un procesos de CI/CD (Continuous Integration/Continuous Delivery).

Por otra parte, en (Zhao, 2020), se define de manera muy simple un pipeline MLOps que contiene las siguientes etapas para un proyecto de Machine Learning: Requerimientos del modelo, recolección de datos, limpieza de datos, etiquetado de datos, ingeniería de características, entrenamiento del modelo, evaluación del modelo, despliegue y monitoreo del modelo. Cabe recalcar que dentro de esta definición, en las etapas de entrenamiento y monitoreo se puede

volver a alguna etapa anterior según se requiera durante el desarrollo del proyecto.

Dentro de (Ruf et al., 2021) se definen las etapas de una manera un poco más detallada: Primero se tiene la fase de ingeniería de requisitos del proyecto, luego la fase de administración de datos donde se enfoca en garantizar la calidad de los datos incluyendo la calidad de los procesos de la gente encargada. La siguiente fase es de preparación de Machine Learning que incluye procesos como la adquisición de datos, limpieza de datos, etiquetado de datos y versionamiento de los mismos, en donde diferentes datos producen diferentes resultados. El proceso de versionamiento de datos resulta útil cuando los datos evolucionan con el tiempo, ya sea en cuestión de registros o la aparición de nuevos atributos. Aquí se puede recalcar una gran ventaja de mantener el versionamiento de datos, la cual es, que permite mantener un registro de la evolución de los datos a lo largo del tiempo. Lo siguiente es la fase de entrenamiento que incluye la validación inicial del modelo junto con el versionamiento del mismo; este paso es importante dado que los modelos tienen fuerte interdependencia entre los datos, y el procedimiento de modelado como tal, por lo que, diferentes versiones del modelo pueden ser requeridos en diferentes escenarios, dando como resultado que este proceso ayude a desplegar el modelo correcto dependiendo de las circunstancias. Finalmente, se concluye con la fase de despliegue, que involucra el despliegue inicial del modelo, la integración con las demás partes del sistema, el monitoreo y la inclusión del modelo dentro de un flujo CI/CD.

2.5.2. Herramientas para administración de MLOPS

MLOps es una temática de investigación relativamente nueva y por lo tanto la cantidad de estudios revisados por pares son reducidos como se menciona en (Kreuzberger et al., 2022). Esto es consistente con lo que se menciona en (Recupito et al., 2022) en donde, de 60 recursos analizados en el estudio, 51 de estos recursos provienen de fuentes no revisadas por pares tales como publicaciones en Blogs, videos, páginas web de desarrolladores y repositorios en Github. No obstante el mismo estudio muestra un incremento en la cantidad de investigación y desarrollo con el paso del tiempo en donde más de la mitad de la información recolectada data del año 2020. Por otra parte, si bien existe investigación que busca establecer bases teóricas en esta área, el interés del presente estudio se centra más en las herramientas utilizadas en MLOps.

Si bien MLOps no busca imponer una serie de herramientas específicas para su desarrollo (Salvucci, 2021), las herramientas que se elijan deben ser eficaces de forma que se puedan cumplir con todos los objetivos del proceso. Debido a esto es necesario realizar un análisis de los diferentes instrumentos que se tienen a disposición para la realización de MLOps. Varias

de estas herramientas han sido revisadas en estudios anteriores (Hewage and Meedeniya, 2022; Recupito et al., 2022; Salvucci, 2021) por lo que a continuación en la sección 2.5.3, se presentan de forma condensada los resultados obtenidos de estos análisis, en donde se discuten herramientas genéricas de software que han demostrado ser útiles dentro del ámbito de MLOps además de analizar diferentes alternativas disponibles dedicadas específicamente para el desarrollo en esta área. Seguido, en 2.5.4, se presentan herramientas que si bien no están centradas en MLOps, pueden ser utilizadas para el desarrollo de Machine Learning. Finalmente, 2.5.5 se discute de forma general sobre el uso de las herramientas mencionadas y las posibles complicaciones y limitaciones que se pueden presentar.

2.5.3. Herramientas para MLOps

Dentro de estas herramientas se encuentran piezas de software de contenedores como Docker (Docker, 2023), el cual, como se discutió previamente en 2.1, permite ejecutar aplicaciones dentro de un ambiente estandarizado, desacoplado de la infraestructura de hardware y asilado reduciendo los tiempos de despliegue significativamente, o Kubernetes (Kubernetes, 2023) el cual funciona como un framework de orquestación que permite administrar de forma automática grandes cantidades de aplicaciones, que se encuentren en forma de contenedores de Docker en diferentes ambientes de forma distribuida. En combinación, estas dos herramientas permiten implementar los diferentes pasos del proceso de MLOps en diferentes ambientes.

Para orquestar los diferentes componentes dentro de un pipeline MLOps se tiene a Airflow (Airflow, 2023), la cual es una plataforma de código abierto que permite desarrollar y monitorizar flujos de trabajo a modo de grafos acíclicos de forma programática (Recupito et al., 2022). Otra alternativa de código abierto para la automatización de trabajos es Jenkins (Jenkins, 2023) el cual puede ser definido como un servidor de automatización (Salvucci, 2021) y puede ser utilizado para orquestar un pipeline MLOps basado en eventos, por ejemplo, un commit en un repositorio.

Todas las herramientas mencionadas son paquetes de software maduros y que se han utilizado en varios ámbitos de forma efectiva, y a su vez han demostrado ser de gran utilidad en el área tanto de Machine Learning como de MLOps, no obstante, al ser herramientas genéricas no son capaces de proveer ayudas o comodidades específicas para el desarrollo en estos campos, es por eso que poco a poco se han ido generando herramientas específicas para el Machine Learning y MLOps.

El desarrollo de herramientas para MLOps va en aumento gracias al crecimiento en el interés de esta área. En este aspecto existen varias plataformas y herramientas basadas en cloud

mantenidas por grandes compañías tecnológicas como lo son Amazon, Google o Microsoft. Dentro de esta categoría, se tiene a plataformas como AzureML (Microsoft, 2023), desarrollada por Microsoft. Esta es una herramienta basada en cloud que permite implementar cada una de las fases de MLOps mediante su SDK (Software Development Kid). Por otro lado se tiene a Vertex AI (Google, 2023) desarrollada y mantenida por Google con características similares a las encontradas en AzureML. Adicional esta Valohai (Valohay, 2023) el cual ofrece propiedades similares a las herramientas desarrolladas por Google y Microsoft.

Cabe recalcar que las herramientas mencionadas anteriormente son herramientas propietarias y con ello el costo asociado puede evitar la utilización del software en cuestión. Sin embargo, la comunidad de código abierto ofrece alternativas que buscan extender el uso de MLOps.

Dentro de este tipo de herramientas se encuentra Kubeflow (Kubeflow, 2023). Kubeflow es un proyecto de código abierto que busca simplificar el flujo de trabajo de Machine Learning. Esta herramienta está desarrollada sobre Kubernetes y trata de explotar las capacidades de este último, al mismo tiempo que vuelve escalables a los modelos de Machine Learning desarrollados y simplifica su despliegue. Si bien este artefacto está compuesto de varios componentes, los más utilizados son los ”Kubeflow Pipelines” (Salvucci, 2021) que permiten definir un flujo MLOps mediante un DAG (Direct Acyclic Graph).

Por otra parte, así mismo como un proyecto de código abierto, se tiene a MLflow (Mlflow, 2023), que consiste en una API que *Permite integrar los principios MLOps dentro de un proyecto de Machine lerning con mínimos cambios al código existente* (Alla and Adari, 2021). Los componentes a destacar de este software son; MLFlow Tracking, el cual permite la reproducibilidad, automatización y comparación de diferentes experimentos; MLFlow Project, que permite monitorear el ambiente de ejecución de un experimento; MLFlow Models, un formato estándar para empaquetar modelos de Machine Learning; MLFlow Model Registry, el cual funciona a modo de un control de versiones de un modelo y finalmente componentes dedicados a la entrega que permite integrar un modelo entrenado en un sistema en producción.

Adicional, existen otras herramientas diferentes a las mencionadas que permiten la implementación de MLOps como TensorFlow Extended, AWS SageMaker, Polyaxon, MLReef y ClearML las cuales tienen características similares al resto de artefactos de software ya mencionados.

Finalmente vale la pena repasar dos herramientas de utilidad al momento de desarrollar MLOps, la primera es DVC (Data Version Control) (DVC, 2023) que, según su documentación, ayuda a la administración de grandes cantidades de datos de forma colaborativa haciendo la función de software de versionado para modelos de Machine Learning, flujos de trabajo y conjuntos de

datos sobre repositorios Git (Git, 2023). La segunda herramienta es Seldon Core (SELDON, 2023) el cual es usado principalmente para el despliegue de modelos de Machine Learning en Kubernetes.

2.5.4. Otras herramientas de Machine Learning

Dentro de las alternativas disponibles para Machine Learning se encuentran herramientas como RapidMiner (RapidMiner, 2023), Orange (Orange, 2023) o KNIME (Knime, 2023). Estos artefactos se centran en la programación visual, es decir, a diferencia de la mayoría de herramientas para Machine Learning como Tensorflow o Pytorch, en las que se desarrollan modelos mediante código, utilizando estos paquetes se puede definir un programa mediante bloques visuales que se encuentra interconectados y definen un flujo de información, estos tres programas mencionados anteriormente utilizan grafos acíclicos para representar el programa y finalmente generar un modelo.

Por otra parte, un artefacto de software útil para el desarrollo de Machine Learning es Weka (Weka, 2023) el cual, según la documentación oficial, es un paquete de software que provee implementaciones de algoritmos de aprendizaje que pueden ser aplicados de forma sencilla a un dataset. Si bien este software permite definir pipelines de forma similar a Orange, KNIME y RapidMiner, el soporte es limitado por lo que esta característica no es muy utilizada.

2.5.5. Análisis del estado actual de las herramientas

Una vez expuestas las herramientas disponibles queda en evidencia que el manejo de todos los recursos necesarios que entran en juego dentro de un proyecto de Machine Learning de forma manual resulta en un reto bastante grande. Esto se confirma en (Kreuzberger et al., 2022) en donde se afirma que una automatización eficaz es clave en estos procesos, especialmente cuando se desea asegurar ”reproducibilidad” y se toman en consideración recursos como los conjuntos de datos, modelos y código.

En este contexto, cada una de las herramientas mencionadas anteriormente satisfacen con una o varias de las necesidades en cierta medida, sin embargo, la solución más completa según (Recupito et al., 2022) es MLFlow el cual es limitado en áreas como el almacenamiento y monitoreo de los datos y ofrecer de alguna manera los modelos producidos a modo de API. Además, a pesar de que MLFlow logra atacar de forma eficaz la mayoría de lo inconvenientes a los que uno se enfrenta en MLOps, su utilización puede verse entorpecida debido a la necesidad de implementarse mediante código, a diferencia de otras herramientas como Orange,

KNIME o RapidMiner que definen procesos de forma gráfica.

Por otra parte, algo que se asume en la mayoría de este tipo de herramientas es que existe toda una infraestructura sobre la que se puede aplicar MLOps, esto puede dificultar en gran medida el desarrollo de este tipo de proyectos debido a que se requiere configurar previamente todos los servicios necesarios. Como se mencionó previamente en la sección 1, por norma general esta configuración se realiza de forma local, lo cual puede obstaculizar en gran medida el desarrollo de un flujo de trabajo MLOps. Herramientas como Orange o RapidMiner no sufren de esta problemática, no obstante no logran simular de forma precisa el infraestructura que se puede presentar en un proyecto en producción y utilizar un modelo desarrollado en estas herramientas representa un costo adicional.

Esto implica que existe la necesidad de automatizar el aprovisionamiento de recursos y, que de preferencia, sean sencillos de remover cuando no se requieran a diferencia de instalar todos los servicios de forma local. Hasta donde se sabe, el reto del aprovisionamiento de recursos enfocado al MLOps es explorado por primera vez en (Miñón et al., 2022). En este proyecto denominado Pangea, se confirma una falencia en las herramientas de MLOps al no dar facilidades para desplegar los distintos servicios requeridos para el proceso. Para solucionar esta problemática, en este estudio se propone un sistema en donde se utilizan tecnologías como Ansible (Ansible, 2023) y Terraform (Terraform, 2023) de forma que se puedan gestionar los recursos y la infraestructura de hardware de forma automática. En este estudio se centran principalmente en el aprovisionamiento de ambientes heterogéneos como Cloud, Fog y Edge no obstante también se ofrece un modo denominado *modo de pruebas* en donde el pipeline MLOps es desplegado mediante Docker-Compose. Si bien este estudio ataca a la problemática de forma eficaz en varios aspectos, tal y como lo mencionan los autores, se requiere más desarrollo para dar soporte a un mayor rango de casos de uso además del ya presentado en el estudio, un caso aplicado a una mina en donde se hace uso de datos recolectados mediante diversos sensores, además de ser pensado principalmente para ambientes de Cloud, Fog y Edge y no para ambientes experimentales. Por otra parte, tiene falencias en aspectos como el versionamiento de conjunto de datos, modelos y pipelines en general. Adicional, esta herramienta requiere de la utilización de un lenguaje denominado PADL para la especificación del pipeline.

Con este análisis quedan en claro ciertas falencias en las herramientas existentes para MLOps y experimentación. En primer lugar, no existen alternativas sencillas que permitan desplegar todos los recursos necesarios para automatizar un proceso de Machine Learning. Por otro lado, en ninguna de las herramientas revisadas permiten expresar un experimento con todos sus

componentes y dependencias de una forma consistente. Además la mayoría de herramientas requieren el conocimiento de algún tipo de lenguaje específico como es el caso de MLFlow o el proyecto Pangea. Debido a esto, la presente investigación trata de suplir dichas falencias centrándose principalmente en ambientes locales de prueba y trata de mejorar la forma en la que se definen los experimentos ofreciendo interfaz web, abstrayendo la utilización de un lenguaje específico que determine los recursos necesarios y las dependencias entre ellos.

3. Escenario

Considere el caso de un investigador dentro del área social. Este usuario posee poca experiencia en el uso de herramientas informáticas. Suponga que este usuario desea realizar una investigación sobre un tema cualquiera, para lo cual probablemente requiera la creación de una encuesta, seleccionar una muestra, realizar la encuesta a la muestra que seleccionó, tabular la encuesta y finalmente analizar los datos. Dependiendo de lo que se quiera analizar, posiblemente una hoja de cálculo sería suficiente, sin embargo, si el problema requiere aplicar un algoritmo de Machine Learning, y con ello, sin que este usuario sepa, deba aplicar una métodología como CrispDM, empiezan las dificultades.

Por ejemplo, si este usuario desea reproducir el experimento planteado en (Pinzon, 2022), donde se aplica el algoritmo SVM al dataset Iris. Este ejemplo en concreto implica que este usuario deberá instalar Python, junto a todas las librerías y dependencias necesarias. En el caso de que adicionalmente requiere que el dataset se encuentre almacenado en una base de datos, necesita instalar y realizar las configuraciones básicas de algún gestor de base de datos, como por ejemplo PostgreSQL. Ambas tareas pueden resultar en un considerable nivel de dificultad para este tipo de usuario. Un problema adicional se presenta en que puede ser posible que se requiera hacer la experimentación una sola vez, por lo tanto lo más probable es que el usuario no remueva las herramientas instaladas anteriormente, por lo que los servicios de cada artefacto consumirán recursos de forma innecesaria.

Si el usuario usó la base de datos para almacenar el dataset, necesitará un conocimiento básico del lenguaje de consulta SQL para poder obtener la información con el objetivo de procesarla con el lenguaje de programación. En caso de que el usuario desee cambiar el código presentado en el ejemplo (Pinzon, 2022) puede resultar en una tarea sumamente compleja y frustrante, debido a la poca experiencia con las herramientas informáticas, su conocimiento de Machine Learning será nulo, o por lo menos extremadamente básico. En el caso de querer reproducir algún otro experimento, puede no describir correctamente lo que quiere realizar, dejando información valiosa sin descubrir, o produciendo resultados no relevantes o redundantes dado su poco conocimiento del lenguaje y las librerías a usar. Finalmente en caso de que el entrenamiento se haya llevado a cabo con éxito, requiere esfuerzo adicional para realizar un

despliegue del modelo exitoso en pro de predecir nuevas instancias.

Para usuarios con este perfil que no poseen los conocimientos suficientes y que no disponen de personal especializado en informática para que les ayuden en estas situaciones, los problemas anteriormente mencionados pueden llevar al usuario a situaciones de frustración, mucho estrés, y posiblemente incumplir los plazos de investigación o entrega lo que puede significar costes extras en retrasos en organizaciones como universidades.

El hecho de querer reproducir un experimento, puede ser algo sumamente complejo. En el caso de uso que se presentará más adelante en esta tesis, el investigador tardó más de un día completo reproduciendo su propio experimento tiempo después en un nuevo equipo en donde el 90% del tiempo se utilizó únicamente en preparar el ambiente de ejecución que incluye instalación del lenguaje, librerías requeridas, y solucionar problemas de compatibilidad con las nuevas versiones de las librerías utilizadas, o incluso con actualizaciones del sistema operativo.

Por todo lo expuesto, la necesidad de una herramienta con la capacidad de proveer un ambiente de ejecución listo para usar, en donde sea sencillo para un usuario con conocimientos básicos de experimentación automatizar un pipeline de Machine Learning, y donde tiempo después pueda reproducir sus propios experimentos con un mínimo esfuerzo es evidente. Además, la creación de un repositorio de experimentos donde los investigadores puedan consultar y reproducir experimentos se vuelve un tema necesario.

4. Diseño e Implementación de la herramienta

El proyecto tiene tres objetivos específicos; se requiere automatizar la configuración y despliegue de los servicios necesarios que se vayan a utilizar en un experimento. Se busca automatizar cada una de las fases de un experimento de acuerdo a las guías MLOps y finalmente se busca que cada proceso sea reproducible.

En primera instancia se deben definir cuales son las fases de un experimento que se buscan automatizar. Para esto se toma como base a las fases de MLOps que fueron exploradas en la sección 2.5.1. Como se mencionó en dicha sección, las fases más comunes en MLOps incluye pasos como; extracción de datos, exploración de datos, ingeniería de atributos, construcción del modelos, validación del modelo, despliegue del modelo, monitoreo y retroalimentación como parte final del proceso, adicional a estas fases se suelen incluir pasos como entendimiento del problema a resolver no obstante no es una etapa estándar.

Puesto que este proyecto se centra en la experimentación, fases como el entendimiento del problema, monitoreo y retroalimentación no se van a considerar al momento del diseño y desarrollo, por otra parte la fase de ingeniería de atributos cumple la función de procesamiento de datos de forma que las fases que se toman en cuenta para el proyecto son las siguientes; extracción de datos, exploración de datos, procesamiento de datos, construcción del modelos, validación del modelo y despliegue. Con respecto a la fase de despliegue, puesto que la investigación se centra en ambientes locales se propone que esta fase consista únicamente en poner a disposición los modelos generados mediante una API REST mas no un despliegue en producción.

Por otro lado, el aprovisionamiento de servicios a utilizar en un experimento es uno de los procesos que también se busca automatizar siendo esta una fase que se debería considerar previa a la extracción de los datos. Finalmente, en favor de poder automatizar todas las fases mencionadas se necesita partir de una descripción del experimento la cual contenga todos los pasos a seguir y los recursos necesarios para la ejecución del mismo incluyendo servicios y datos. Con todos estos pasos en mente,las fases finales que se consideran dentro del presente trabajo se ilustran la Fig 4.1 en donde todo comienza por la descripción de un experimento seguido de una etapa de aprovisionamiento y continuando con fases mas convencionales en

un proceso MLOps.

Todo el proceso comienza con la fase de *Definición de experimento*, en esta fase se determina cual es la secuencia de pasos a seguir en el experimento y donde se definen las dependencias y recursos necesarios para el mismo, la segunda fase, *Aprovisionamiento de recursos* consiste en desplegar todos los servicios necesarios para llevar a cabo el experimento descrito en la fase anterior. Las fases de *Extracción de datos, Exploración de datos, Procesamiento de datos, Construcción de modelos y Validación de modelos* ya fueron explicadas en la sección 2.5.1 pero a breves rasgos, estas fases se encargan de todo el procesamiento de los datos, desde la extracción de los mismos, hasta la generación y validación de los modelos. Finalmente la fase de *Despliegue* únicamente se realizará de forma local exponiendo los modelos generados mediante una API REST como se mencionó anteriormente.

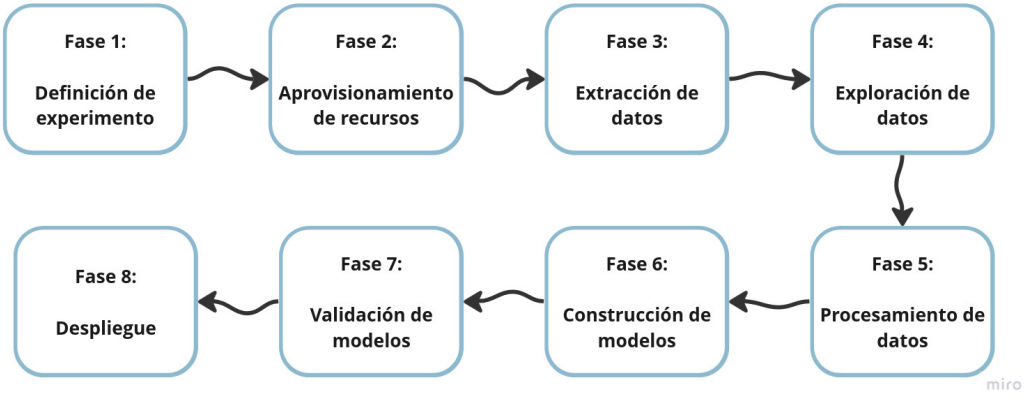


Fig 4.1: Fases consideradas para el proyecto

Una vez definidas las fases a seguir, a continuación se describen todas las consideraciones tomadas para cumplir con los objetivos del trabajo.

La representación de los experimentos es clave, puesto que a partir de la forma en la que se describe un experimento se pueden asegurar características como la reproducibilidad, permitir determinar los recursos empleados para un experimento y habilitar la posibilidad de automatizar todo el proceso. Dicha representación debe incluir la mayor cantidad de aspectos posibles como por ejemplo: todos los pasos que se deben realizar para obtener el resultado deseado. Teniendo en cuenta esto, es indispensable que cada fase a realizar durante el experimento sea descrita detalladamente especificando cada tarea de forma individual, por ejemplo, la lectura de datos, codificación de una columna o agrupaciones de datos, de forma que cada actividad a realizar represente como tal un algoritmo o proceso en específico. Esto implica que al expresar una tarea se deben incluir los parámetros del algoritmo, sus respectivas entradas y salidas, junto a los metadatos de estas, como por ejemplo, el tipo de dato que debe tener cada una de

las columnas de un conjunto de datos antes y después de aplicar un determinado algoritmo, además del ambiente en el que se va a ejecutar una tarea en concreto.

Una vez se tiene una representación de los experimentos robusta, se puede utilizar la misma información para determinar el tipo de servicios que se requieren para posteriormente desplegarlos y configurarlos. Las configuraciones de cada servicio deben asegurar todas las dependencias necesarias para poder completar cada etapa del proceso mediante la generación de una infraestructura adecuada. Por otra parte, para automatizar cada fase definida en el experimento se requiere que el sistema sea capaz de orquestar toda la infraestructura generada de forma que la única interacción requerida por el usuario sea la definición previa del experimento que se desea realizar. Finalmente se pretende que se pueda mantener un registro de las diferentes versiones tanto de un experimento como de los conjuntos de datos utilizados.

4.1. Representación de experimentos

La forma en la que se representan los experimentos es crucial para este proyecto puesto que en función de como se represente un experimento se determina el como cumplir los objetivos del proyecto. Lo primero es obtener una forma completa de representar los experimentos de Machine Learning, esto debe incluir posibilidades de definir cada una de las operaciones a realizar dentro del experimento, los parámetros de dichas operaciones, ambiente de ejecución, dependencias y datos a utilizar, junto a metadatos que permitan describir estos últimos, con el fin de asegurar la reproducibilidad de un experimento mientras que cada operación descrita en el experimento corresponda a una de las fases elegidas para este trabajo. Por otra parte, puesto que los experimentos no necesariamente cumplen con todas las fases consideradas en el proyecto, se necesita que el modelo utilizado sea flexible de forma que no sea imprescindible cumplir con todo el ciclo de vida definido en este proyecto. Estas consideraciones se pueden cumplir al representar la información a modo de grafo debido a las múltiples relaciones que se pueden crear durante una experimentación. Con este tipo de representaciones en mente, se opta por la utilización de una ontología cumpliendo que el modelo sea representado mediante un grafo y que el modelo a utilizar sea flexible. Por otra parte, la posibilidad de utilizar razonadores con la ontología aprovechando las posibilidades de inferencia de este tipo de tecnologías podría facilitar el descubrimiento de nuevo conocimiento y disminuir la cantidad de información que se debe definir por adelantado. Cabe recalcar que el uso de una ontología, como consecuencia, genera un repositorio de experimentos que luego puede ser aprovechado, no obstante la utilización de esta información queda fuera del alcance de este proyecto por lo que esta posibilidad se analiza a breves rasgos en los trabajos futuros en la

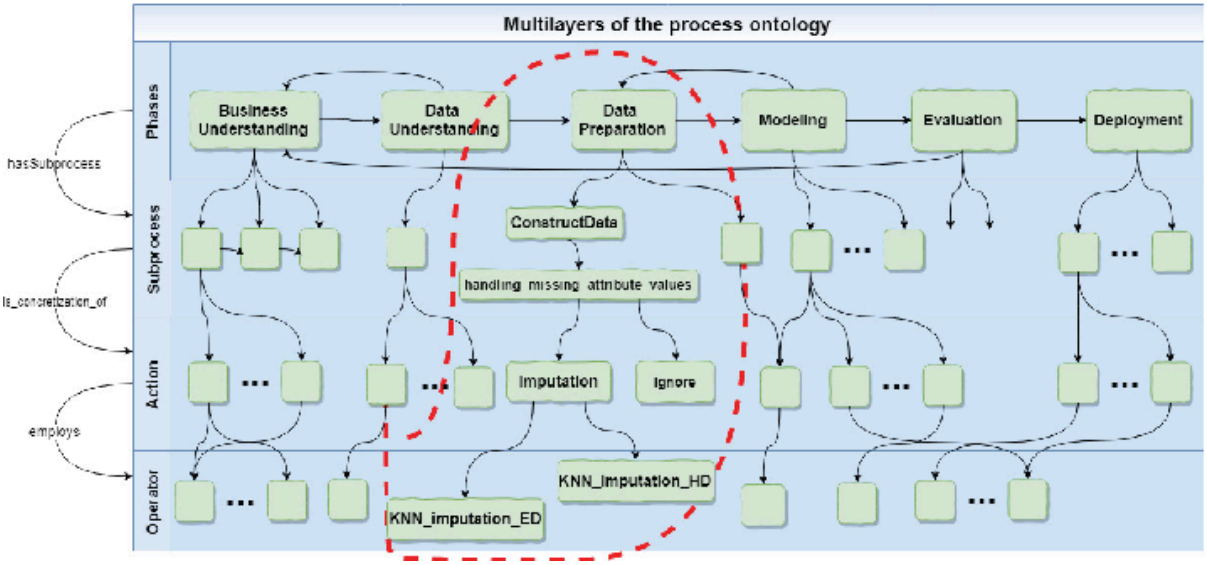


Fig 4.2: Ontología base para la representación de experimentos (Tianxing et al., 2021).

sección 6.

Puesto que MLOps sigue siendo un tema relativamente nuevo en investigación, al momento de desarrollar este proyecto, no se han encontrado ontologías que representen experimentos basándose en MLOps. Por otra parte definir una ontología desde cero queda fuera del alcance del proyecto por lo que se opta por utilizar una ontología basada en la metodología CrispDM (Wirth and Hipp, 2000) siendo esta última una metodología madura y establecida para proyectos de Machine Learning. De manera muy general, CrispDM aborda las siguientes etapas; entendimiento del negocio, entendimiento de los datos, preparación de los datos, modelamiento, evaluación y despliegue haciendo que tenga cierto símil con las etapas de MLOps que fueron revisadas en la sección 2.5.1. Puesto que CrispDM no coincide por completo con las guías MLOps, cada una de las fases descritas en la ontología cumple con una o varias de las etapas MLOps descritas anteriormente de forma que no se puede explotar todas las características que puede ofrecer una ontología, no obstante, con algunas modificaciones se puede aprovechar gran parte del modelo haciendo que pueda cumplir su rol para la representación de datos. Debido a esto, se tomó como base a la ontología descrita en (Tianxing et al., 2021) siendo esta, la propuesta más actual y más robusta según se discutió en la sección 2.2.1. La estructura de la ontología se ilustra en la Fig 4.2 extraída de (Tianxing et al., 2021).

El modelo comprende varios niveles de abstracción en donde el nivel más alto hace referencia a una fase en concreto de la metodología CripsDM mientras que el nivel más bajo implica especificar con precisión un algoritmo en concreto a aplicar durante el proceso. En este trabajo únicamente se toma el nivel más bajo de la ontología debido a que el uso de razonamiento sobre la ontología queda fuera del alcance del proyecto, además el uso de capas superiores

resulta innecesario puesto que basta únicamente con el nivel más bajo que describe a los algoritmos a utilizar para representar en su totalidad a un experimento según los objetivos del proyecto. A la ontología seleccionada finalmente se le realizan ligeras modificaciones que permiten representar el experimento MLOps en su totalidad. Entre estas modificaciones se incluyen las clases *Enviroment* y *Parameter* de forma que se puede preservar el tipo de ambiente en el que se ejecuta un algoritmo y los parámetros utilizados para el mismo asegurando la reproducibilidad. Parte de la ontología final se muestra en la Fig 4.3.

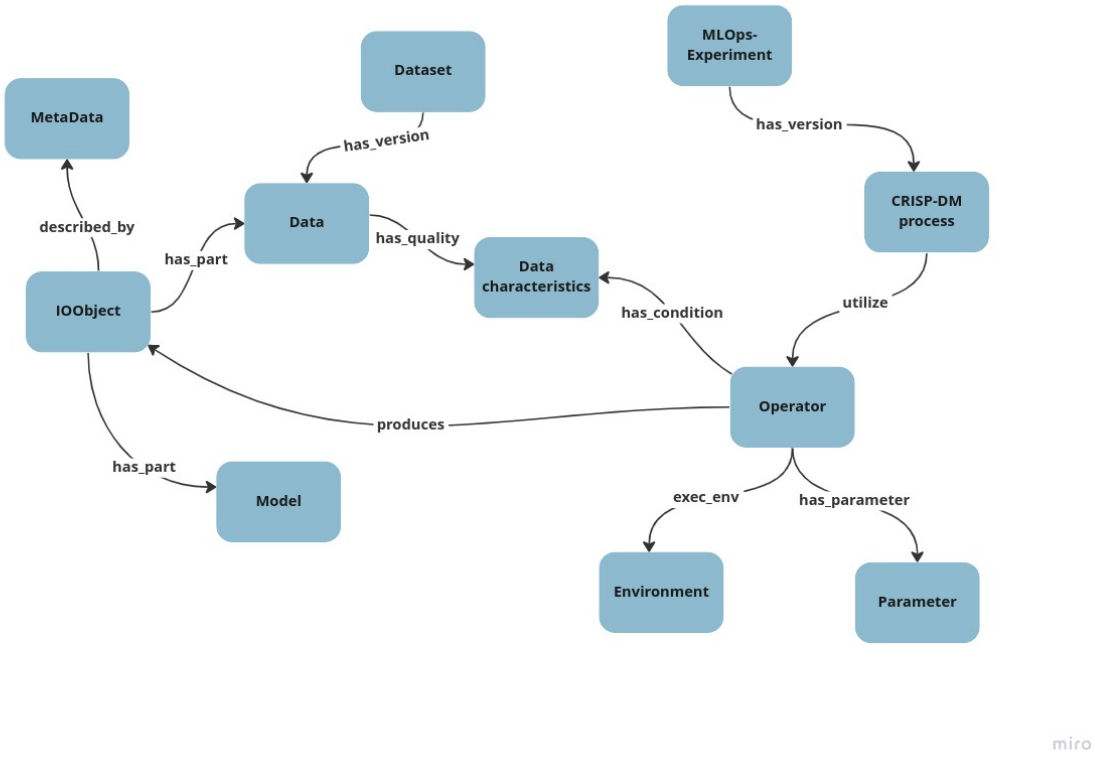


Fig 4.3: Ontología para la representación de experimentos.

A continuación se explica de forma breve la ontología para entender como representar un experimento y como se busca dar soporte para características como el versionamiento de experimentos y conjuntos de datos. El punto de partida para entender el modelo es la clase *MLOps-Experiment* la cual toma un rol parecido al de un proyecto relacionándose con diferentes versiones de un proceso, mas no es la definición del proceso que se desea ejecutar como tal. Lo siguiente es la clase *CRIP-DM process* la cual representa una versión en concreto de un proceso. Seguido se tiene a la clase *Operator* que representa a un algoritmo específico o tarea que se utiliza en un flujo concreto. Cada operador puede tomar como entrada o salida a entidades de la clase *IOObject* sean modelos de Machine Learning o Datos los cuales

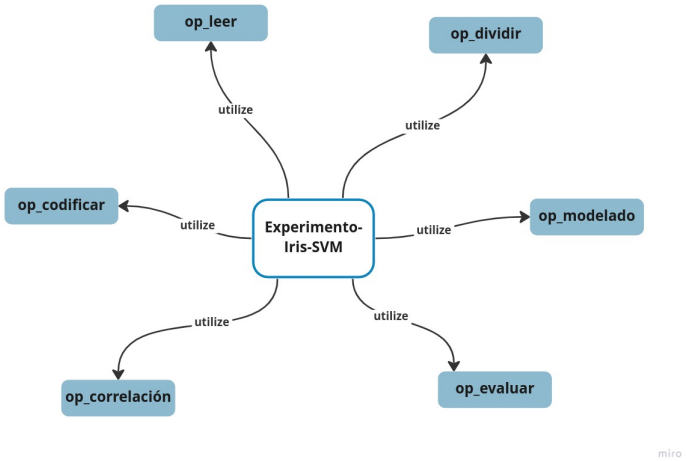


Fig 4.4: Representación básica de experimento.

pueden ser descritos mediante los *Metadatos* dentro la misma ontología. Por otra parte, la clase *Parámetro* representa los distintos parámetros que puede utilizar un operador concreto. Adicionalmente se tiene a la clase *Ambiente* la cual indica el recurso o ambiente que utiliza el experimento para ejecutar el operador en cuestión. Estos ambientes hacen referencias a los servicios que entran en juego dentro del experimento, es decir, son servicios como ambientes de Python o bases de datos como PostgreSQL. La ontología completa se la puede revisar en el *repositorio del proyecto*[1].

4.1.1. Ejemplo de uso de Ontología

Para ilustrar el modo de uso de la ontología para representar un experimento se tomó el ejemplo detallado en en (Pinzon, 2022), en donde se describe la realización de un proceso de clasificación mediante el algoritmo SVM (Support Vector Machines) sobre el dataset Iris (FISHER, 1936).

El proceso que se toma como ejemplo se divide en 6 pasos, lectura de datos, codificación de dataset, análisis de correlación, división de dataset, creación de modelo y evaluación de modelo. Tomando en cuenta esto, en las Fig 4.4 y Fig 4.5 se muestra todo el proceso representado mediante la ontología.

En la Fig 4.4 se observa como se relaciona el experimento *Experimentp-Iris-SVM* con cada uno de los operadores empleados en el mismo. Dichos operadores toman el nombre de op_leer, op_codificar ,op_correlación, op_dividir, op_modelado y op_evaluar los cuales representan cada uno de los pasos en el experimento según corresponda. Por otra parte en la Fig 4.5

[1] https://github.com/MashiPe/TesisMLOps.git

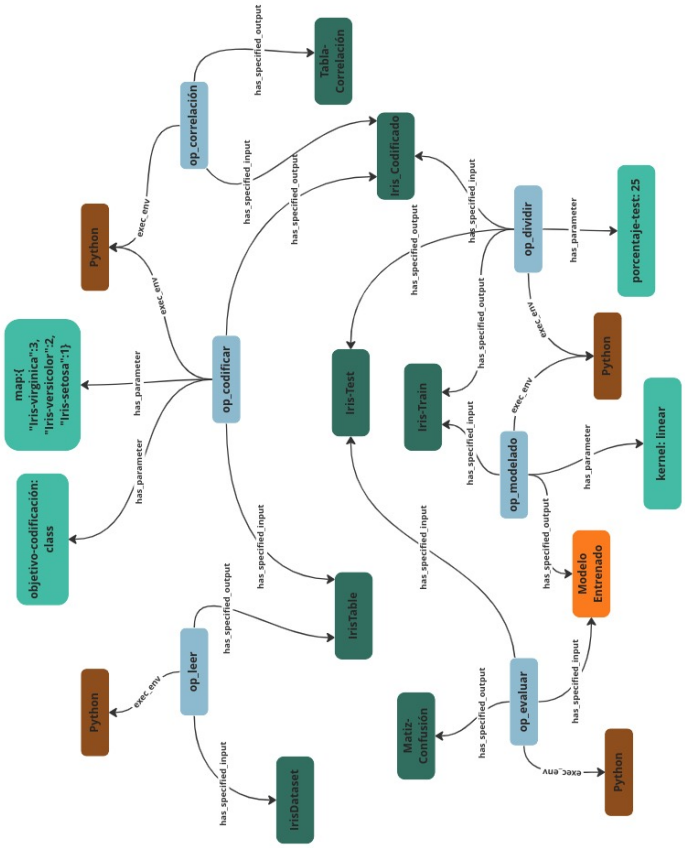


Fig 4.5: Representación detallada de experimento. Aquí se puede observar cada uno de los operadores junto a sus correspondientes dependencias, parámetros y ambientes de ejecución.

se muestra una representación de los elementos de entrada, salida y diferentes parámetros empleados por cada uno de los operadores. En el caso del operador op_leer, se tiene una única entrada llamada IrisDataset y una única salida llamada IrisTable, este operador corresponde a la lectura de los datos y como se ilustra en el gráfico es ejecutado en un ambiente *Python*; seguido se tiene al operador op_codificar el cual toma el resultado del operador op_leer y codifica la columna *Class* del dataset numerando cada miembro con un número del uno al tres tal y como se expresa mediante los parámetros del operador y se pretende ejecutar dentro de un ambiente *Python*. Esta lógica es similar para el resto de operadores dando como resultado una representación en forma de grafo del experimento. Cabe recalcar que el orden en el que se ejecuta cada uno de los operadores es definido de manera implícita mediante las dependencias de entrada y salida de cada uno de los operadores, debido a la forma de grafo acíclico.

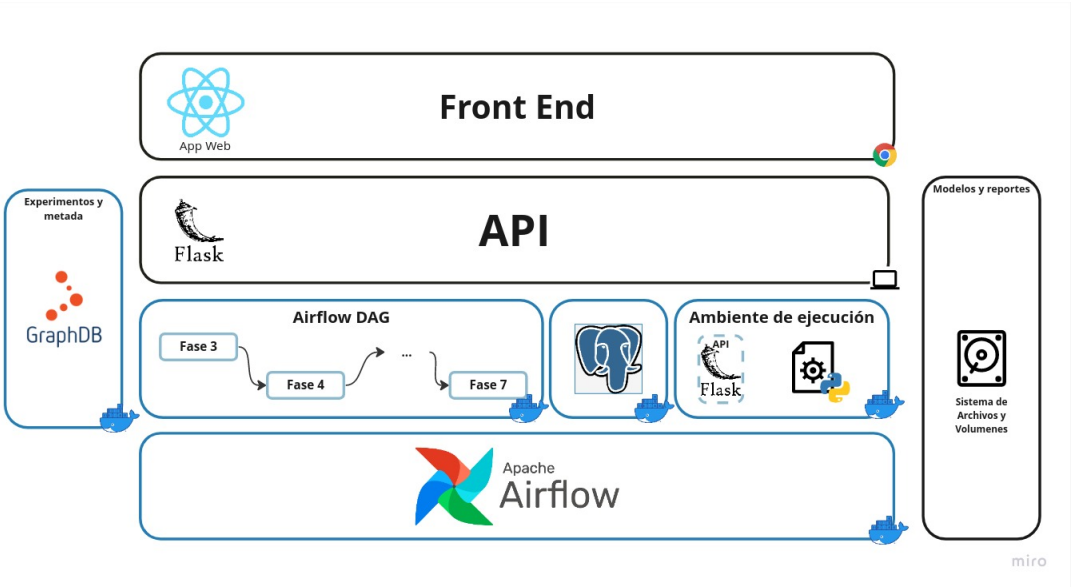


Fig 4.6: Arquitectura del sistema

4.2. Arquitectura del sistema

Al momento de diseñar el sistema, este se considerada como varios componentes o módulos que interactúan entre sí, como se ilustra en la Fig 4.6, en donde se puede observar los diferentes módulos que componen al sistema.

El sistema esta formado por un módulo de Front End, desde el cual el usuario interactúa directamente con el sistema. Seguido se tiene un módulo de API REST, el cual sirve como intermediario entre el módulo de Front End y el resto de módulos del sistema. Continuando, se tienen módulos como una base de datos PosgreSQL en la cual se almacenan conjuntos de datos a modo de tablas. También se tiene a Airflow junto a sus respectivos DAGs, que se utilizan en conjunto para orquestar la ejecución del experimento. Por otra parte se tienen distintos ambientes de ejecución que se encargan de llevar a cabo las tareas específicas del experimento. Finalmente tiene a GraphDB utilizado como triple-store para almacenar las instancias de la ontología previamente descrita y que se utiliza como modelo de datos, el sistema de archivos y volúmenes de Docker en donde se almacenan resultados como gráficos o reportes, estos dos últimos componentes funcionan de forma transversal en el sistema puesto que son utilizados por el resto de componentes. Para concluir con la sección se da una explicación condensada del como cada componente interactúa automatizando las fases tomadas en cuenta durante este proyecto.

4.2.1. Front End

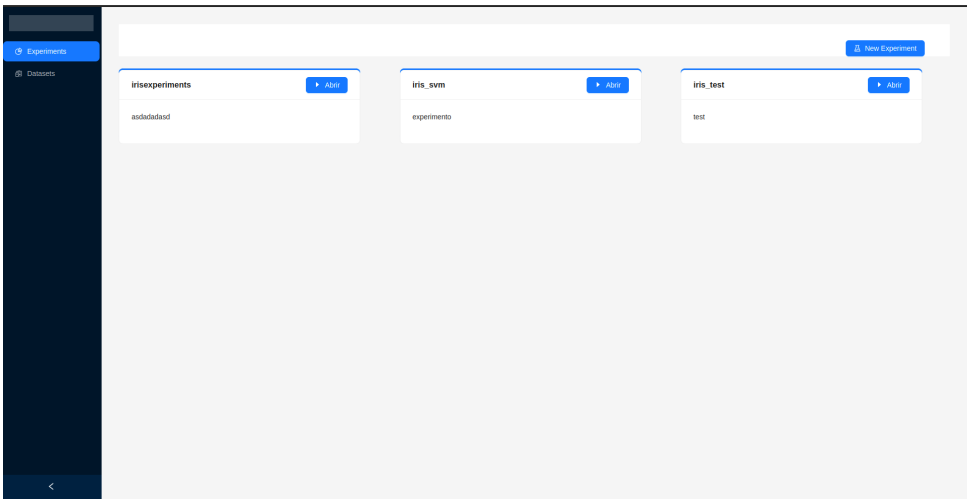
Este componente se encarga de facilitar la interacción del usuario presentando una interfaz gráfica, fue desarrollado como una interfaz web utilizando la librería React (React, 2023) con ayuda de AntD (Design, 2023) que facilita el desarrollo al proveer componentes de React pre-armados. Puesto que dentro de la interfaz se manipulan datos complejos y variados, además de necesitar comunicación constante con la *API (4.2.2)*, resulta necesario alguna solución robusta que ayude con estas tareas, por lo tanto se hizo uso de Redux (Redux, 2023) tanto para *state management* como para facilitar la comunicación entre la interfaz y la API utilizando un sistema de almacenamiento centralizado y automatizando tareas repetitivas.

La interfaz gráfica se la dividide en 2 partes; la primera, visualizada en la Fig 4.7, se centra en la administración tanto de experimentos y los datasets que se requieran usar en cada uno de estos. En esta interfaz, por una parte, se puede acceder a todos los experimentos agregados al sistema como se visualiza en la Fig 4.7a, y por otra se pueden administrar los conjuntos de datos y organizarlos en distintas versiones que mas tarde pueden ser utilizados en diferentes flujos MLOps como se ilustra en la Fig 4.7b.

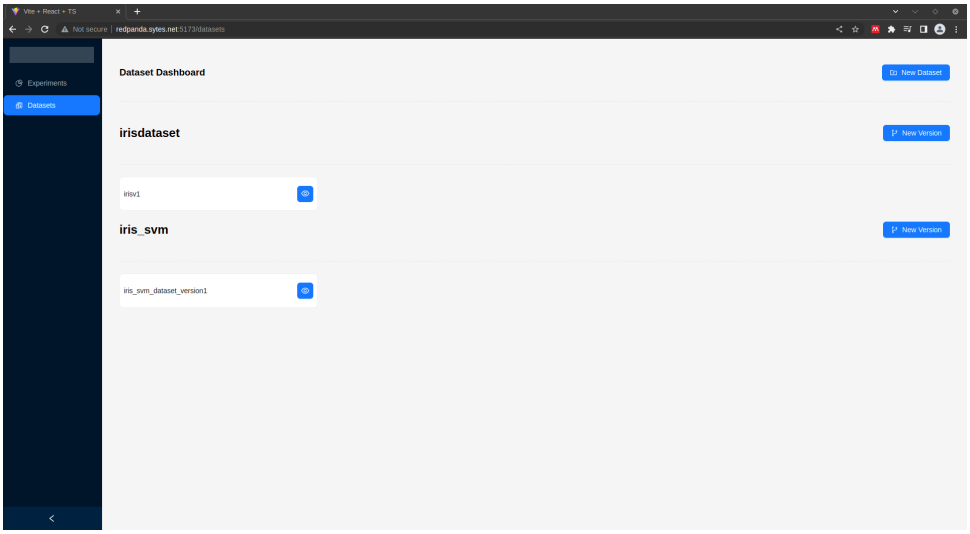
Este componente sería utilizado durante la primera fase considerara dentro del proyecto correspondiente a la fase de *Definición de experimento*. Concretamente se encarga de enviar información correspondiente a la definición del experimento y conjuntos de datos en archivos en formato CSV a la API, para mayores detalles de la API referirse a la sección 4.2.2.

La segunda parte de la interfaz corresponde a un editor en el cual se describen los experimentos, esta interfaz se muestra en la Fig 4.8 y se la puede dividir en tres componentes. El primer componente, etiquetado como *Barra Lateral*, permite crear nuevas versiones del experimento, seleccionar la versión en la que se quiera trabajar y tomar una pequeña previsualización de los conjuntos de datos que se tienen disponibles de forma global en la plataforma. El segundo componente es la *Barra de Operadores* desde la que se pueden agregar los diferentes operadores al experimento que se encuentran agrupados según las fases utilizadas de CrisPDM y el tipo de función que ejercen dentro de un proceso MLOps. El tercer y último componente es el *Canvas* y comprende dos vistas; por una parte se tiene la lista de los operadores agregados al experimento en la que se despliegan los parámetros o se pueden editar los mismos, mientras que la vista de grafo permite visualizar de mejor manera las dependencias y el orden de ejecución de cada uno de los operadores involucrados dentro del experimento.

Adicionalmente, se tienen distintos botones ubicados en la parte superior del editor los cuales cumplen la función de navegar a la página de administración, desplegar los recursos necesar-



(a) Administración de Experimentos



(b) Administración de Datasets

Fig 4.7: Interfaz gráfica de administración

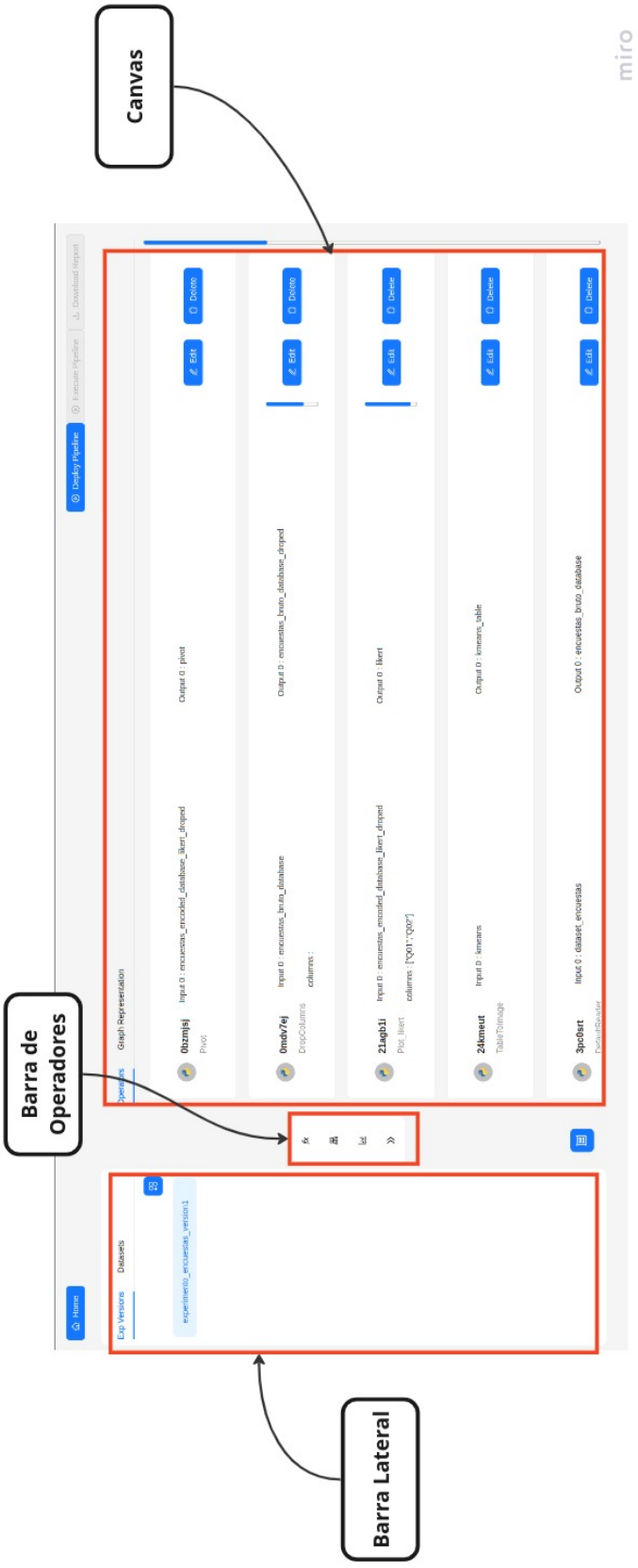


Fig 4.8: Editor de Experimentos

ios según se descripción en el experimento, iniciar la ejecución de la versión del experimento seleccionado en ese momento y descargar un reporte con los resultados obtenidos en la ejecución del experimento.
Para mayor detalle de la utilización de esta interfaz referirse al Manual de Usuario en la sección de anexos.

4.2.2. API

Este componente fue desarrollado en Python con el framework Flask (Flask, 2023), y se encarga de comunicar la interfaz gráfica con los demás componentes: la Orquestación en Airflow, la ontología en GraphDB, el almacenamiento en la base de datos y desplegar los ambientes de ejecución. Este componente, junto con la interfaz gráfica, son los únicos componentes que se encuentran fuera de un contenedor de Docker, la razón es tener mayor facilidad de crear y levantar los contenedores necesarios; caso contrario, si la API estuviera en un contenedor, levantar los demás servicios representaría una mayor dificultad, puesto que la utilización de Docker dentro de un contenedor requiere montar el socket de Docker en dicho contenedor, aumentando la complejidad la comunicación con el servicio.
La API se encuentra conformada por varios endpoints cuyas funciones son las siguientes:

- Operaciones CRUD dentro de la ontología.
- Operaciones de consulta de tablas en las diferentes bases de datos.
- Generar el pipeline de ejecución del experimento en Airflow.
- Generar e iniciar los ambientes de ejecución necesarios para el experimento mediante Docker Compose.
- Generar un informe con los resultados de la ejecución de un experimento.
- Realizar las predicciones como parte de los despliegues de los diferentes modelos.

La funcionalidad específica de cada uno de los endpoints disponibles se listan en la tabla B.1, que se encuentra en el Anexo B.5.5. Este componente es utilizado en las fases de; *Definición de experimento*, recibiendo y modificando las instancias de la ontología según la información recibida del Front End, también emplea este componente en la fase de *Aprovisionamiento de recursos* al generar el pipeline MLOps en forma de DAG de Airflow y desplegar los recursos necesarios para el experimento, finalmente se aprovecha esta API para poner a disposición los modelos generados en los experimentos en la fase de *Despliegue*.

Generación de pipeline

El sistema de generación de pipelines se realiza mediante *plantillas o templates* y el motor de plantillas Jinja (Jinja, 2023). En cada plantilla se definen las entradas, salidas y parámetros que va a tener un operador en específico. Posteriormente, en cada plantilla se especifica el ambiente de ejecución y proceso correspondiente, para finalmente ejecutar la tarea en cuestión mediante una petición REST al ambiente de ejecución. Una vez se tienen las plantillas, estas son ingresadas dentro de Jinja con los parámetros respectivos y generar la definición de un *task* dentro de un DAG de Airflow, este procedimiento se lo realiza para cada uno de los operadores que se utilizan en un experimento. Como paso final para la generación de un DAG, se agregan cada una de las tareas del experimento en un único script de Python que representa un DAG de Airflow. Un ejemplo de esto se observa en la Fig 4.9, aquí se toma como base al operador *Split*, el cual, toma un conjunto de datos como entrada y produce dos conjuntos de datos como salida según la proporción que se indique en el parámetro. Para mayores detalles referirse al manual de desarrollo en el Anexo B.

Fig 4.9: Ejemplo de procesamiento de plantilla.

4.2.3. Ambientes de ejecución

Para la ejecución de cada uno de los algoritmos, la herramienta se encarga de proveer un ambiente de ejecución listo para usar, este ambiente consiste en un contenedor de Docker

llamado “ejecutor de scripts” con base en Ubuntu 20.04, que incluye el lenguaje Python 3.8 y R. Como se puede ver en la Fig 4.6 de la arquitectura del sistema, este contenedor contiene una API interna, a la que llegan las peticiones de cual es el script o algoritmo que debe ejecutarse en un momento dado. Esta API está compuesta únicamente por dos endpoints; el primero: “/ejecutarpython/<script>” que ejecuta los scripts escritos en Python; y el segundo: “/ejecutarR/<script>” que ejecuta los scripts escritos en R.

A cada uno de estos endpoints se le envía el nombre del script a ejecutar como parte de la URL, por ejemplo */ejecutarpython/kmeans* representaría la ejecución del algoritmo K-Means. Por otra parte, en el *body* de la petición REST, se envía información en formato JSON con una única clave “parametros” que contiene cada uno de los parámetros requeridos por el script, los cuales son:

- Tablas de entrada
- Tablas y/o figuras de salida
- Parámetros necesarios para la tarea a realizar por el script
- Archivo *inifile* correspondiente a la versión del experimento, con el objetivo de identificar la base de datos temporal para almacenar los resultados.

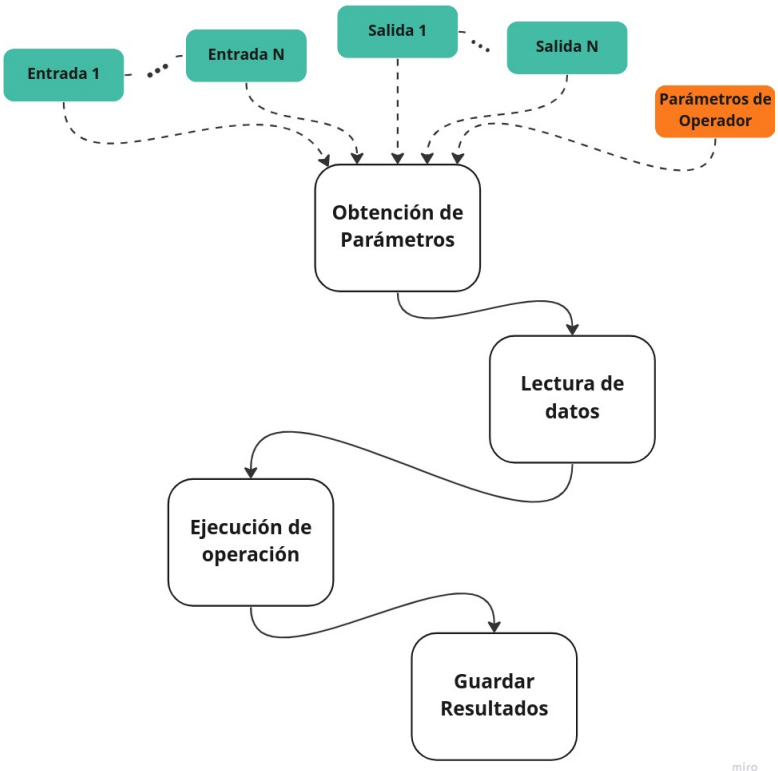


Fig 4.10: Estructura general de un script

Scripts

La parte central de este contenedor son los scripts. La estructura general de estos se muestra en la Fig 4.10, en donde se ilustra como cada script realiza 4 tareas específicas: Obtención de los Parámetros, Lectura de Datos, Ejecución de la Operación, Guardado de Resultados. Cada una de estas funciones es explicada a continuación:

- **Obtención de los parámetros:** Cada script es ejecutado desde la API de este contenedor mediante el comando: *python3 script.py parámetros*. El argumento del comando de ejecución en todos los casos es una cadena de texto, la cual internamente se convierte a un diccionario. Se han definido 3 tipos de parámetros los cuales son:
  - Las entradas del algoritmo representadas a modo de tablas en la base de datos.
  - Las salidas del algoritmo representadas a modo tabla y/o archivo en donde se almacenan los resultados del proceso.
  - Todos los parámetros que el algoritmo a ejecutar requiera.
- **Lectura de datos:** En esta tarea se realiza la conexión a la base de datos para posteriormente hacer la lectura de los datos que serán transformados en un dataframe para procesos consecuentes.
- **Ejecución de la operación:** En este paso, se realiza la tarea o algoritmo deseado con los datos obtenidos en el paso anterior. Ejemplo: En el script ”split_dataset.py”, se separan los conjuntos de datos en dos grupos, uno para entrenamiento de un modelo y otro para pruebas del modelo.
- **Guardar Resultados:** Finalmente, los resultados se guardan en una nueva tabla en la base de datos para su uso posterior. Ejemplo: En el script ”split_dataset.py” se guardan tanto el dataset de entrenamiento y el dataset de prueba, cada uno en una tabla en la base de datos, o un archivo que representa ya sea un gráfico o un modelo entrenado.

Los ambientes de ejecución entran en juego durante la ejecución de las fases *Extracción de datos, Exploración de datos, Procesamiento de datos, Construcción de modelos y Validación de modelos* al ejecutar cada uno de los scripts utilizados durante un experimento. Los scripts realizados en este proyecto satisfacen la demostración de SVM con el dataset “Iris” y al caso de uso desarrollado en la sección 5, la lista completa se la puede ver en la tabla B.2 en el Anexo B.5.6 Información de cómo incluir más scripts se encuentra en el Anexo B.7.

4.2.4. Orquestación

Como se expuso en la sección 4.2.3, estos ambientes son independientes el uno del otro por lo que la orquestación resulta parecido a un problema en un sistema distribuido. Por otra parte, la naturaleza de grafo en la que se representan los experimentos implica que pueden existir varios componentes del pipeline que pueden ser ejecutados de forma paralela. Además, puesto que los operadores que entran en juego en el experimento pueden o no tener dependencias con uno o varios operadores, dentro del proceso es necesario que un agente sea capaz de orquestar cada ambiente de forma individual, para esto se utilizó Apache Airflow, el cual, como se especificó anteriormente en la sección 2.5 de *Antecedentes y Trabajos Relacionados*, permite definir flujos de trabajo mediante grafos acíclicos. Además, puesto que este software es agnóstico del tipo de aplicación que se quiera implementar, junto a su ya comprobada efectividad en su aplicación en proyectos de MLOps como se discutió en el estado del arte, resulta un candidato perfecto para utilizarlo a modo de orquestador dentro del proyecto. El despliegue de Airflow se consigue mediante la utilización de Docker similar a los ambientes de ejecución. La utilización de Airflow elimina el problema de control de flujo en un proceso de forma eficaz asegurando que cada una de las partes involucradas en el mismo se ejecuten únicamente según sus dependencias. No obstante, se deben tomar dos consideraciones importantes al utilizar esta herramienta y son discutidas a continuación.

Las tareas definidas en un pipeline de Airflow son ejecutadas dentro de los *workers*, los cuales no necesariamente cuentan con todos las dependencias necesarias de un ambiente de ejecución capaz de efectuar todos los procesos requeridos. Por otra parte, si se buscara convertir a un ambiente de ejecución en un *worker* de Airflow se perdería la noción de cada ambiente puesto que la herramienta no hace ninguna diferenciación entre los *workers* disponibles, es decir que, en el caso de tener ambientes diferentes, por ejemplo, un ambiente de Python y uno de R, el instrumento no tendría forma de saber a que ambiente le corresponde que tarea. Es por esto que se decidió separar los *workers* de Airflow de los ambientes de ejecución y comunicarlos mediante el llamado *ejecutor de scripts* revisado anteriormente, de forma que cada tarea definida en el proceso se encarga de llamar al ambiente correspondiente junto al proceso adecuado para cumplir la tarea definida en el pipeline MLOps.

Otro problema a considerar es el intercambio de información entre cada tarea. Airflow como tal no se encarga de pasar información entre cada componente del grafo, de forma que en cada paso del flujo es necesario recuperar la información según se requiera. Para solventar esto, cada uno de los scripts se encarga de recuperar los datos pertinentes para su ejecución por lo

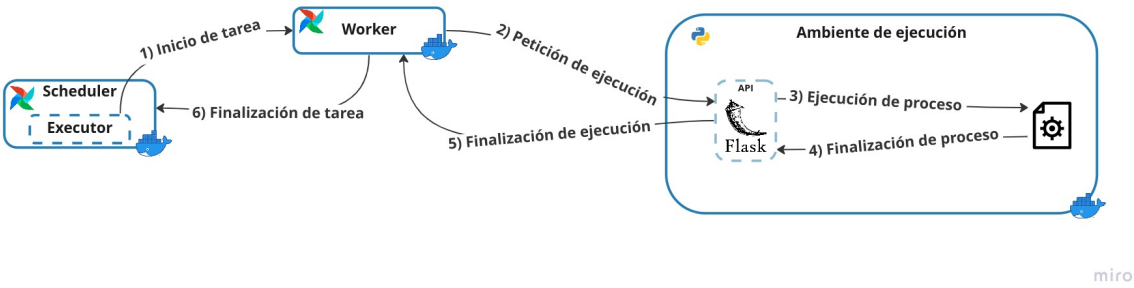


Fig 4.11: Ejemplo simple de orquestación

que el intercambio de datos entre tarea y tarea queda completamente aislado del orquestador como tal.

Finalmente, en la Fig 4.11 se muestra a detalle como se realiza la orquestación de un pipeline MLOps. En dicha figura se puede observar como Airflow únicamente se comunica mediante peticiones REST, de forma que la función de Airflow se reduce solamente a la orquestación accionando las tareas necesarias asegurando sus dependencias, mientras que los scripts y los ambientes de ejecución se encargan de los detalles específicos de cada proceso, como por ejemplo, la recuperación de los datos necesarios para cada tarea en específico. De esta forma el módulo de orquestación se ve involucrado en las fases de *Extracción de datos, Exploración de datos, Procesamiento de datos, Construcción de modelos y Validación de modelos*.

4.2.5. Almacenamiento

El almacenamiento es utilizado durante todas las fases del proyecto puesto que en este componente se manejan todos los datos de la aplicación como se describe a continuación.

El almacenamiento dentro del sistema cumple dos funciones principales. Por un lado se encarga de guardar la información que maneja el sistema, dicha información incluye a las descripciones de los experimentos expresados mediante la ontología, conjuntos de datos y sus respectivas versiones, gráficos generados durante la ejecución de un experimento y modelos entrenados. Por otra parte, como se discutió en 4.2.4, Airflow como tal no se encarga de transferir datos entre actividades por lo que es necesario idear una forma de trasladar los datos entre operadores.

Debido a esto se decidió utilizar una base de datos como una forma de transmitir los resultados entre operadores. Considerando estas características, el sistema contempla dos tipos de almacenamiento que interactúan durante la ejecución y edición del pipeline:

- Almacenamiento persistente
- Almacenamiento temporal

El almacenamiento persistente se conforma por tres partes; por un lado se hace uso de una base de datos de grafo, concretamente GraphDB (Ontotext, 2023), en donde se almacenan las descripciones de los experimentos que cumplen con la ontología tomada a modo de representación de datos de los distintos experimentos, por otra parte se hace uso de PostgreSQL (Postgresql, 2023), en donde se almacenan los conjuntos de datos que posteriormente son utilizados en un pipeline y finalmente se aprovecha el sistema de archivos en donde se guardan tanto gráficos como modelos entrenados.

El almacenamiento temporal está compuesto únicamente por una base de datos en PostgreSQL, dicha base de datos es generada por el experimento, por lo que se elimina la posibilidad de que diferentes experimentos interfieran en el procesamiento de otros. Una vez creada la base de datos para el experimento, se almacenan tanto los datos iniciales como los resultados intermedios que se generen durante el procesamiento. Se hace hincapié que este almacenamiento es temporal por lo que toda información que sea alojada en las bases de datos del experimento será eliminada entre cada ejecución del pipeline.

En la Fig. 4.12 se puede ver como el sistema interactúa con cada tipo de almacenamiento, por un lado se visualiza como cada resultado intermedio del flujo del pipeline se almacena y puede ser consumido por tareas y operadores posteriores. Por otra parte los modelos y gráficos generados durante la experimentación son enviados directamente al almacenamiento persistente y aquellos resultados intermedios o datasets producidos durante la ejecución de un experimento pueden ser enviados al almacenamiento persistente para ser utilizados en experimentos posteriores.

4.3. Proceso completo

Con todos los componentes del sistema definidos, en la Fig 4.13 se observa como cada componente entra en juego para la ejecución de las fases definidas para este proyecto.

La primera fase consiste en la descripción del experimento, aquí entran en juego el Front End, la API y se hace uso de GraphDB y PostgreSQL. La fase consiste en que el usuario ingresa tanto la definición del experimento como de los datos a utilizar mediante un archivo CSV, esta información es enviada a la API la cual se encarga, por una parte persistir los datos en PosgreSQL y de producir la descripción ontológica tanto del experimento como de los datos y almacenar ambas descripciones en GraphDB.

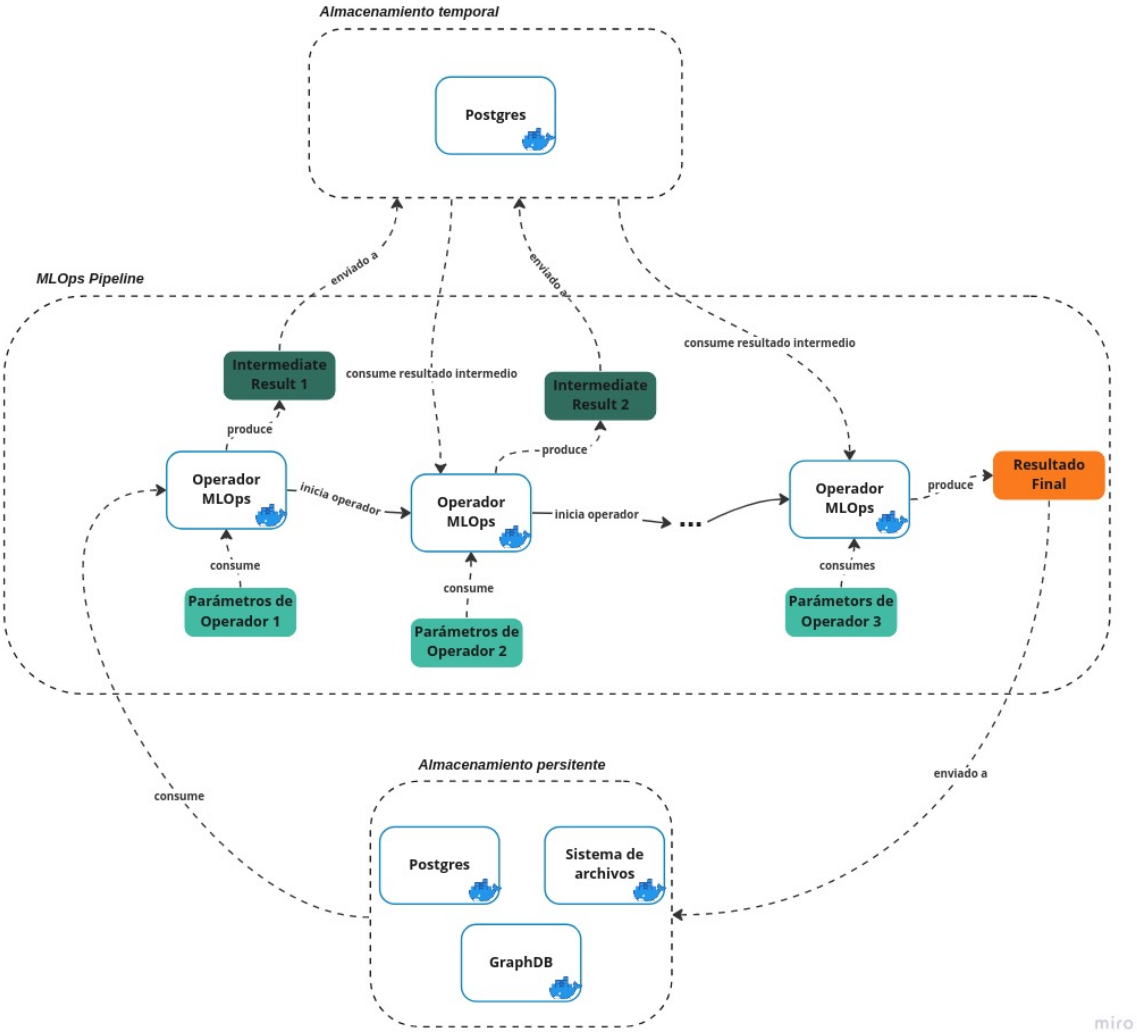


Fig 4.12: Flujo de información y almacenamiento

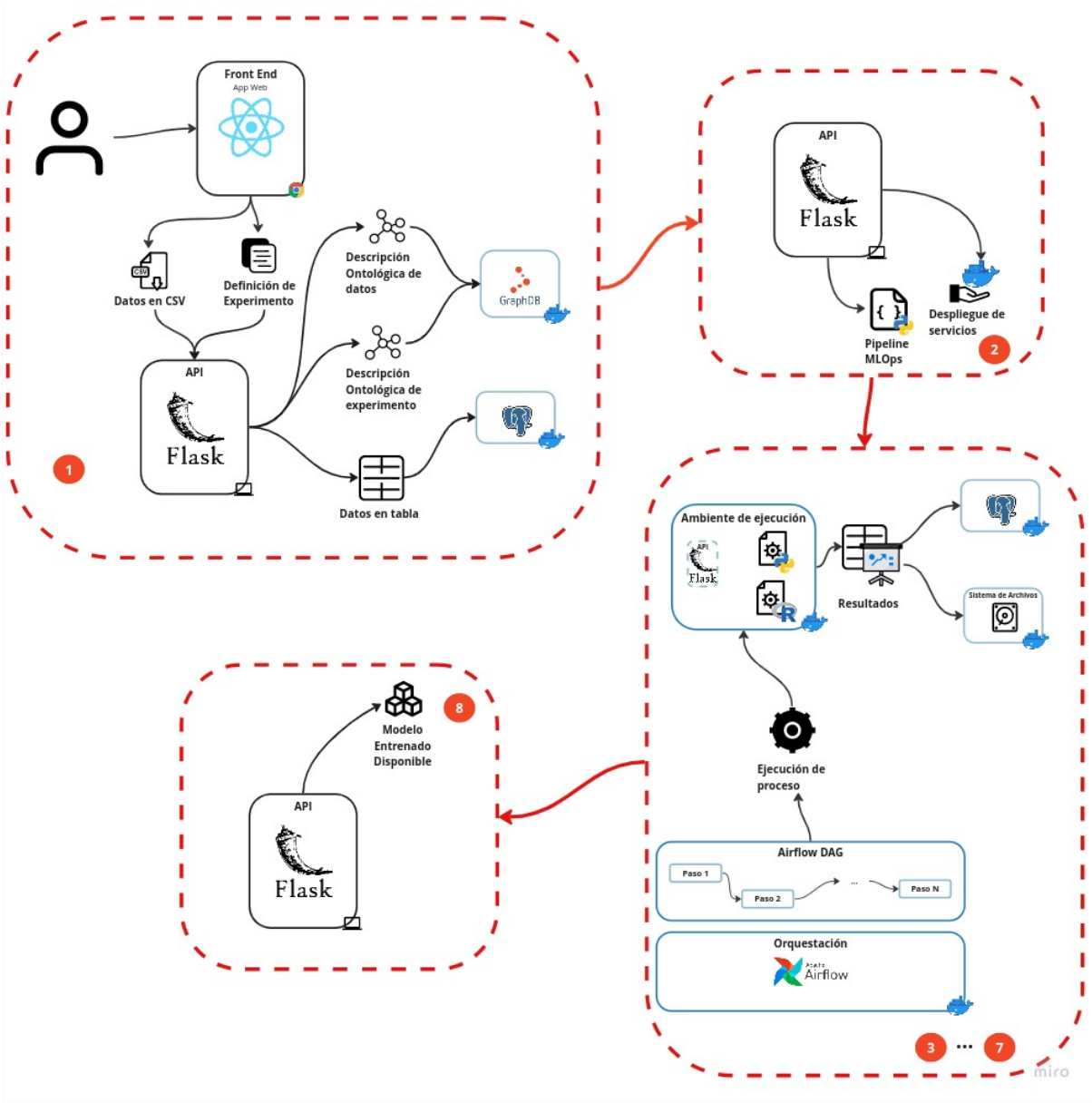


Fig 4.13: Flujo de ejecución de fases

Una vez se tienen las descripciones ontológicas necesarias, la API despliega los servicios necesarios que se hayan definido en la ontología, y genera un pipeline MLOps definido mediante un script de Python que describe un DAG de Airflow. Este proceso cumple con la segunda fase correspondiente al aprovisionamiento de recursos.

Con todos los servicios necesarios desplegados la ejecución de cada uno de los pasos del experimento quedan a cargo de Airflow y el DAG generado en la fase anterior. Airflow se encarga de orquestar a cada ambiente de ejecución según se haya definido en el DAG, automatizando de forma eficaz fases como la extracción de datos hasta la validación de modelos según sea el caso, y almacenando los resultados según corresponda. Cabe recalcar que un experimento puede o no hacer uso de todas las fases.

Finalmente la fase de despliegue queda a cargo de la API, la cual una vez finaliza la ejecución de los procesos anteriores pone a disposición de forma automática el modelo mediante peticiones REST concluyendo con todo el flujo del experimento.

5. Caso de uso: Encuestas

Durante una investigación de la Facultad de Hospitalidad se realizaron encuestas que luego fueron utilizadas en el marco del proyecto: Fostering a platform for research-based education to support sustainable development through Tourism in the Cajas Massif Biosphere Area (CMBA), donde se realizó una encuesta a 825 participantes de 8 comunidades rurales y urbano-rurales del Área de la Biosfera del Macizo del Cajas ubicadas en Cuenca y Naranjal (Baños, Sayausí, Agroproductores del Azuay, Tejedoras de sombrero de paja toquilla, Migüir, Cluster Naranjal, 6 de Julio, Tsuer Entsa) con el objetivo de analizar cómo y por qué surgieron o evolucionaron durante la pandemia las actitudes de los residentes hacia el turismo como medio de desarrollo sostenible, así como la capacidad de recuperación de sus comunidades para hacer frente a la crisis. Al momento del análisis, los investigadores realizaron varias operaciones para la transformación de datos, para finalmente realizar PCA y K-means al dataset. En su versión original el proceso se lo realizó de forma manual usando R Markdown, por lo tanto a continuación se recreará todo el proceso en la herramienta propuesta, en un ambiente de Python, y se compararán los resultados.

5.1. Aprovisionamiento de Recursos

La herramienta propuesta ofrece por defecto los servicios de Airflow y PostgreSQL los cuales, son esenciales para la automatización de la experimentación. Adicionalmente, para este caso de uso, se proporciona un ambiente de ejecución de Python que incluye las dependencias necesarias para llevar a cabo el experimento. El aprovisionamiento de estos servicios son el resultado de describir el experimento a través de la interfaz web. Este proceso se lo describe a lo largo del resto del capítulo.

5.2. Ingesta de datos

El primer paso que realizó el investigador con los datos originales, fue mostrar una previsualización del dataset. Para reproducir esto, en la herramienta, sección de datasets, se creó un dataset, junto con una nueva versión del mismo y es donde se permite subir el archivo csv del

experimento, la previsualización se ve al momento de la carga del archivo como se puede en la Fig 5.1.

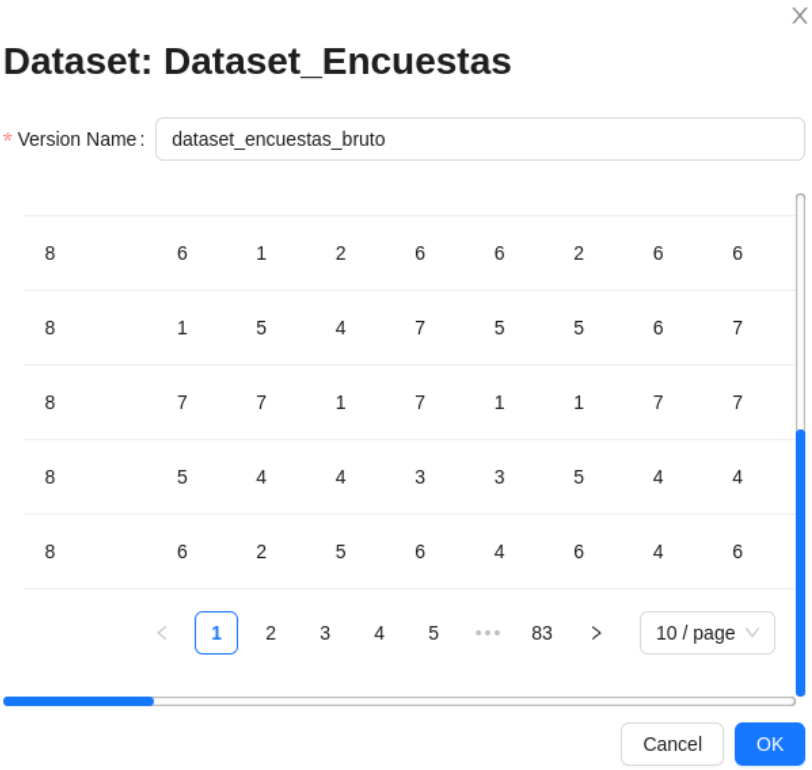


Fig 5.1: Previsualización del CSV

Antes de continuar con las siguientes fases es necesaria la creación del experimento, por ello a continuación, se creó el experimento correspondiente con el nombre ”experimento_encuestas” como se puede ver en la Fig 5.2.

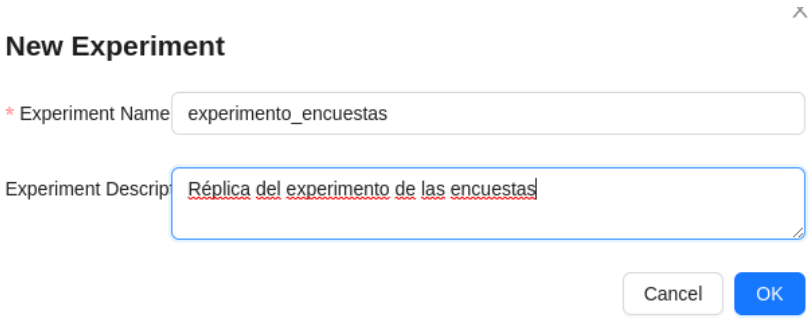


Fig 5.2: Creación del experimento

Dentro del experimento, se debe crear una nueva versión del mismo para poder continuar. Cada versión del experimento puede representar un mismo proceso pero con diferentes parámetros, o variaciones de los operadores.

Posteriormente, se agregó el operador ”default-reader” que permite enviar los datos al almacenamiento de experimento, con lo se completa la ingesta de los datos en la herramienta. La

configuración de este operador se muestra en la Fig 5.3.

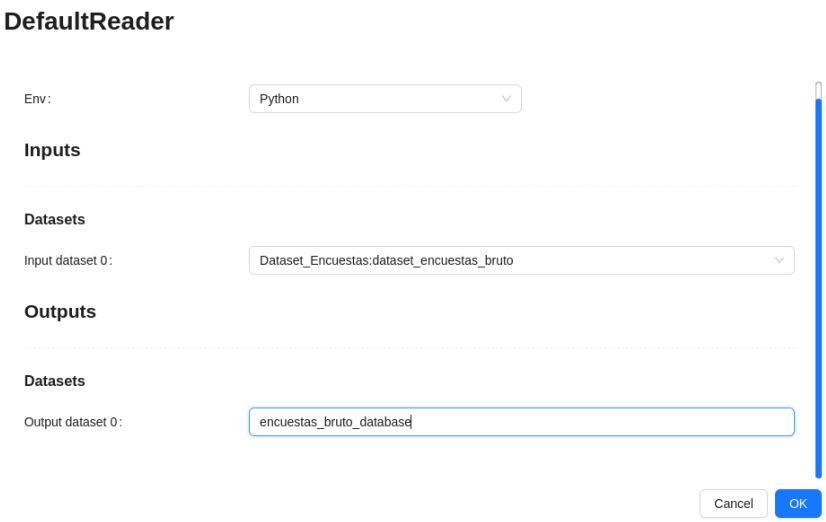


Fig 5.3: Default Reader

5.3. Exploración y Procesamiento de datos

En el proceso manual, se realizaron operaciones de preprocesamiento del dataset, con el objetivo de eliminar columnas que no se utilizan o de codificar las variables numéricas a su equivalente categórico siguiendo una escala de Likert. Como primer paso de este procesamiento se usó el operador ”Reformat Data” para codificar la columnas ”Localidad”, ”Genero”, ”Lugar-Trabajo” del dataset como se ve en la Fig5.4.

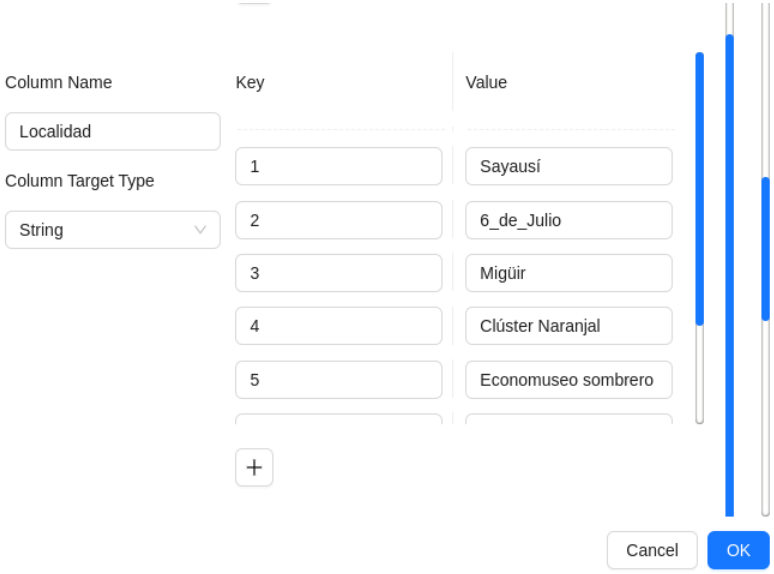


Fig 5.4: Codificación de las columnas

El siguiente paso es codificar las columnas con los valores enteros de Likert a su respectivo

”String” para ello se tiene un operador llamado ”Encode_likert”, su configuración se puede ver en la Fig 5.5.

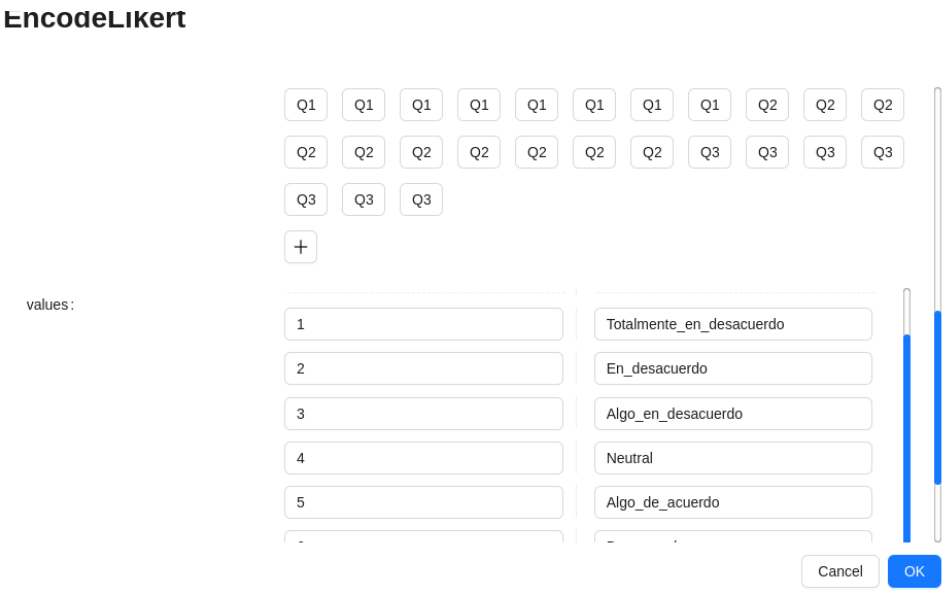


Fig 5.5: Codificación de las columnas Likert

Para finalizar con el tratamiento de los datos, se eliminan las columnas que no son de interés para este análisis. Las columnas eliminadas son: ”ActividadLaboral”, ”NivelEducativo”, ”Latitude”,”Longitude”, ”EstaCajas”, este proceso se ilustra en la Fig 5.6.

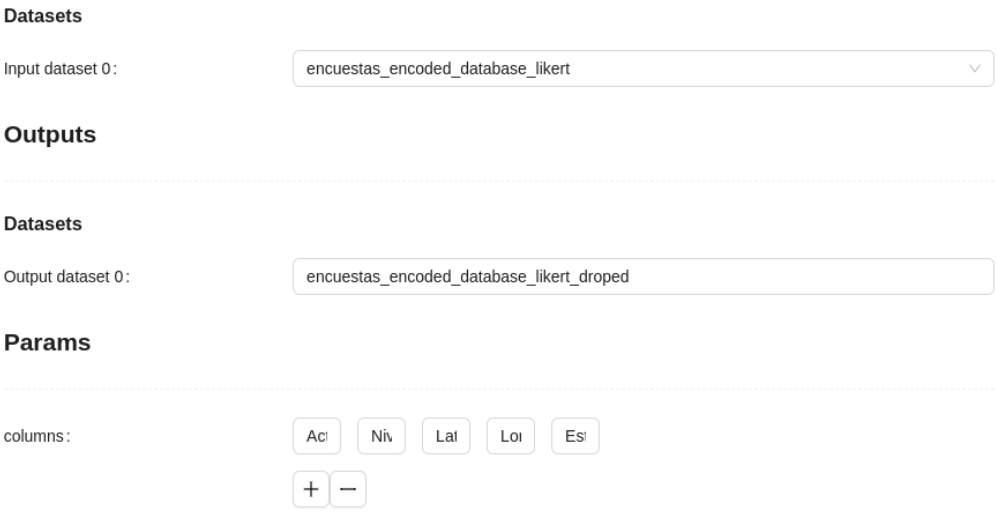


Fig 5.6: Eliminación de las columnas que no se utilizarán

Una vez finalizado la primera parte del procesamiento de los datos, se procede a aplicar los operadores ”summary” y ”pivot” con el fin de obtener un resumen de las columnas. Para realizar este análisis, la configuración de los operadores se lo puede ver en las Fig 5.7 y Fig 5.8.

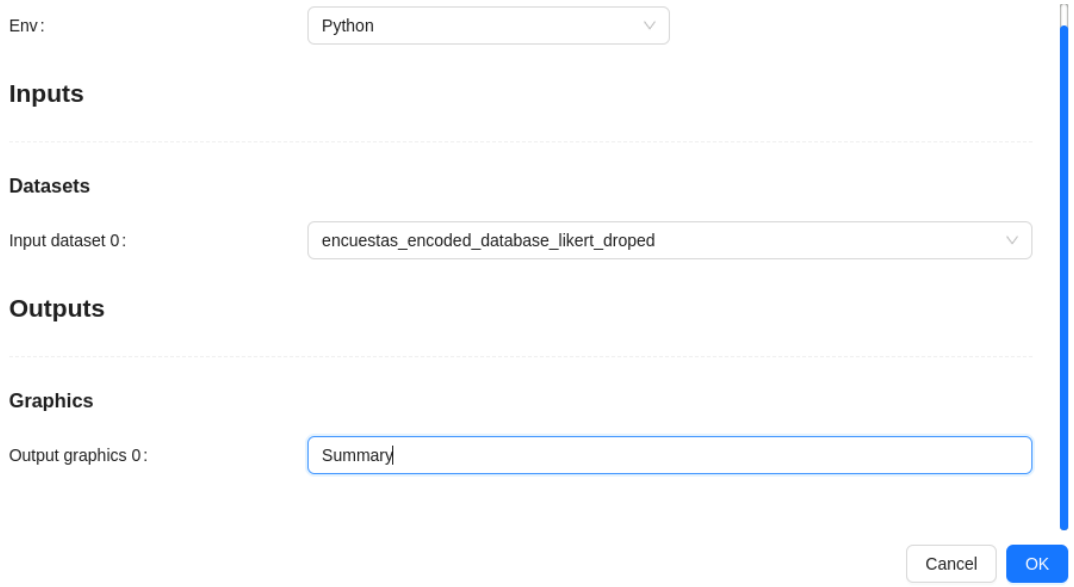


Fig 5.7: Operador Summary

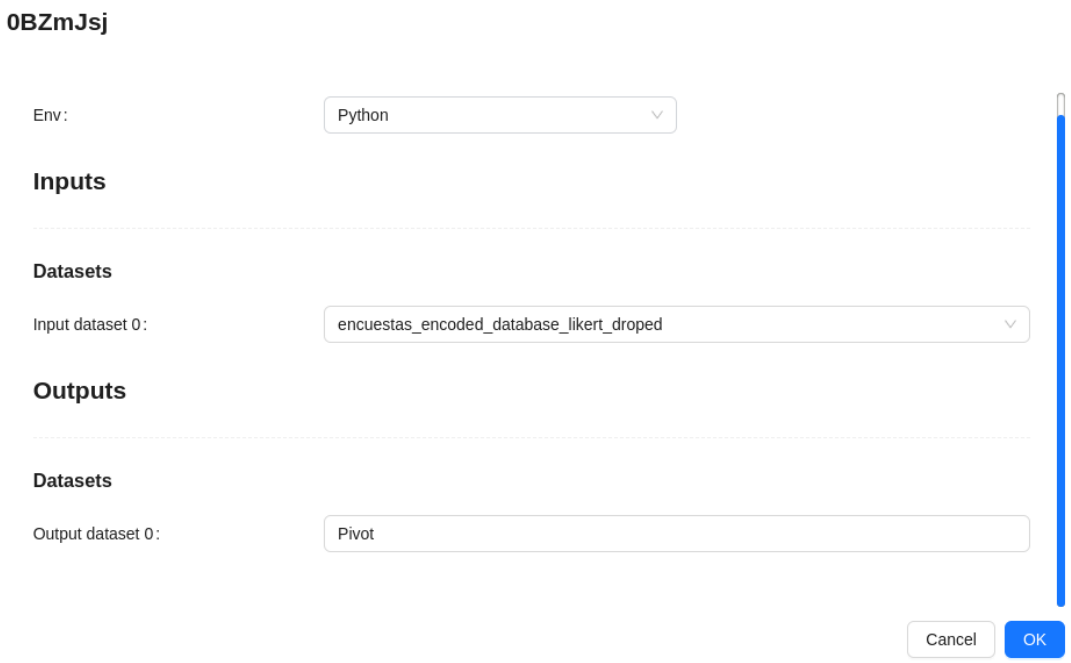


Fig 5.8: Operador Pivot

Con el fin de apreciar mejor los resultados se ofrece el operador ”TableToImage”, que muestra parte de la información almacenada en una tabla de la base de datos temporal como una imagen, su configuración se puede ver en la Fig 5.9.

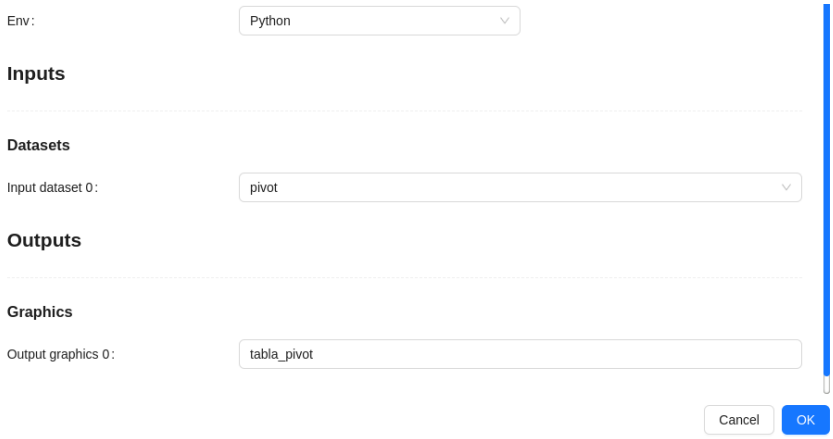


Fig 5.9: Operador ”TableToImage”

Los resultados al aplicar los operadores ”summary” y ”pivot” obtenidos en la investigación original y mediante la herramienta propuesta se los presenta a continuación en las Fig 5.10 y Fig 5.11.

```
                    Q02                          Q03
Totalmente en desacuerdo:295   Totalmente en desacuerdo: 86
En desacuerdo            :184  En desacuerdo            :106
Algo en desacuerdo       : 44  Algo en desacuerdo       : 32
Neutral                  : 67  Neutral                  :198
Algo de acuerdo          : 90  Algo de acuerdo          : 90
```

Fig 5.10: Resultados Originales Pivot

value	Q01	Q02	Q03
Algo_de_acuerdo	65	90	90
Algo_en_desacuerdo	12	44	32
De_acuerdo	105	64	144
En_desacuerdo	115	184	106
Neutral	46	67	198

Fig 5.11: Resultados Pivot de la herramienta

El siguiente paso del experimento, consiste en obtener las gráficas del histograma y de la escala de Likert, para ello se utilizan los operadores; ”density” y ”plot_likert”, cuya configuración se ilustra en las Fig 5.12 y Fig 5.13.

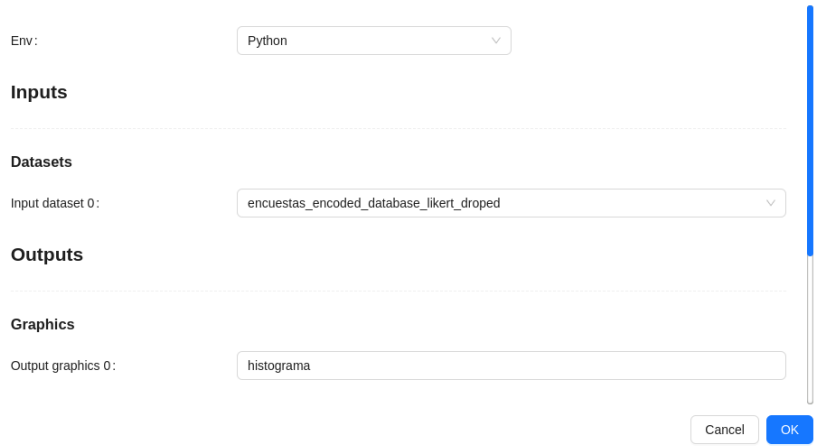


Fig 5.12: Operador Density

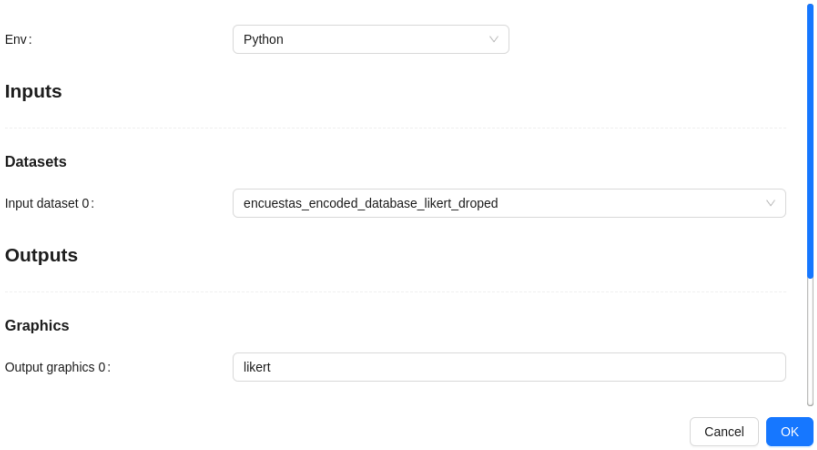


Fig 5.13: Operador plot_likert

Los resultados originales y los obtenidos en este trabajo al aplicar los operadores ”density” y ”plot_likert” se los muestra en las Fig 5.14 y Fig 5.15.

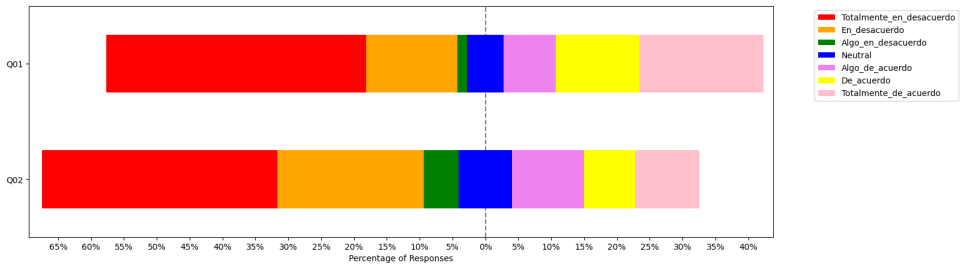


Fig 5.14: Gráfica Likert de la herramienta

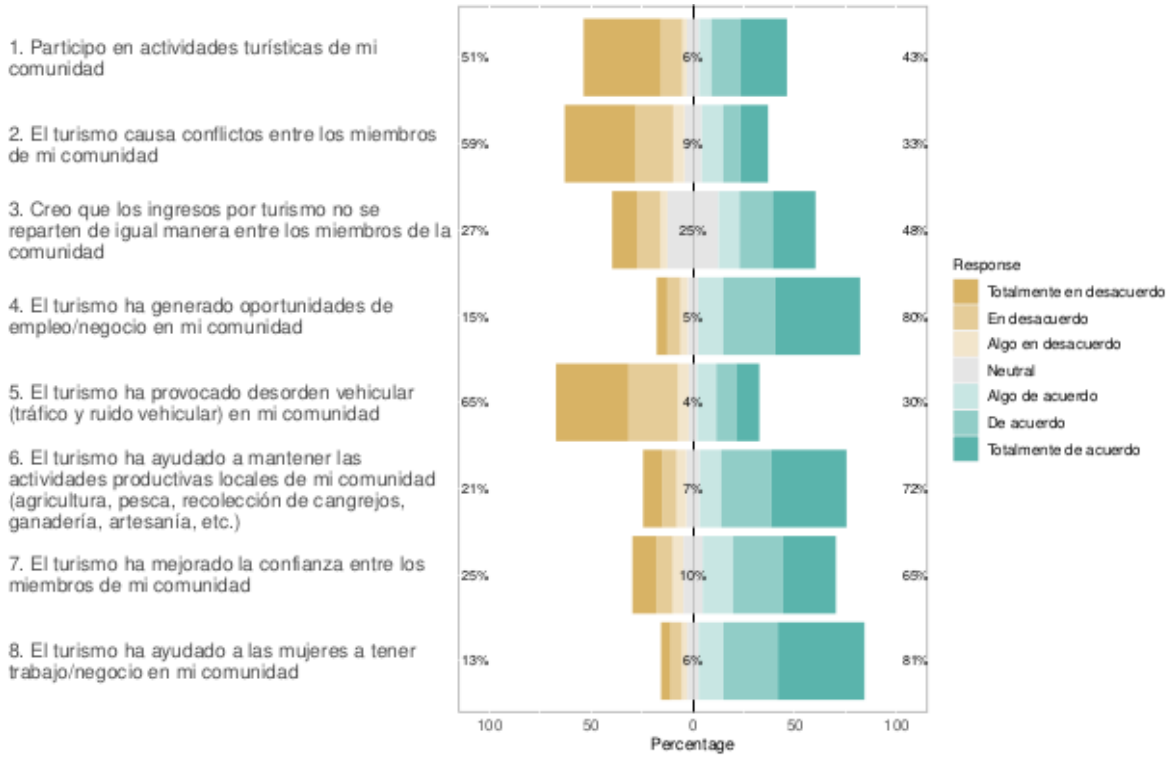


Fig 5.15: Gráfica Likert original

Continuando con el proceso, para iniciar los análisis que derivan en PCA y K-Means, se debe realizar primero un análisis de correlación, la configuración del operador que se encarga de este proceso se lo puede ver en la Fig 5.16.

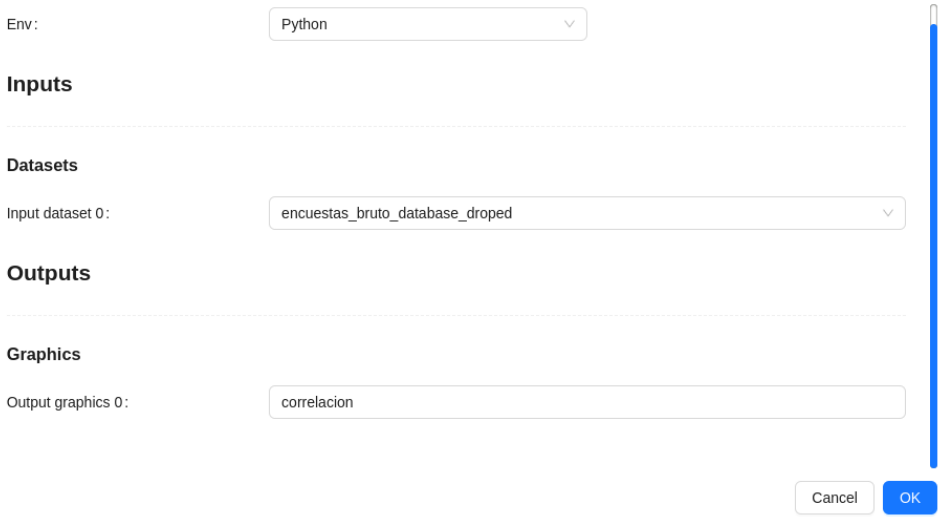


Fig 5.16: Operador de Correlación

Los resultados de la correlación obtenidos tanto de la investigación manual como de esta reproducción se muestran en las Fig 5.17 y Fig 5.18.

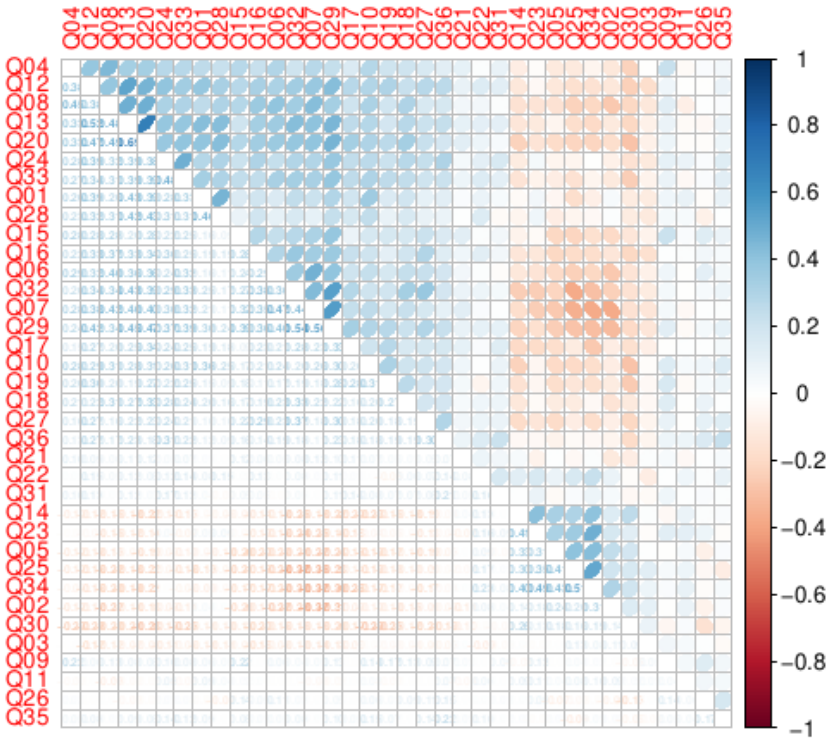


Fig 5.17: Gráfica de correlación original

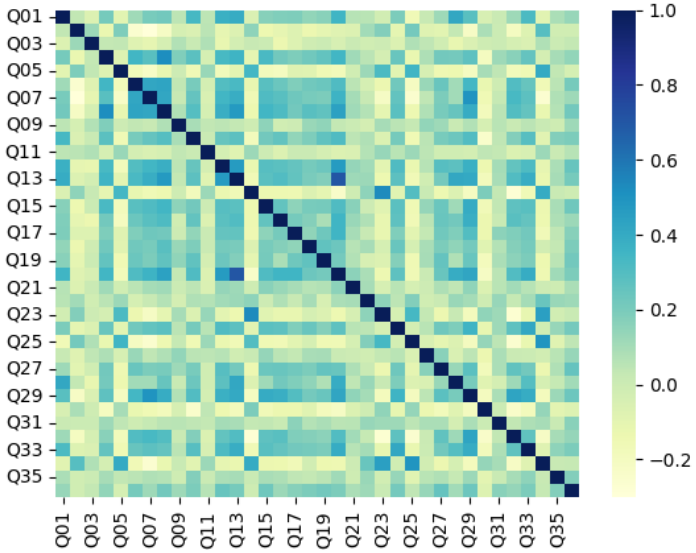


Fig 5.18: Gráfica de correlación de la herramienta

El siguiente paso es realizar un análisis de factores, para ello se aplican un Test de Barlett Y KMO, el operador que realiza estas tareas es ”TestBarlett” y su configuración se muestra en la Fig 5.20.

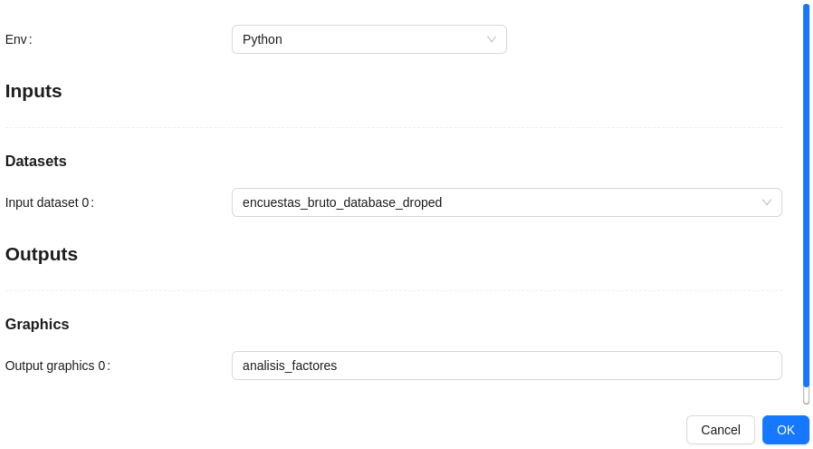


Fig 5.19: Análisis de Factores

Los resultados de estos análisis, tanto del proceso manual como al utilizar la herramienta se presentan en las Fig 5.20 y Fig 5.21 respectivamente.

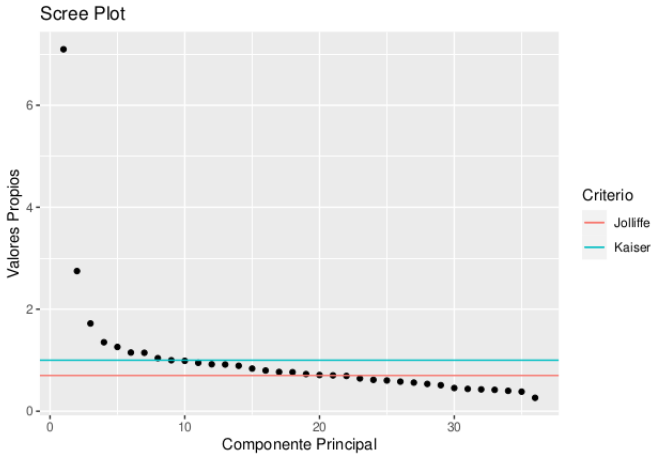


Fig 5.20: Resultado original del análisis

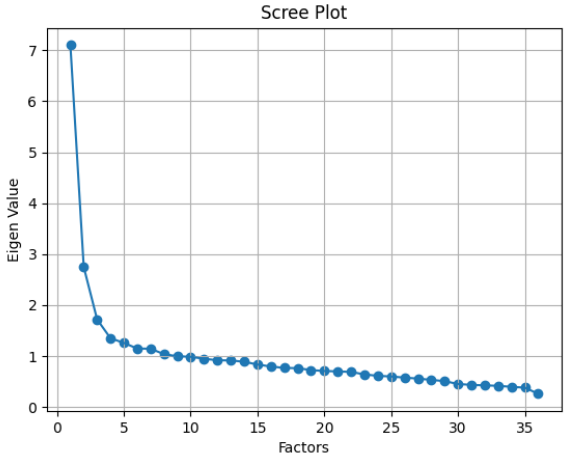


Fig 5.21: Resultado del análisis de la herramienta

Con el resultado de este análisis se procede a aplicar PCA a la matriz de correlación. La configuración del operador se puede ver en la Fig 5.22. El valor del número de componentes es tomado del paso anterior.

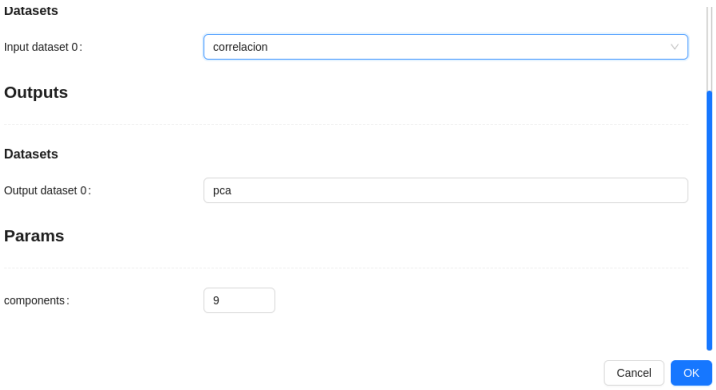


Fig 5.22: PCA

Las gráficas de la varianza aplicada para cada componente, tanto para la investigación original como para esta reproducción se presenta en las Fig 5.23 y Fig 5.25.

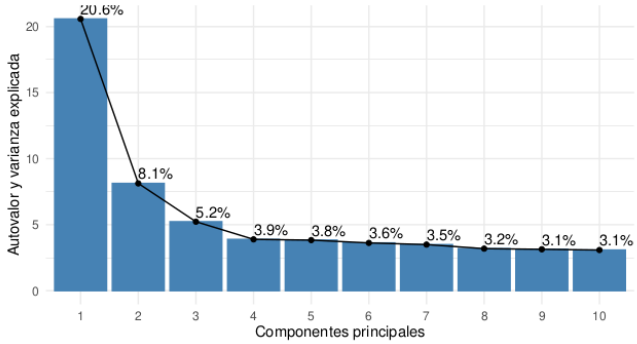


Fig 5.23: Gráfica de la varianza de la investigación original

Además se agrega el análisis de que tanto influye una variable para una componente en la Fig 5.24. Por motivos de ejemplificación únicamente se muestran tres variables

Componente	Q01	Q02	Q03
PC1	-0.17276637989844443	0.17721760331665132	0.15005130378787068
PC2	-0.13346548702382 56	-0.07249192710721866	0.19462088569438613
PC3	-0.07111950601639891	-0.029971655699096 46	-0.14210052 6306057
PC4	-0.03176474848705702	0.015078415679855255	0.018395278899596 86
PC5	-0.19882765515851925	-0.08034285645550059	0.27489646775516924
PC6	-0.05694081571457748	0.0261904184124251	0.09080877542076336
PC7	0.007150563212282654	0.0784847175132626	0.587984957324686
PC8	0.06992747887700014	0.12968301774315472	0.6008288974693611
PC9	-0.2161278144016905	-0.21644885191440014	-0.04246518934856477

Fig 5.24: Análisis de las componentes

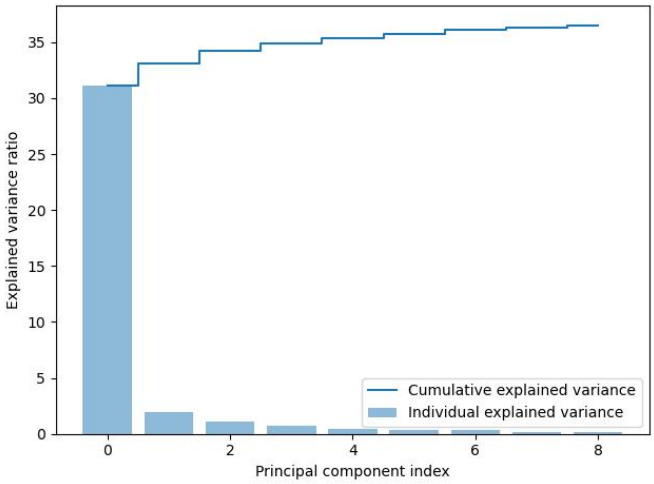


Fig 5.25: Gráfica de la varianza de la herramienta propuesta

5.4. Construcción del modelo y validación

Para finalizar con la reproducción de este experimento, se emplea el algoritmo de K-Means, para ello previamente se aplica el método del codo, La configuración del operador se la puede ver en la Fig 5.26. El resultado obtenido mediante la herramienta se ilustra en la Fig 5.27 y el resultado original en la Fig 5.28.

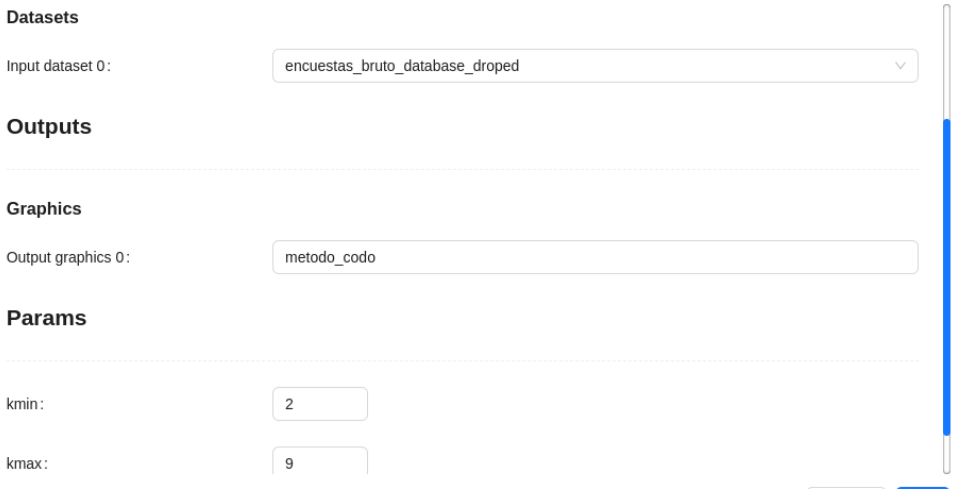


Fig 5.26: Método del codo

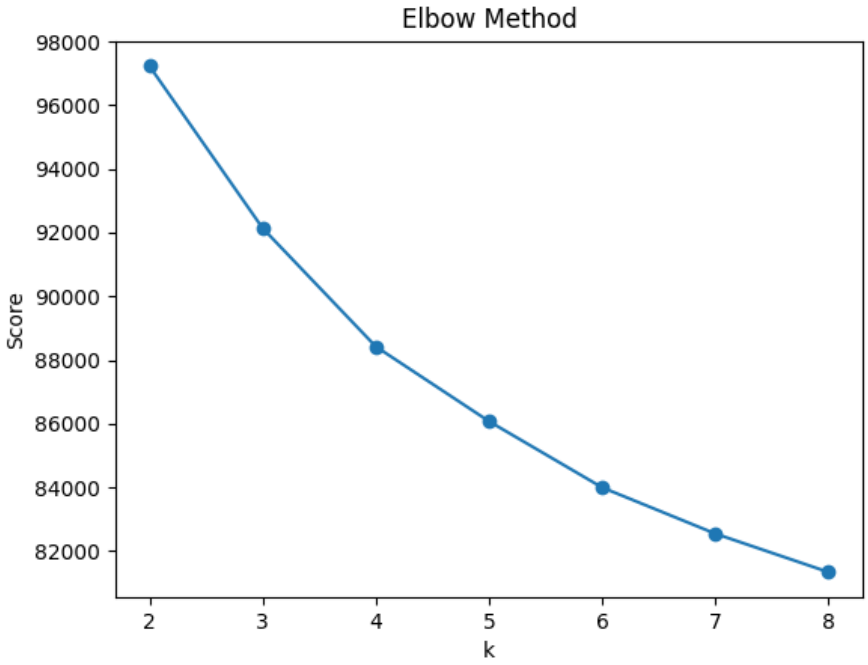


Fig 5.27: Resultado método del codo

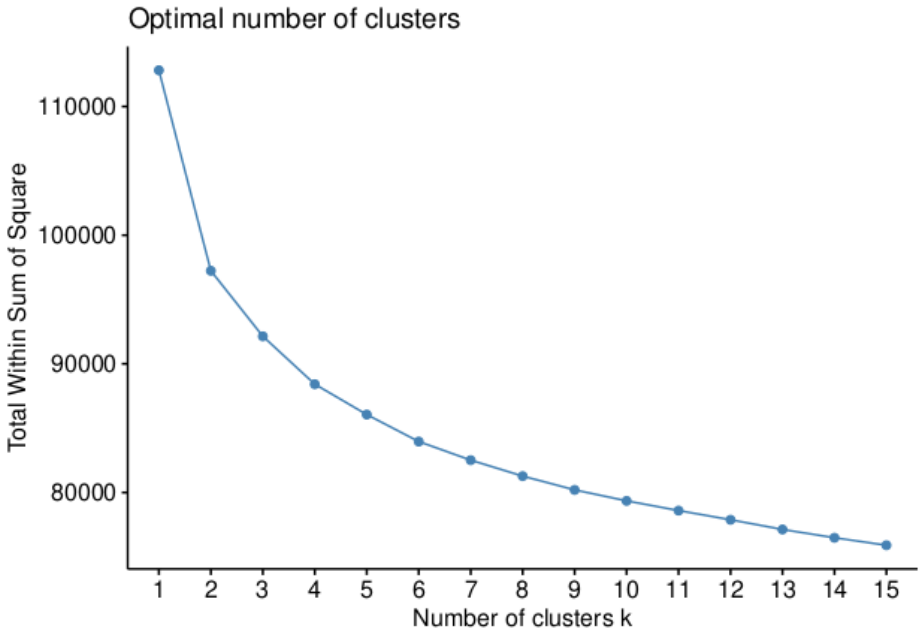


Fig 5.28: Resultado original método del codo

Basándose en los resultados obtenidos en el método del codo, la configuración de K-Means será la que se aprecia en la Fig 5.29. Generar este modelo finaliza con el proceso propuesto en la investigación original.

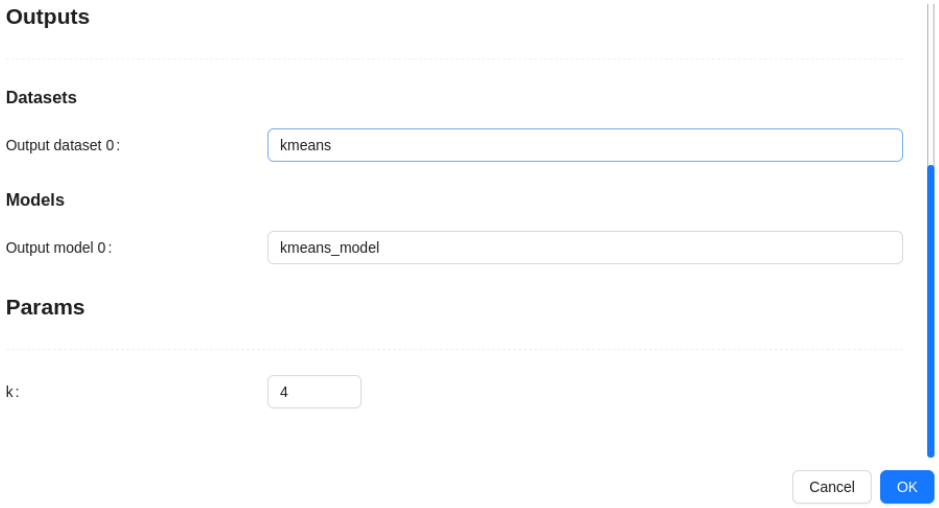


Fig 5.29: K-Means

5.5. Comparación de Resultados

Al contrastar lo resultados de la experimentación original con los obtenidos al reproducir el experimento mediante la herramienta propuesta se obtiene el siguiente análisis:

- Los resultados de todos los algoritmos son idénticos con excepción de PCA. Esto se puede dar por múltiples razones, en este caso se lo atribuye a posibles diferencias entre las implementaciones en R y Python, como por ejemplo diferencias en la precisión o enfoque al ejecutar PCA. No obstante estas diferencias no influyen en el resto del proceso.
- Al comparar el reporte en PDF generado mediante R Markdown con el reporte generado por la herramienta se favorece el resultado obtenido con R Markdown. Esto se debe a que no se da la posibilidad de incluir descripciones en cada paso del experimento como se hace en R Markdown haciendo que el reporte resultante de la herramienta solo contenga gráficas que, sin el contexto adecuado, pueden resultar complicadas de interpretar. El script de R Markdown completo se lo pone a disposición en el repositorio del proyecto mientras que el reporte generado mediante el artefacto propuesto se lo tiene en el Anexo C.
- Al reproducir el experimento en la herramienta propuesta se pone a disposición de forma automática el modelo generado mediante la API REST, a diferencia del experimento original que, una vez aplicado el algoritmo de K-Mean, sólo se presentan los resultados mas no se hace ningún procedimiento para desplegar el modelo. Esto resulta en una clara ventaja para la herramienta puesto que los modelos resultantes quedan a disposición mediante una API.

6. Conclusiones y trabajos futuros

A continuación se presentan las conclusiones sobre el trabajo con respecto a los objetivos del proyecto para posteriormente analizar trabajos futuros y posibles mejoras.

6.1. Conclusiones

Como resultados del trabajo, se provee un prototipo de plataforma que facilita la experimentación científica. Las características principales del prototipo son: aprovisionamiento automático de los recursos necesarios, automatización de la ejecución del experimento basado en las fases definidas en la sección 4, finalmente la plataforma genera experimentos reproducibles de forma sencilla.

En cuanto al aprovisionamiento de recursos, la herramienta realiza este proceso gracias a la utilización de Docker para crear ambientes de ejecución con las dependencias necesarias y Docker-Compose para el despliegue de los recursos. En este trabajo se aprovisionó una infraestructura que incluye; Airflow, PostgreSQL, Python 3.8 tomando como base Ubuntu 20.04. Los recursos aprovisionados para un usuario poco experimentado resulta complicado instalar y configurar de forma exitosa.

La automatización del experimento se realizó en Airlfow, haciendo uso de un pipeline a partir de la descripción ontológica de un experimento, permitiendo que el proceso sea transparente para el usuario final. Nuevamente la facilidad que provee Docker y Docker-Compose ayudaron a que no exista dificultad al integrar Airflow con los demás componentes de la herramienta.

Por otra parte, la reproducibilidad de un experimento no representa un problema, puesto que todos los datos de los operadores son almacenados en la ontología, por lo tanto, solo se requiere ejecutar nuevamente el experimento para reproducirlo. Gracias a esto reproducir el caso de uso mencionado en 5 resulta sencillo y con resultados comparables a la experimentación realizada de forma manual.

Cabe recalcar que el uso de la ontología resultó sumamente beneficioso y clave en esta investigación, puesto que, a partir de esta representación se realiza la definición y despliegue de los recursos necesarios para un experimento, esto deriva en un pipeline que automatiza todo

el proceso de experimentación, y asegura la reproducibilidad del experimento. Todo gracias a que se definen de forma precisa los operadores involucrados, y a su vez ayuda a la definición de las distintas posibles versiones del experimento, así como las definiciones de los conjuntos de datos y sus respectivas versiones.

Para finalizar, a lo largo del escrito se expuso el desarrollo de un prototipo que permite la experimentación científica de Machine Learning basado en las guías MLOps cumpliendo el objetivo general. Además se automatizó la configuración y despliegue de servicios para un experimento mediante Docker, se automatizaron las fases del proceso MLOps que fueron discutidas en la sección 4 y se verificó la reproducibilidad de los experimentos al comparar los resultados con un proceso manual en la sección 5, cumpliendo con los objetivos específicos.

6.2. Trabajos futuros

Debido a que la plataforma presentada es únicamente un prototipo existen varias partes que se pueden mejorar. En primera instancia se buscaría presentar de mejor manera los reportes generados de forma automática, al dar la posibilidad de incluir descripciones de cada paso realizado, además de ofrecer opciones de personalización para los gráficos y tablas generadas durante el experimento. También se buscaría mejorar la ontología utilizada para la representación de experimentos, de forma que sea mas acorde a las guías MLOps. Finalmente se debería buscar aumentar operadores, modelos y ambientes de ejecución disponibles de forma que la herramienta sea más flexible y pueda ser usada en varios escenarios.

Dentro de algunas ideas que se plantea para extender el proyecto, se contempla posibilidades de escalar automáticamente de forma horizontal los servicios utilizados, aprovechando la independencia de cada uno de los componentes del sistema de manera que se puedan optimizar y aprovechar de mejor manera los recursos, y mejorar la cantidad de experimentos concurrentes que se pueden ejecutar. Esto podría lograrse aprovechando herramientas como Kubernetes que permiten generar réplicas de contenedores de Docker bajo demanda y balancear la carga entre cada réplica. Por otra parte se podría buscar integración con plataformas cloud de forma que no existan limitantes en cuestión de hardware para la ejecución de flujos de Machine Learning. Adicionalmente, se podría aprovechar la base de conocimiento que se genera al describir distintos algoritmos mediante una ontología generando motores de sugerencia que sirvan como guías al momento de describir experimentos. Es decir que, en términos de los operadores de ontología, se pueda buscar experimentos en donde se hayan usado dichos operadores para dar así una idea a investigadores novatos de los procesos que se pueden hacer para obtener resultados.

Finalmente entre las limitaciones del prototipo propuesto se encuentra la falta de opciones para integrar los modelos generados con otros sistemas de software de forma automática siendo esta un área de MLOps que no se tomó en cuenta para este proyecto, ya que simplemente se incluyó una predicción del modelo generado a manera de API REST en el sistema. Adicionalmente al estar restringidos para ambientes locales, la plataforma se puede ver limitada por los recursos de hardware disponibles de forma que la herramienta presentada no sea una opción viable dependiendo de la extensión de la investigación.

Referencias

Airflow. (2023). *What is airflow? — airflow documentation*. https://airflow.apache.org/docs/apache-airflow/stable/index.html

Alla, S., & Adari, S. K. (2021). *Beginning mlops with mlflow*. Apress. https://doi.org/10.1007/978-1-4842-6549-9

Allemang, D., Hendler, J., & Gandon, F. (2020). Semantic web for the working ontologist. *Semantic Web for the Working Ontologist*. https://doi.org/10.1145/3382097

Ansible. (2023). *Ansible documentation*. https://docs.ansible.com/

Casalicchio, E., & Iannucci, S. (2020). The state-of-the-art in container technologies: Application, orchestration and security. *Concurrency and Computation: Practice and Experience*, *32*, e5668. https://doi.org/10.1002/CPE.5668

Containers, L. (2023). *Lxd documentation*. https://linuxcontainers.org/lxd/docs/latest/

de Ruiter, J., & Harenslak, B. (2021). *Data pipelines with apache airflow*. Manning Publications. https://www.oreilly.com/library/view/data-pipelines-with/9781617296901/

Design, A. (2023). *Components overview - ant design*. https://ant.design/components/overview

Docker. (2023). *Reference documentation | docker documentation*. https://docs.docker.com/reference/

DVC. (2023). *Home | data version control · dvc*. https://dvc.org/doc

Felipe, A., & Maya, V. (2016). *The state of mlops*.

FISHER, R. A. (1936). The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, *7*, 179–188. https://doi.org/10.1111/J.1469-1809.1936.TB02137.X

Flask. (2023). *Welcome to flask — flask documentation (2.2.x)*. https://flask.palletsprojects.com/en/2.2.x/

Git. (2023). *Git - documentation*. https://git-scm.com/doc

Google. (2023). *Vertex ai | google cloud*. https://cloud.google.com/vertex-ai#section-5

Hewage, N., & Meedeniya, D. (2022). Machine learning operations: A survey on mlops tool support. https://doi.org/10.48550/arXiv.2202.10169

Hitzler, P., Krötzsch, M., & Rudolph, S. (2009). *Foundations of semantic web technologies*. Chapman & Hall/CRC.

Hwang, K. S., Park, K. S., Lee, S. H., Kim, K. I., & Lee, K. M. (2018). Autonomous machine learning modeling using a task ontology. *Proceedings - 2018 Joint 10th International Conference on Soft Computing and Intelligent Systems and 19th International Symposium on Advanced Intelligent Systems, SCIS-ISIS 2018*, 244–248. https://doi.org/10.1109/SCIS-ISIS.2018.00051

Jenkins. (2023). *Jenkins user documentation*. https://www.jenkins.io/doc/

Jinja. (2023). *Api — jinja documentation (3.1.x)*. https://jinja.palletsprojects.com/en/3.1.x/api/#basics

Knime. (2023). *Knime workbench guide*. https://docs.knime.com/latest/analytics_platform_workbench_guide/index.html#workspaces

Kreuzberger, D., Kühl, N., & Hirschl, S. (2022). Machine learning operations (mlops): Overview, definition, and architecture. https://doi.org/10.48550/ARXIV.2205.02302

Kubeflow. (2023). *Documentation | kubeflow*. https://www.kubeflow.org/docs/

Kubernetes. (2023). *Kubernetes documentation | kubernetes*. https://kubernetes.io/docs/home/

Microsoft. (2023). *What is azure machine learning? - azure machine learning | microsoft learn*. https://learn.microsoft.com/en-us/azure/machine-learning/overview-what-is-azure-machine-learning

Miñón, R., Diaz-De-arcaya, J., Torre-Bastida, A. I., & Hartlieb, P. (2022). Pangea: An mlops tool for automatically generating infrastructure and deploying analytic pipelines in edge, fog and cloud layers. *Sensors (Basel, Switzerland)*, *22*. https://doi.org/10.3390/S22124425

Mlflow. (2023). *Mlflow documentation — mlflow 2.1.1 documentation*. https://mlflow.org/docs/latest/index.html

Ontotext. (2023). *General — graphdb 10.2.0 documentation*. https://graphdb.ontotext.com/documentation/10.2/

Orange. (2023). *Orange data mining - getting started*. https://orangedatamining.com/getting-started/

Panov, P., Džeroski, S., & Soldatova, L. N. (2008). Ontodm: An ontology of data mining. *Proceedings - IEEE International Conference on Data Mining Workshops, ICDM Workshops 2008*, 752–760. https://doi.org/10.1109/ICDMW.2008.62

Pinzon, A. (2022). Https://medium.com/@pinnzonandres/iris-classification-with-svm-on-python-c1b6e833522c. https://medium.com/@pinnzonandres/iris-classification-with-svm-on-python-c1b6e833522c

Podman. (2023). *What is podman? — podman documentation*. https://docs.podman.io/en/latest/

Postgresql. (2023). *Postgresql: Documentation: 15: 1. what is postgresql?* https://www.postgresql.org/docs/current/intro-whatis.html

Potdar, A. M., Narayan, D. G., Kengond, S., & Mulla, M. M. (2020). Performance evaluation of docker container and virtual machine. *Procedia Computer Science*, *171*, 1419–1428. https://doi.org/10.1016/J.PROCS.2020.04.152

RapidMiner. (2023). *Home - rapidminer documentation*. https://docs.rapidminer.com/

React. (2023). *Getting started – react*. https://reactjs.org/docs/getting-started.html

Recupito, G., Pecorelli, F., Catolino, G., Moreschini, S., Nucci, D. D., Palomba, F., & Tamburri, D. A. (2022). A multivocal literature review of mlops tools and features. https://github.com/gilbertrec/MLR-MLOps-Tools-Features-Appendix

RedHat. (2023). *What is a hypervisor?* https://www.redhat.com/en/topics/virtualization/what-is-a-hypervisor

Redux. (2023). *Api reference | redux*. https://redux.js.org/api/api-reference

Ruf, P., Madan, M., Reich, C., & Ould Abdeslam, D. (2021). Demystifying mlops and presenting a recipe for the selection of open-source tools. *Applied Sciences*, *11*, 39. https://doi.org/10.3390/app11198861

Salvucci, R. D. E. G. C. D. A. B. C. E. (2021). Mlops-standardizing the machine learning workflow.

Sculley, D., Holt, G., Golovin, D., Davydov, E., Phillips, T., Ebner, D., Chaudhary, V., Young, M., Crespo, J.-F., & Dennison, D. (2015). Hidden technical debt in machine learning systems.

SELDON. (2023). *Seldon core — seldon-core documentation*. https://docs.seldon.io/projects/seldon-core/en/latest/index.html

Shahin, M., Ali Babar, M., & Zhu, L. (2017). Continuous integration, delivery and deployment: A systematic review on approaches, tools, challenges and practices. *IEEE Access*, *5*, 3909–3943. https://doi.org/10.1109/ACCESS.2017.2685629

Sinha, P. K., Gajbe, S. B., Debnath, S., Sahoo, S., Chakraborty, K., & Mahato, S. S. (2022). A review of data mining ontologies. *Data Technol. Appl.*, *56*, 172–204. https://doi.org/10.1108/DTA-04-2021-0106

Soldatova, L. N., & King, R. D. (2006). An ontology of scientific experiments. *Journal of The Royal Society Interface*, *3*, 795–803. https://doi.org/10.1098/RSIF.2006.0134

Taye, M. M. (2010). Understanding semantic web and ontologies: Theory and applications. *Journal of Computing*, *2*.

Terraform. (2023). *Terraform-docs*. https://terraform-docs.io/

Tianxing, M., Myint, M., Guan, W., Zhukova, N., & Mustafin, N. (2021). A hierarchical data mining process ontology. *undefined*, *2021-January*. https://doi.org/10.23919/FRUCT50888.2021.9347590

Valohay. (2023). *Valohai help*. https://help.valohai.com/hc/en-us

Vanschoren, J., & Soldatova, L. (2010). Expose: An ontology for data mining experiments.

W3C. (2012). *Owl - semantic web standards*. https://www.w3.org/OWL/

Weka. (2023). *Documentation - weka wiki*. https://waikato.github.io/weka-wiki/documentation/

Wirth, R., & Hipp, J. (2000). Crisp-dm: Towards a standard process modell for data mining. *undefined*.

Zhao, Y. (2020). Mlops and data versioning in machine learning project-industrial internship.

Zhou, Y., Yu, Y., & Ding, B. (2020). Towards mlops: A case study of ml pipeline platform. *Proceedings - 2020 International Conference on Artificial Intelligence and Computer Engineering, ICAICE 2020*, 494–500. https://doi.org/10.1109/ICAICE51518.2020.00102

Anexo A. Manual de Usuario

A continuación se describe a detalle como utilizar el sistema.

A.1. Administración de datasets

El sistema permite la creación y administración de distintos conjuntos de datos. Para esto en primer lugar se tiene que crear un *dataset*. Este *adminstrador de datasets* actúa como una forma de agrupar datos en diferentes versiones, de forma que si se desea probar un mismo proceso con diferentes variaciones de los datos, los cambios se pueden hacer de forma inmediata sin perder información previa.
Para crear un dataset se siguen los siguientes pasos que son ilustrados en la Fig A.1:

1. Seleccionar el administrador de datasets en la pantalla principal. Esto abrirá un listado con los distintos datasets que han sido importados al sistema.
2. Dar clic en el botón *New Dataset*. Esto abrirá un panel en el que se pide el nombre del nuevo dataset a crear.
3. Una vez se tiene el nombre del dataset se da clic al botón de confirmar y se creará un nuevo dataset del que se pueden crear diferentes versiones

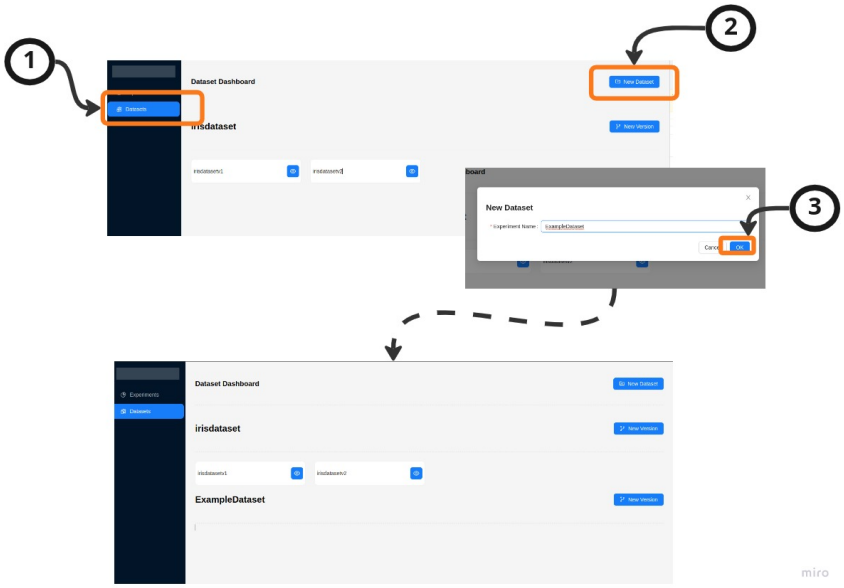


Fig A.1: Proceso de creación de dataset

A.1.1. Creación de versión de dataset

Una vez se tiene un *dataset* creado (refiérase a sección A.1 para mas información) se puede crear una versión del dataset el cual representa un conjunto de datos en específico.
Para crear un versión del dataset se siguen los siguientes pasos que son ilustrados en la Fig C.7:

1. En la pantalla de administración de datasets, dar clic al botón *New version* del dataset correspondiente. Esto abrirá una pantalla en la que se pide la información necesaria para una nueva versión del dataset.

2. Llenar la información necesario como nombre de la versión, un archivo en formato CSV que contenga los datos y el separador que corresponde al archivo. Una vez seleccionado el archivo se obtendrá una previsualización de los datos.

3. Una vez ingresada toda la información, dar clic en el botón de confirmar, esto importa los datos del archivo CSV en el sistema y los expone al resto de componentes del sistema. Este proceso puede tomar algunos segundos y al terminar se genera una nueva entrada en el administrador, desde la cual se puede hacer una previsualización de los datos.

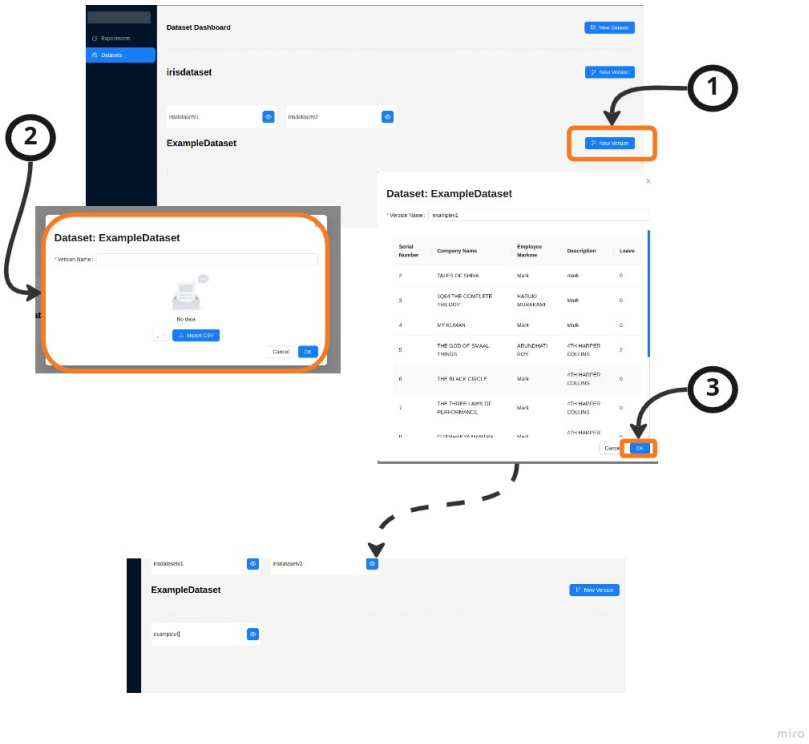


Fig A.2: Proceso de creación de version de dataset

A.2. Administración de experimentos

El sistema permite la creación y administración de distintos experimentos. Para esto en primer lugar se tiene que crear un *Experimento*. Este *experimento* actúa como una forma de agrupar datos en diferentes versiones, de forma que sea sencillo mantener varias versiones de un mismo proceso.

Para la creación de un *experimento* siga los siguientes pasos ilustrados en la Fig A.3

1. Seleccionar el administrador de experimentos en la pantalla principal. Esto abrirá un listado con los distintos experimentos que han sido creados en el sistema.
2. Dar clic en el botón *New Experiment*. Esto abrirá una ventana en la que se pide el nombre del nuevo experimento a crear junto a una descripción.
3. Una vez se tienen los datos completos, dar clic en el botón de confirmar, lo que creará una nueva entrada en el administrador.

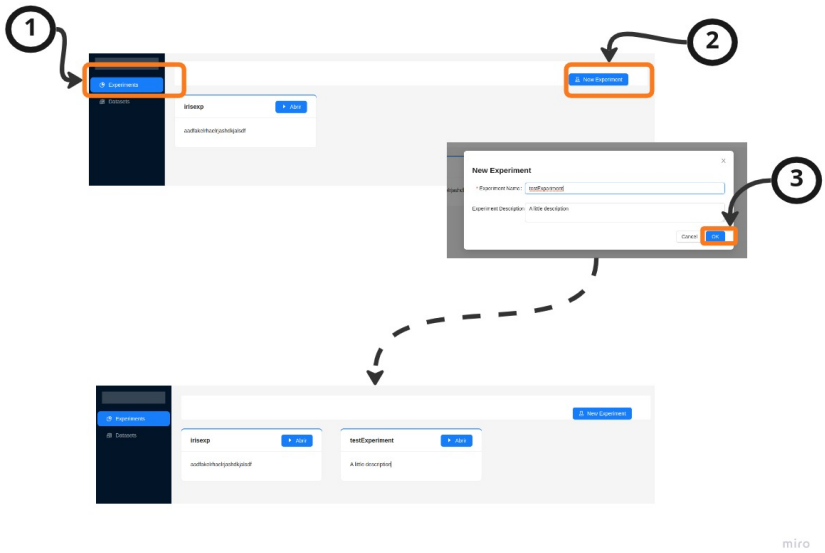


Fig A.3: Proceso de creación de experimento

A.2.1. Creación de versión de experimento

Una vez se tiene un *experimento* creado (refiérase a sección A.2 para más información) se puede crear una versión de dicho experimento el cual representa un pipeline en específico. Para crear un versión del experimento se siguen los siguientes pasos que son ilustrados en la Fig A.4:

1. En la pantalla de administración de experimento, dar clic al botón *abrir* del experimento correspondiente. Esto abrirá el editor desde el cual se pueden crear las diferentes versiones del experimento.
2. En la parte izquierda del editor se encuentra una barra lateral en la que se muestra un listado de las distintas versiones del experimento. Dar clic en el botón *New Version*.
3. Llenar el nombre de la versión y dar click en el botón *confirmar*. Esto genera una nueva entrada en el listado de versiones.

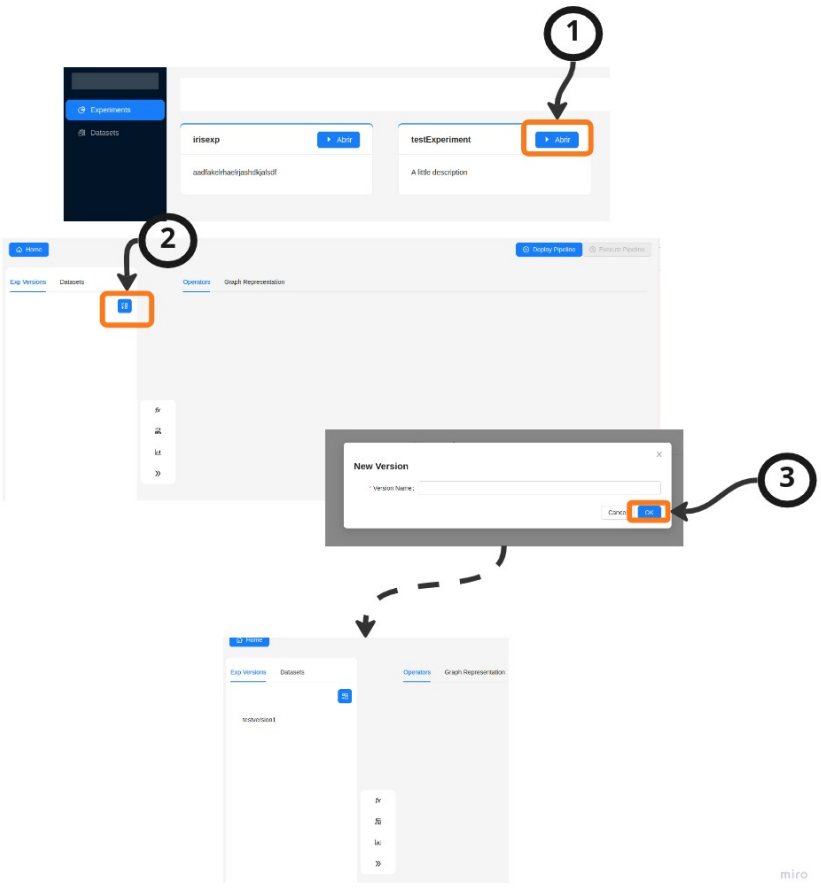


Fig A.4: Proceso de creación de versión de experimento

A.3. Descripción y ejecución de experimento

Si se han seguido todos los pasos descritos, se tiene todo listo para describir un pipeline y ejecutarlo, tal y como se indica en los siguientes pasos y en la Fig A.5:

1. Seleccionar la versión del experimento en el que se desea trabajar.
2. Desde la barra de operadores se puede seleccionar que operador agregar al pipeline. Al seleccionar un operador se abrirá un panel desde el que se puede ingresar los parámetros del operador correspondiente. En el ejemplo ilustrado se muestra el operador *Default Reader* el cual únicamente tiene una entrada y una salida. **NOTA -** El operador *Deaful Reader* es el único operador que puede leer una versión de un dataset previamente importando. Estos datasets pueden ser visualizados cambiando la vista de la barra lateral.
3. Una vez ingresada la información del operador dar clic en el botón *confirmar* el cual generará una nueva entrada en la lista de operadores desde la que se puede ver toda la información ingresada previamente.

4. En caso de ser necesario se puede editar los parámetros de un operador dando clic en el botón *edit* de la entrada del operador.

5. Agregar operadores hasta tener descrito cada uno de los pasos del proceso. **NOTA -** Se puede cambiar la vista de lista a una vista de grafo en la que se puede observar con mas claridad cual es proceso que se esta generando.

6. Una vez se tienen todos los operadores deseados se puede desplegar el pipeline dando clic en el botón *Deploy Pipeline* que empieza el proceso de aprovisionamiento. Esto puede tardas unos minutos la primera vez que se ejecute.

7. Al finalizar el proceso de aprovisionamiento se habilitará el botón de *Start Pipeline* el cual ejecutará la versión del experimento que se tenga seleccionada en ese momento.

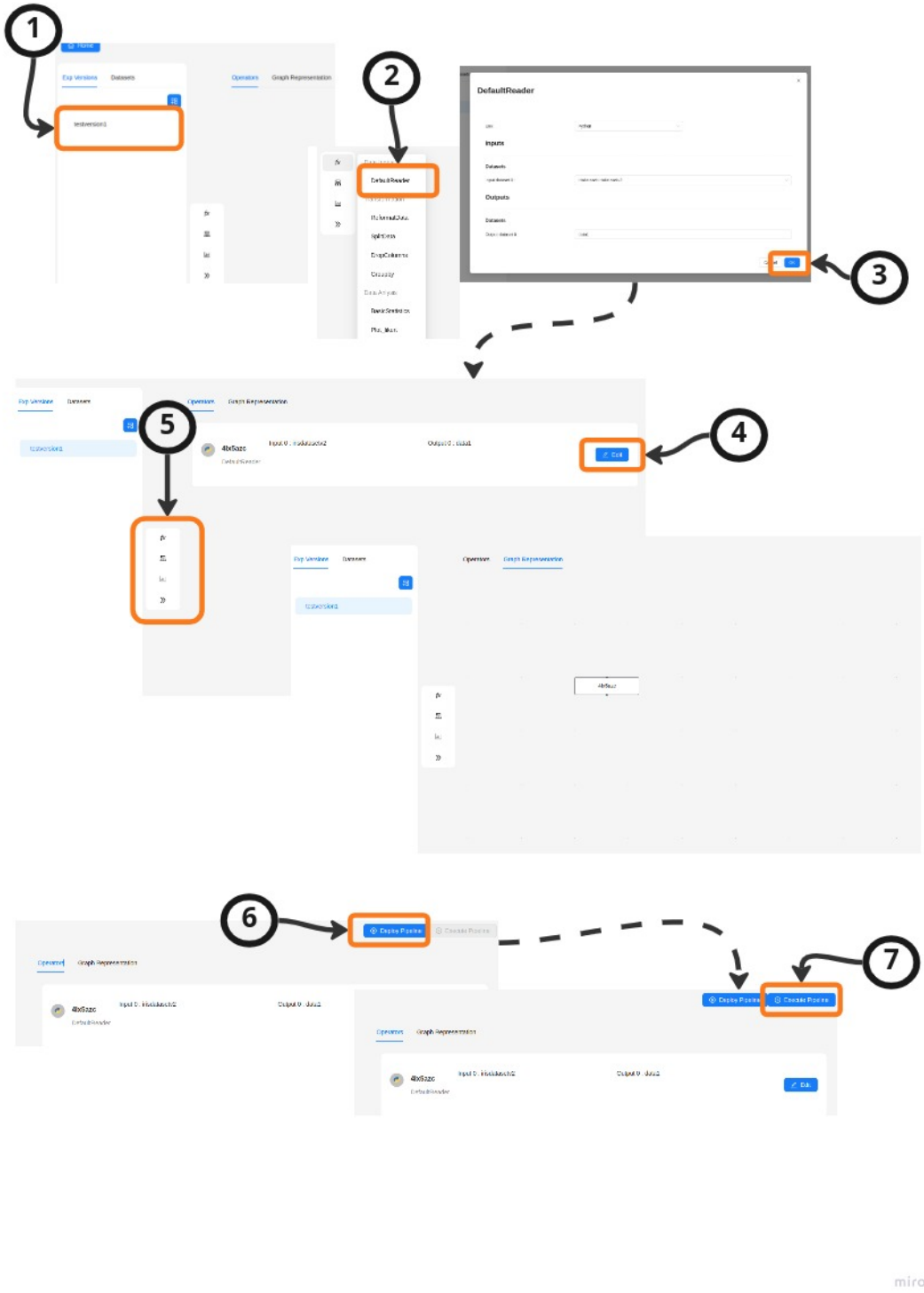


Fig A.5: Proceso de definición y ejecución de pipeline

Anexo B. Manual de Desarrollo

El siguiente manual expone en mayor detalle cada uno de los componentes que conforman el sistema. Para detalles de la arquitectura y como fue ideado el sistema referirse a la sección 4. El código fuente del proyecto se encuentran en repositorio del proyecto.

B.1. Instalación de sistema

El sistema tiene varias dependencias por lo que se provee un script que instala y configura todos los requisitos necesarios. A continuación se describe como instalar el sistema mediante dicho script y es ilustrado en la Fig B.1.

1. Ingresar al repositorio del proyecto y descargar el proyecto.
2. Descomprimir el archivo zip descargado previamente.
3. Dentro de la carpeta del proyecto, dar clic derecho, propiedades al archivo *install.sh*.
4. Habilitar la opción *Ejecutar como programa*.
5. Dar clic derecho al archivo *install.sh*, ejecutar como programa.
6. Esperar a que la instalación de las dependencias termine

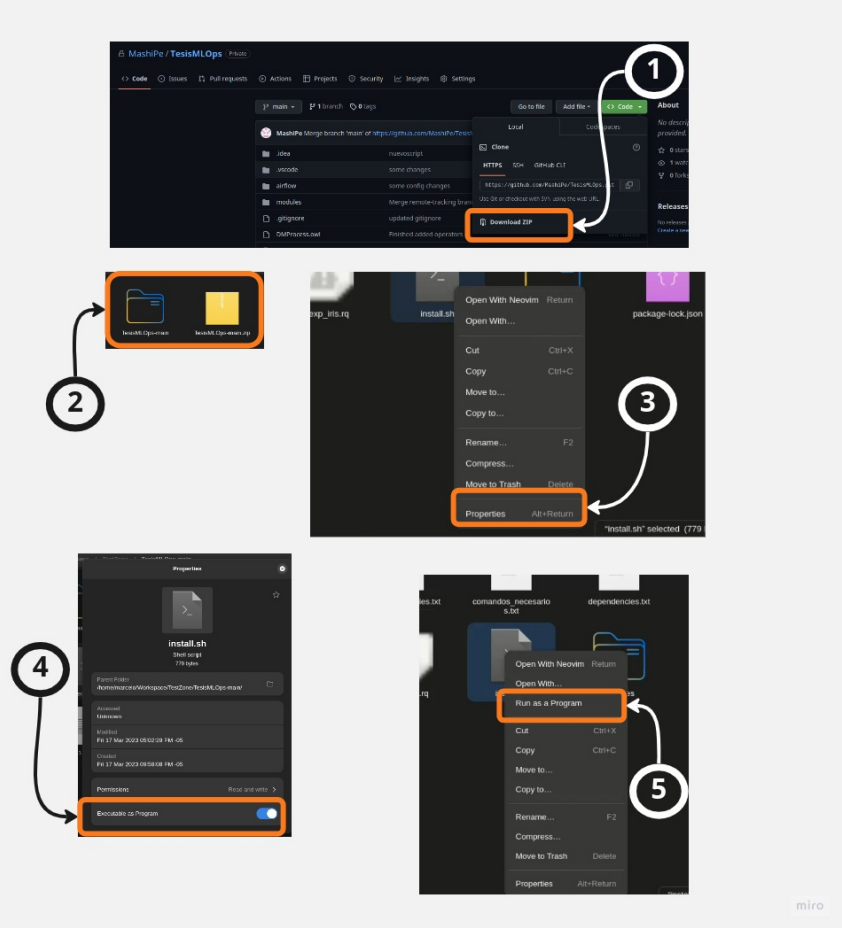


Fig B.1: Proceso de instalación del sistema

B.2. Ejecución de sistema

Antes de ejecutar el programa asegúrese de que todas las dependencias se encuentran instaladas (refiérase a la sección B.1 del manual de usuario). Para mayor comodidad se provee un script que ejecutar todo el sistema, el proceso se describe a continuación y es ilustrado en la Fig B.2.

1. Dentro de la carpeta del proyecto, dar clic derecho, propiedades al archivo llamado *execute.sh*.
2. Habilitar la opción *Ejecutar como programa*.
3. Dar clic derecho al archivo *execute.sh*, ejecutar como programa.
4. Esperar a que el sistema termine el proceso de inicialización.
5. Al terminar el proceso de inicialización entrar al siguiente link.

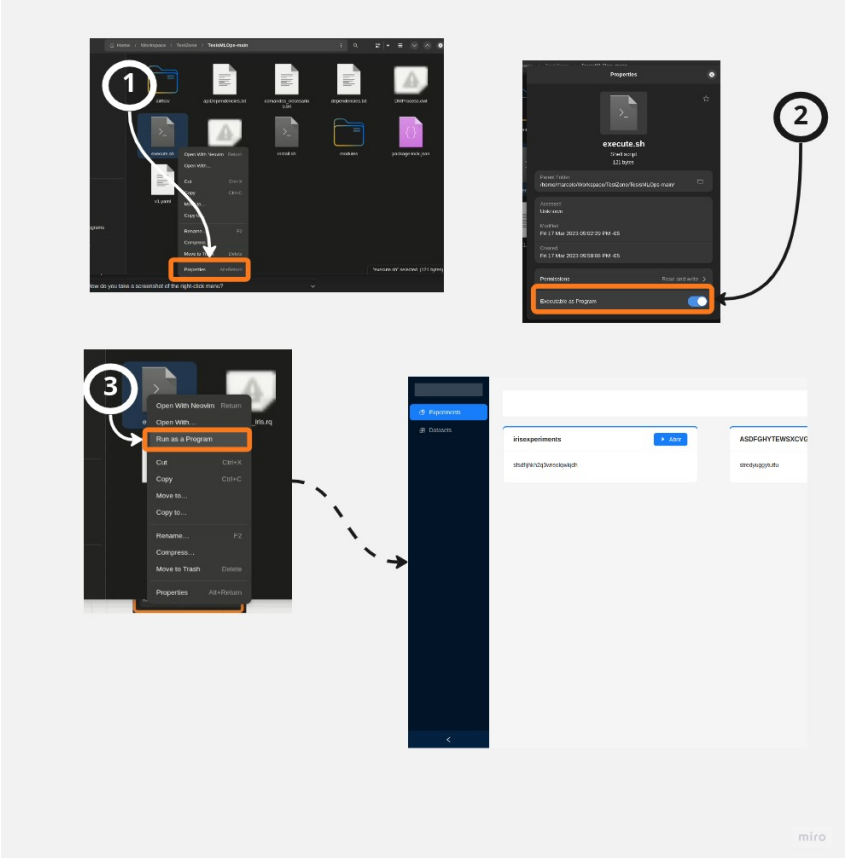


Fig B.2: Proceso de ejecución del sistema

B.3. Dependencias

El sistema fue desarrollado principalmente para sistemas Linux en distribuciones basadas en Ubuntu no obstante el sistema fue probado con con éxito en distribuciones basadas en Arch. Los tecnologías utilizadas en el desarrollo son:

- **Python:** 3.10
- **Docker:** 1.23.0 ***NOTA*** Si se presentan leaks de memoria al momento de ejecutar Airflow utilizar la versión 1.20.10 de Docker.
- **React:** 18.2.0
- **Redux:** 8.0.5
- **Airflow:** 2.5.0
- **Graphdb:** 10.1.0
- **PostgreSQL:** 13.0

Las librerías de desarrollo son las mismas que las dependencias para la ejecución del sistema por lo que se puede utilizar el mismo script de instalación llamado *install.sh*

B.4. Airflow

Todo lo relacionado a Airlfow incluyendo configuraciones y DAGs generados mediante el sistema se encuentran bajo el directorio *Airflow* del repositorio del proyecto.

- **Despliegue:** Airflow se despliega de forma local a modo de contenedores utilizando Docker-Compose cuyo archivo se encuentra directamente el directorio de *Airflow*. Este Docker-Compose se encarga de desplegar todas las dependencias de Airflow. **NOTA** también se encarga de ejecutar las bases de datos PostgreSQL, GraphDB y Redis.
- **Configuraciones:** Las configuraciones específicas de Airflow se encuentran bajo el directorio *Airflow/config* el cual es montado a modo de volumen en los contenedores de Docker de Airflow.
- **DAGs:** Para referencia, los DAGs generados se encuentran bajo el directorio *Airflow/dags*.

Una vez ejecutado Airflow se puede acceder a su interfaz web en el puerto 8080 utilizando el usuario y contraseña *Airflow* para ambos casos. **NOTA:** las credenciales para las bases de datos como PostgreSQL usan el mismo usuario y contraseña.
La comunicación de Airflow con el resto de módulos se realiza mediante la API REST que ofrece Airflow. Para utilizar esta API se requiere una autenticación de tipo simple utilizando las mismas credenciales para acceder a la interfaz web. Este comportamiento puede ser cambiado en el archivo de configuración de Airflow.

B.5. Módulos

El sistema se divide en varios módulos los cuales en conjunto forman los microservicios del sistema.

B.5.1. Aprovisionamiento

El aprovisionamiento se lo logra generando un archivo Docker-Compose con las imágenes y volúmenes requeridos. Todo el ambiente posee

- El contenedor ”ejecutor_scripts”:v1 el cual se encuentra en un repositorio publico, y se la puede descargar con el comando

- una red de Docker llamada Airflow_flow-net, esta red permite a la API comunicarse fácilmente con los contenedores.
- Se establece que el puerto de comunicación tanto en el host como en el contenedor sea el 4001
- Se montan 2 volumenes, uno dentro de los archivos del proyecto que es donde se encontrarán los Scripts, y otro en el directorio home del usuario donde se econtrarán las imágenes y PDFs generados Los archivos Docker-Compose y Dockerfile se los pueden encontrar en el repositorio del proyecto junto con los demás componentes.

B.5.2. Fetch

El módulo fetch se encarga de interactuar con GraphDB mediante la clase *DataFetcher*, y es utilizada principalmente por el servicio de API. Esta clase toma un diccionario de entrada y ejecuta operaciones CRUD dependiendo del método al que se llame. Esta clase abstrae los detalles de la utilización de la API de GraphDB y la ejecución de querys SPARQL.
Las querys de SPARQL se basan en plantillas que se encuentran bajo la carpeta *querys* y se dividen en 3 tipos; *insert, fetch y delete*. Las primeras se encargar de insertar nueva información en la base de datos. Las plantillas de *fetch* se utilizan para recuperar información específica de la base de datos. Finalmente las plantillas de tipo *delete* se encargan de eliminar información de la base de datos.
Todas las plantillas deben contener prefijos para rdf, rdfs, la ontología, y un prefijo separado llamado *MLOps* el cual es utilizado para identificar a las entidades en la ontología. Por otra parte todos los datos son guardados en un grafo distinto al de la ontología.
Las plantillas son cargadas en tiempo de ejecución, y con cada query que se desea hacer, se toman los distintos parámetros y se los reemplaza en la plantilla SPARQL para su posterior ejecución en GraphDB. En el caso que se deseen agregar nuevas plantillas para dar soporte a diferentes querys de SPARQL, se las debe agregar bajo las carpetas de *query* según corresponda, y registrarlas con un nombre único en el diccionario *query_template_pahts* dentro del archivo *data_fetcher.py* indicando el path de la plantilla relativo a la carpeta de módulos.

B.5.3. RaidenUI

La interfaz gráfica del sistema consiste en una aplicación web que puede acceder desde cualquier navegador. La interfaz esta desarrollada con React como framework, Redux como

librería para el *state management*, Antd para facilitar el desarrollo de componentes y react router para el manejo de rutas.
La interfaz fue divida en los siguientes elementos:

- **Routes:** Estos elementos representan las diferentes rutas web que componen la interfaz, como el editor de experimentos, o las paginas de administración para datasets y experimentos.
- **Layouts:** Estos componentes actúan como esqueletos para las rutas y tratan de agrupar elementos de interfaz comunes como lo son barras de navegación, barras laterales, etc.
- **Components:** Estos son pequeños pedazos de interfaz que son utilizados como bloques de construcción por los componentes *Routes y Layouts* para generar las interfaces. En estos componentes se incluyen diferentes modales para recibir información, cartas visuales que despliegan diferente información o generar listas de distintos elementos.
- **Store:** Dentro de estos elementos se hace uso de Redux para administrar la información general de la aplicación web. La información incluye listas de los experimentos junto a información de sus diferentes versiones, lista de datasets junto a previsualizaciones de los datos y un apartado específico que se utiliza al momento de editar un experimento dado.

Generación de formularios para operadores

Para evitar la tediosa tarea de hacer un formulario en específico para cada operador, se optó por generarlos basándose en descripciones de cada operador. Estas definiciones se encuentran bajo el elemento *Store* en el directorio *OperatorDefinitionSlice*. La descripción de un operador se ejemplifica a continuación:

```
{"EncodeLikert":{
    inputDef: {
        datasetInputs: 1,
        modelInputs: 0 },
    outputDef: {
        datasetOutput:1,
        modelOutputs: 0,
        graphicsOutput:0 },
    paramsDef:[
```

```
        {
            name:"columns",
            type:"list" },
        {
            name:"values",
            type:"map" }
        ]
    }
}
```

Esta definición comprende tres partes; la primera, bajo la llave *inputDef*, define las entradas del operador en donde se indica la cantidad de datasets y la cantidad de modelos que debe recibir el operador. La segunda parte bajo la llave de *outputDef* determina las salidas del operador indicando el numero de dataset de salida, modelos generados y gráficos según corresponda. Finalmente la llave *paramsDef* permite indicar parámetros específicos para cada operador y se debe indicar tanto el nombre del operador como el tipo de entrada que se maneja. Actualmente los tipos soportados son *list, map, string, number, map y complexMap*. Estos elementos se deben incluir bajo una llave con el nombre del operador. **NOTA:** El nombre del operador debe coincidir con la identificación del operador de la ontología.

Adicionalmente se deben definir valores por defecto para cada operador, por ejemplo:

```
{"EncodeLikert": {
    env:"Python",
    input:[""],
    output:[""],
    op_type:"EncodeLikert",
        parameters:{
            "columns":[],
            "values":{},
        }
    ,op_name:""
    },
}
```

Estos valores únicamente son utilizados como valores iniciales al generar los formularios. La definición y los valores por defecto de un operador son registrados en los diccionarios *glob-*

alDefinitions y defaultValues respectivamente.

Finalmente el operador se debe registrar con su nombre en el diccionario *operatorGroups*, ya sea en alguna de las categorías que ya existe o agregando una nueva tomando como ejemplo las ya existentes. Una vez completada toda la información se debe reiniciar el sistema y si todo es correcto se tendrá acceso al operador desde el editor de experimentos.

B.5.4. Template

El sistema de templates se basa en el motor de plantillas *Jinja*. El módulo hace uso de la clase *Pipe_Generator* que se encuentra definida en el archivo *dag_generator.py* y es usado principalmente en el servicio API del sistema y se encarga de generar los DAGs de Airflow a modo de scripts de python, además de generar la base de datos para el experimento en específico junto al *inifile*, el cual es un archivo que contiene la información de conexión a la base de datos para el experimento, esta información incluye host, nombre de la base de datos, usuario, contraseña y puerto.

El proceso que sigue es el siguiente: En primera instancia se crea la base de datos temporal que se usará durante la ejecución del experimento junto al *inifile* correspondiente. Posteriormente se genera cada *task* de Airflow en función de cada operador que exista para el experimento. Esto se realiza tomando el template correspondiente según el tipo de operador con el que se trabaja. Cada template se basa en la misma forma, una lista de entradas bajo el nombre de *input*, una lista de salidas bajo el nombre de *output*, el nombre del operador junto a la versión del experimento a la que pertenece, el *inifile* para la conexión con la base de datos temporal y una serie de parámetros específica para cada operador. Una vez se tienen los tasks de Airflow para cada uno de los operadores se generan las dependencias de cada uno de los task, las cuales son definidas a modo de pares, el orden de estos pares importa siendo el segundo elemento dependiente del primero. Finalmente se toma toda esta información y se la inserta en un solo archivo el cual cumple con los formatos de Airflow y posteriormente se lo guarda en la carpeta de Airflow correspondiente.

Para agregar un nuevo template a este módulo se lo debe registrar en el diccionario *template_paths* que se encuentra dentro del archivo *dag_generator.py* usando como clave el tipo de operador y como valor el path del template. El path debe ser relativo al directorio *template* y la clave debe corresponder a un operador que se encuentre registrado en la ontología.

URL	Método	Tarea	Parámetros	Resultado, en caso de status 200
/exp/version	POST	Agrega una nueva versión del experimento a la ontología	Iri del experimento, información del nuevo experimento	Json con información de la nueva versión
/exp/version/<version_iri>	GET	Obtiene información del experimento correspondiente	Iri del expermento	Json con información de la version
/exp/version/operator	POST	Agrega un nuevo operador a la versión	Iri del experimento, operador a agregar	Json con información del operador
/exp/version/operator/update	POST	Actualiza un operador	Iri del experimento, operador a actualizar	Json con la información del operador
/exp/<exp_iri>	GET	Obtiene la información de un experimento	Iri del experimento,	Json con la información del experimento
/datasetlist	GET	Obtiene la lista de datasets	NA	Json con la lista de datasets

/explist	GET	Obtiene la lista de experimentos	NA	Json con la lista de experimentos
/newexp	POST	Agrega un nuevo experimento a la ontología	Iri del experimento	Json con la información del experimento
/newdataset	POST	Agrega un nuevo dataset	Información del dataset	Json con la información del dataset
/newdatasetversion	POST	Agrega una nueva versión del dataset	Archivo csv del dataset, nombre de la versión del dataset	Json con la información del dataset
/genpipeline	POST	Genera el pipeline para su ejecución en Airflow	Nombre del experimento, versión del experimento	Un script de Airflow
/desplegar	POST	Despliega los componentes del sistema	Lista de servicios a desplegar	0
/gettable	GET	Obtiene la tabla deseada de la base de datos de persistencia	Nombre de la tabla a consultar	Json con los registros de la tabla

/predict/K-Means	GET	Realiza la predicción de un modelo de K-Means previamente entrenado	Nombre del experimento, versión del modelo, parámetros para la predicción	String con el valor resultante de la predicción
/getcolumns	GET	Obtiene las columnas de una tabla	Nombre de la tabla a consultar	Json con las columnas

Table B.1: Endpoints API principal

B.5.6. Scripts

Nombre Script	Parámetros	Lenguaje	Tarea	Resultados
conf_matrix	Tabla de entrada, modelo, archivo ini	Python	Obtener la matriz de confusión	Una tabla en la base de datos con la matriz de confusión
csv_to_PostgreSQL	Archivo csv, tabla destino, archivo ini, separador	Python	Leer el archivo csv y almacenarlo en la base de datos	Una tabla en la base de datos
density	Tabla de entrada, labels, archivo de salida de la imagen, archivo ini	Python	Obtener el histograma correspondiente a una columna	Un archivo .jpg del histograma
drop_column	Tabla de entrada, tabla salida, archivo ini, lista de columnas	Python	Borra las columnas indicadas	Una tabla en la base de datos
elbow	Tabla de entrada, nombre del archivo de salida, k mínimo y máximo, columnas a incluir en el análisis	Python	Realiza el método del codo	Un archivo de jpg con la gráfica del método

encode_categorical	Tabla de entrada, tabla de salida, archivo ini, diccionario de columnas a reemplazar con su respectivo nuevo valor	Python	Codifica las columnas indicadas con el valor especificado	Una tabla en la base de datos
generatePDF	Titulo del experimento, lista de imágenes	Python	Genera un reporte con todos los resultados de imagenes obtenidos	Un archivo PDF
get_correlation_graph	Tabla de entrada, tabla salida, archivo de salida, archivo ini	Python	Genera la matriz de coorelación	Una tabla en la base de datos y un archivo .jpg
groupby_colums	Tabla de entrada, archivo de salida, archivo ini, columnas del group by, función de agregación, columna a aplicar la agregación	Python	Realiza una operación group by sencilla a una tabla	Un archivo .jpg
K-Means	Tabla de entrada, k, nombre del archivo del modelo, archivo ini	Python	Realiza el archivo de K-Means	Una tabla en la base de datos y un archivo del modelo

pca_modelo	Tabla de entrada, componentes, tabla de salida, archivo ini	Python	Realiza el analisis de PCA	Una tabla en la base de datos
pivot	Tabla de entrada, tabla de salida, archivo ini, columnas	Python	Realiza una tabla pivote simple con las columnas indicadas	Una tabla en la base de datos
plot_all	Tabla entrada, archivo de salida, archivo ini	Python	Realiza un Scatter plot	Un archivo .jpg
plot_likert_columns	Tabla de entrada, archivo de salida, archivo ini	Python	Realiza un plot de las columnas seleccionadas para la escala de likert	Un archivo .jpg
s_score	Tabla de entrada, archivo de salida, k mínimo y máximo, columnas	Python	Realiza el silhouette score y lo grafica	Un archivo .jpg
split	Tabla de entrada, tabla entrenamiento, tabla test, tamaño del split, archivo ini	Python	Realiza el split de una tabla en dataset de entrenamiento y prueba	Dos tablas en la base de datos

summary	Tabla entrada, tabla salida, archivo ini	Python	Realiza un resumen de las columnas numéricas de una tabla	Una tabla en la base de datos
SVM	Tabla entrenamiento, nombre del modelo, kernel, archivo ini	Python	Realiza el entrenamiento del SVM	Un archivo del modelo
table_to_image	Tabla entrenamiento, archivo salida, archivo ini	Python	Convierte parte de una tabla en la base de datos a un archivo .jpg	Un archivo .jpg
test_barlett	Tabla entrenamiento, archivo imagen	Python	Realiza el test_barlett, análisis KMO y análisis de factores	Un archivo.jpg

Table B.2: Scripts

B.6. Trabajando con la ontología

La definición de la ontología se encuentra en el archivo *DMProcess.owl* en la raíz del proyecto. Este archivo puede ser abierto utilizando Protege para luego ser subido a GraphDB.
El archivo fue reutilizado del artículo Tianxing et al., 2021 con ligeras modificaciones para cumplir mejor con las necesidades del proyecto. Para mas detalles de la ontología referirse al artículo.

B.7. Agregando nuevos operadores al sistema

A continuación se presenta un pequeño tutorial para agregar nuevos operadores. El proceso se realiza asumiendo que se lo va a desarrollar para un ambiente de python.
Tomemos como ejemplo el operador *Drop_Columns* el cual toma un conjunto de datos y elimina las columnas que se indiquen como parámetro.
En primera instancia se tienen que identificar las entradas y salidas las cuales en este caso consisten en una tabla de entrada y una tabla de salida. Posteriormente los parámetros que en este caso sería una lista con los nombres de las columnas que se desean eliminar de una tabla.

B.7.1. Creación de script

En este punto se deben tomar en cuenta dos aspectos importantes. En primer lugar se debe tomar en cuenta el esquema genérico que toman todos los scripts el cual fue presentado en 4.2.3. En segundo lugar es que el script será ejecutado en el ambiente de ejecución por lo que se tiene disponible cualquier librería o recurso extra dando mayor flexibilidad para el desarrollo de scripts. Para este script se utiliza la librería pandas para la manipulación de la tabla de datos.
NOTA: en caso de que el ambiente no cuente con alguna librería que se desea utilizar se la puede agregar modificando el *Dockerfile* del ambiente en cuestión y volviéndolo a construir.
El primer paso es tomar todas las entradas y parámetros del script. El *ejecutor de scripts* provee toda esta información mediante la linea de comandos codificado en una cadena JSON como único argumento. Para evitar conflictos con la linea de comandos la cadena utiliza asteriscos en lugar de comillas por lo que es necesario cambiar este caracter previo a convertirlo en un diccionario manejable.
Una vez se tiene el diccionario con los datos necesarios se procede a recuperar los datos de la base de datos. Toda la información necesaria para la conexión a la base de datos se

provee a través del parámetro *ini_file* y que se encuentra ubicado en el directorio */root/scripts* del ambiente de ejecución. En este caso se utiliza la librería pandas para la lectura de estos datos.

Una vez se tienen los datos se puede ejecutar el proceso del script según se necesite. En este caso se toman las columnas que se encuentran en el parámetro *columns* y se las elimina una a una.

Finalmente los datos deben ser enviados a la misma base de datos de donde se recuperó la información previamente. En este caso también se hace uso de la librería pandas.

El script completo queda de la siguiente forma:

```python
import sys
import json
from sqlalchemy import create_engine
from config import config
import pandas

base="/root/scripts/"
if __name__ == '__main__':
    #Obtencion de parametros
    args = sys.argv
    json_str = args[1]
    data = json_str.replace("*", '"')
    data1 = json.loads(data)
    dataset_name = data1["table_input"]
    out_name = data1["table_output"]
    columns = data1["columns"]

    #Lectura de datos

    params = config(config_db=base+data1["ini_file"])
    conn_string = ("postgresql://postgres:pass@" + params["host"] +
                   "/" + params["dbname"] + "?user=" + params["user"] +
                   "&password=" + params["password"])
    engine = create_engine(conn_string)
    dataset = pandas.read_sql_query("select * from " + dataset_name.lower(),
                                    con=engine)
    dataset.drop('index', inplace=True, axis=1)

    #Proceso de elimincación

    for i in  columns:
        print(i)
        dataset.drop(i,inplace=True,axis=1)

    #Escritura de datos

    dataset.to_sql(data1["table_output"].lower(), con=engine, if_exists="replace")

```

Fig B.3: Ejemplo de script

Finalmente se debe colocar el archivo dentro del directorio *scripts* dentro del módulo *aprovisionamiento*.

B.7.2. Creación de template

Una vez se tiene el script se recomienda la escritura del template puesto que este paso sirve como puente entre Airflow y los ambientes de ejecución. En este paso se debe tener en cuenta el nombre de los parámetros utilizados en el script. Los templates siguen una lógica similar en todos los casos, primero se toman los parámetros de jinja y se los escribe a modo de variables de Python. Por razones de depuración se recomiendo realizar un output a la consola con los parámetros que se tienen. Enviar una petición REST al ambiente correspondiente con los parámetros en forma de JSON y el nombre del script correspondiente.
El template completo se lo presenta a continuación:

```
@task
def drop_columns{{input[0]}}_{{op_name}}_fun():

    inifile = '{{ inifile | default("NA",true) }}'
    in_dataset = '{{ input[0] | default('NA',true) }}'
    out_dataset = '{{ output[0] | default('NA',true) }}'
    columns = []
{% for key in columns %}

    columns.append('{{key}}')
{% endfor %}

    print( """ Maping data with parameters:
                in_dataset: {}
                out_dataset: {}
                columns: {}""".format(in_dataset,out_dataset,columns) )

    url = 'http://ejecutor:4001/ejecutarpython/drop_column.py'
    body ={'parametros': {'table_input':in_dataset,'table_output':out_dataset,
    'ini_file':inifile,'columns': columns } }

    x = requests.post(url, json = body)


    print(x.text)
```

Fig B.4: Ejemplo de template

Una vez se tiene el template se debe colocar el archivo dentro del directorio *operators* dentro del módulo *template*. Finalmente se debe agregar su respectiva entrada en el archivo *dag_generator.py* según se explicó en la sección *template* del manual.
Nota: para mas información del funcionamiento de los templates y sintaxis referirse a la documentación de Jinja.

B.7.3. Registro en ontología

Puesto que la ontología funciona como modelo de datos del sistema es necesario registrar el nuevo operador dentro de la ontología. Si bien esto se puede hacer mediante SparQL resulta mas sencillo hacerlo mediante el programa *Protege*.

Para registrar el nuevo operador en la ontología se debe abrir el archivo de la ontología ubicado en la raíz del proyecto bajo el nombre *DMProcess.owl* y agregarlo como un nuevo operador como se muestra en la siguiente imagen.

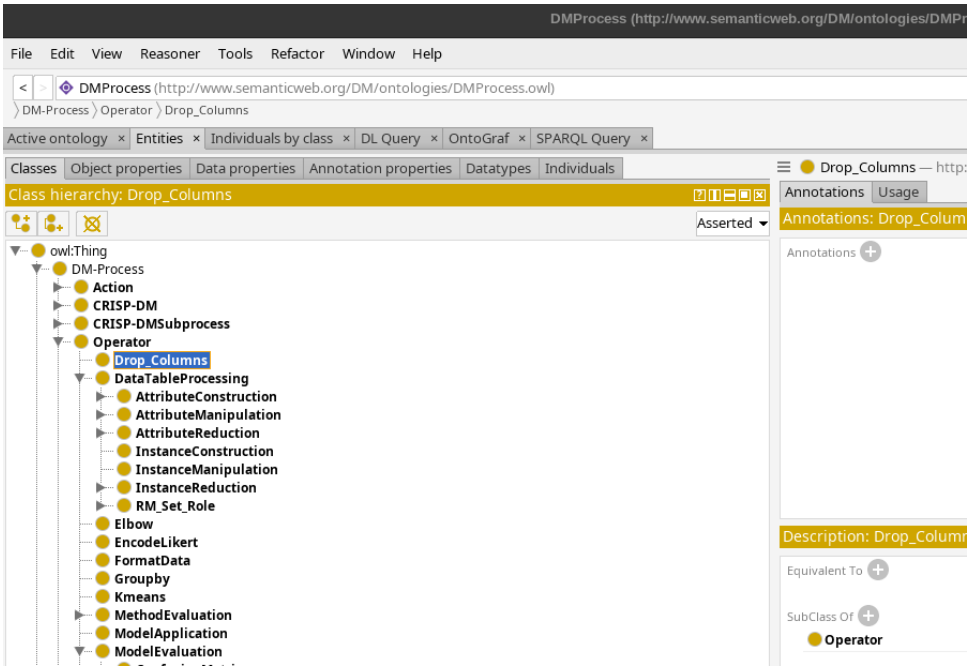


Fig B.5: Ejemplo de registro en ontología

Una vez se tiene el nuevo archivo de la ontología, se debe volver a importarlo en GraphDB previamente eliminando el grafo que contiene a la versión anterior de la ontología. **NOTA:** es importante importar la ontología en un grafo diferente al que se almacenan los datos, para mantenerlos aislados y no eliminar datos importantes en un futuro.

B.7.4. Registrando el nuevo operador en la interfaz gráfica

Finalmente se debe registrar el nuevo operador según se indica en la sección de RainenUI de este manual. En este caso las nuevas entradas serian las siguientes.

```
"DropColumns":{
    inputDef: {
        datasetInputs: 1,
        modelInputs: 0
    },
    outputDef: {
        datasetOutput:1,
        modelOutputs: 0,
        graphicsOutput:0
    },
    paramsDef:[
        {
            name:'columns',
            type:'list'
        }
    ]
} as OperatorDefinition,
```

Fig B.6: Ejemplo de definición de operación

```
'DropColumns':{
                env:'Python',
                input:[''],
                output:[''],
                op_type:'DropColumns',
                parameters:{
                    'columns':[]
                }
                ,op_name:''
            },
```

Fig B.7: Ejemplo de valores por defecto

```
operatorsGroups : {
        'Data Preparation':{
            groups: [ {
                title:'Data Ingest',
                operators: ['DefaultReader']
            } as subGroup,{
                title:'Transformation',
                operators: ['ReformatData','SplitData','DropColumns',
                            "Groupby","EncodeLikert"]
            } as subGroup
```

Fig B.8: Ejemplo de registro en categoría

Una vez realizados todos los pasos solo se debe reiniciar la plataforma y el nuevo operador estará disponible para su uso.

Anexo C. Reporte de Caso de uso

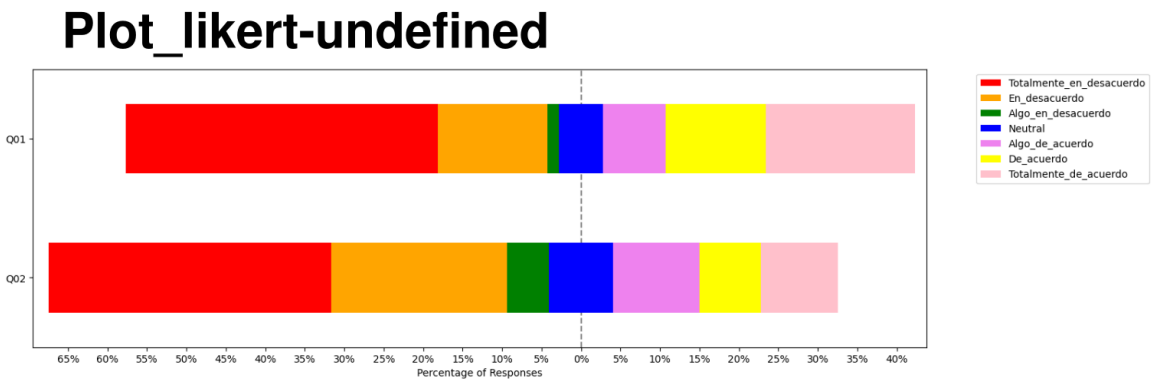


Fig C.1: Resultado de gráfica de likert

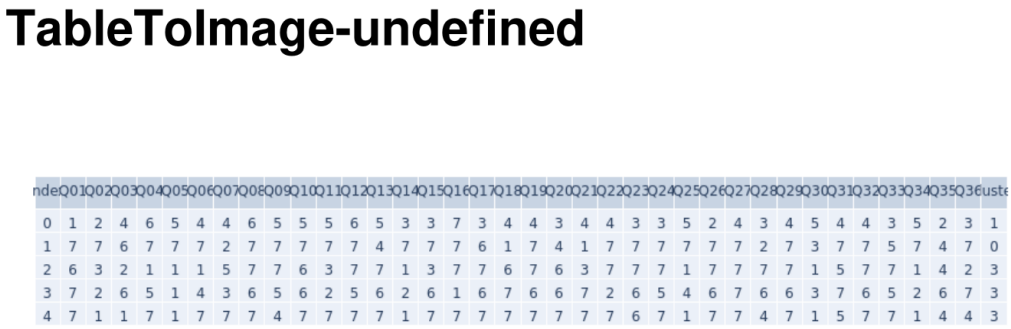


Fig C.2: Resultado K-Means

BasicStatistics-undefined

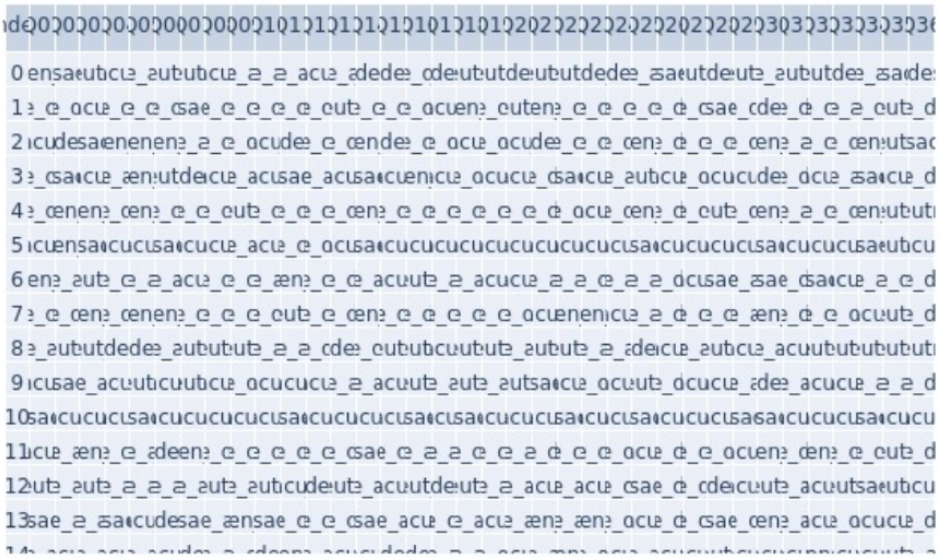


Fig C.3: Resultado de estadísticos básicos

TableToImage-undefined

value	Q01	Q02	Q03
Algo_de_acuerdo	65	90	90
Algo_en_desacuerdo	12	44	32
De_acuerdo	105	64	144
En_desacuerdo	115	184	106
Neutral	46	67	198

Fig C.4: Resultado Pivot

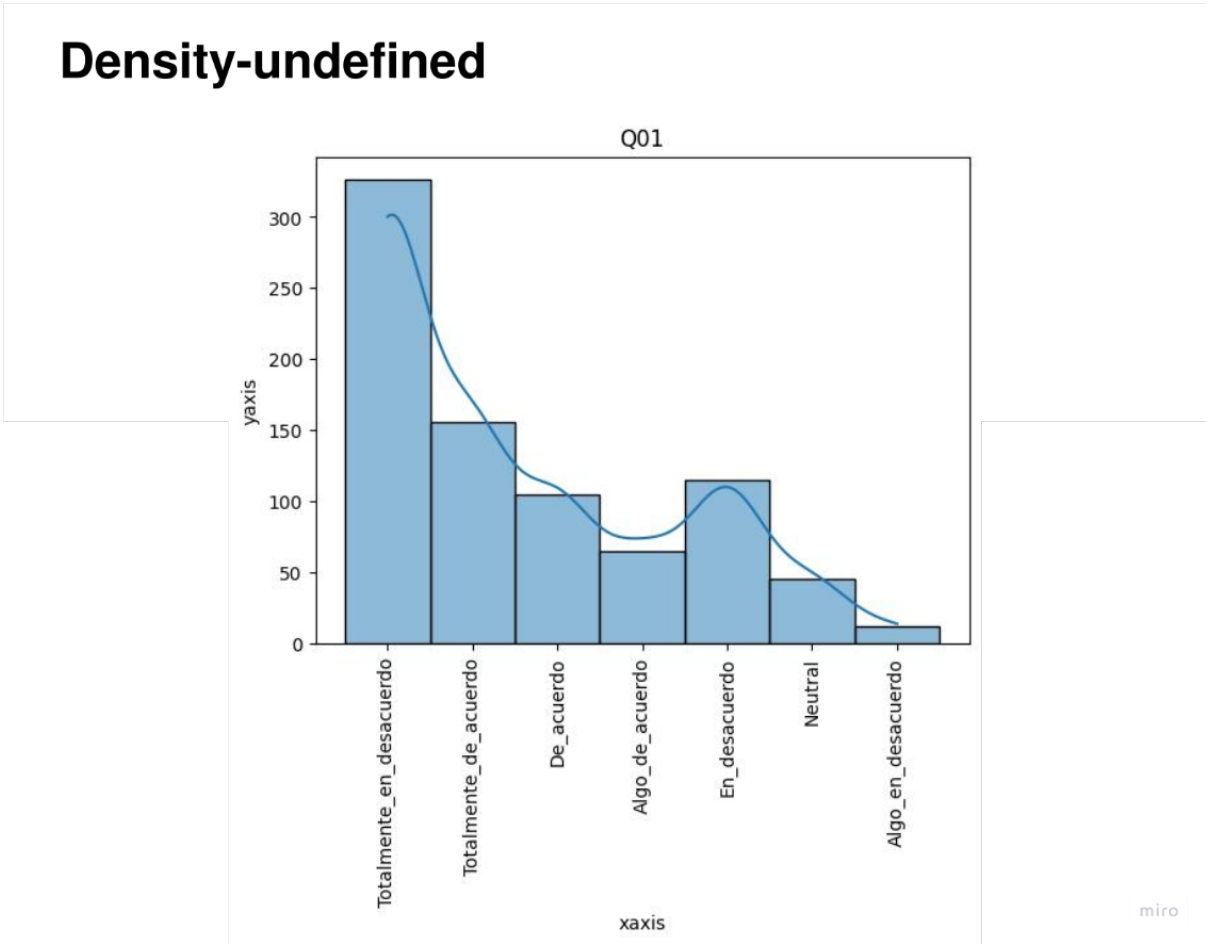


Fig C.5: Resultado de histograma

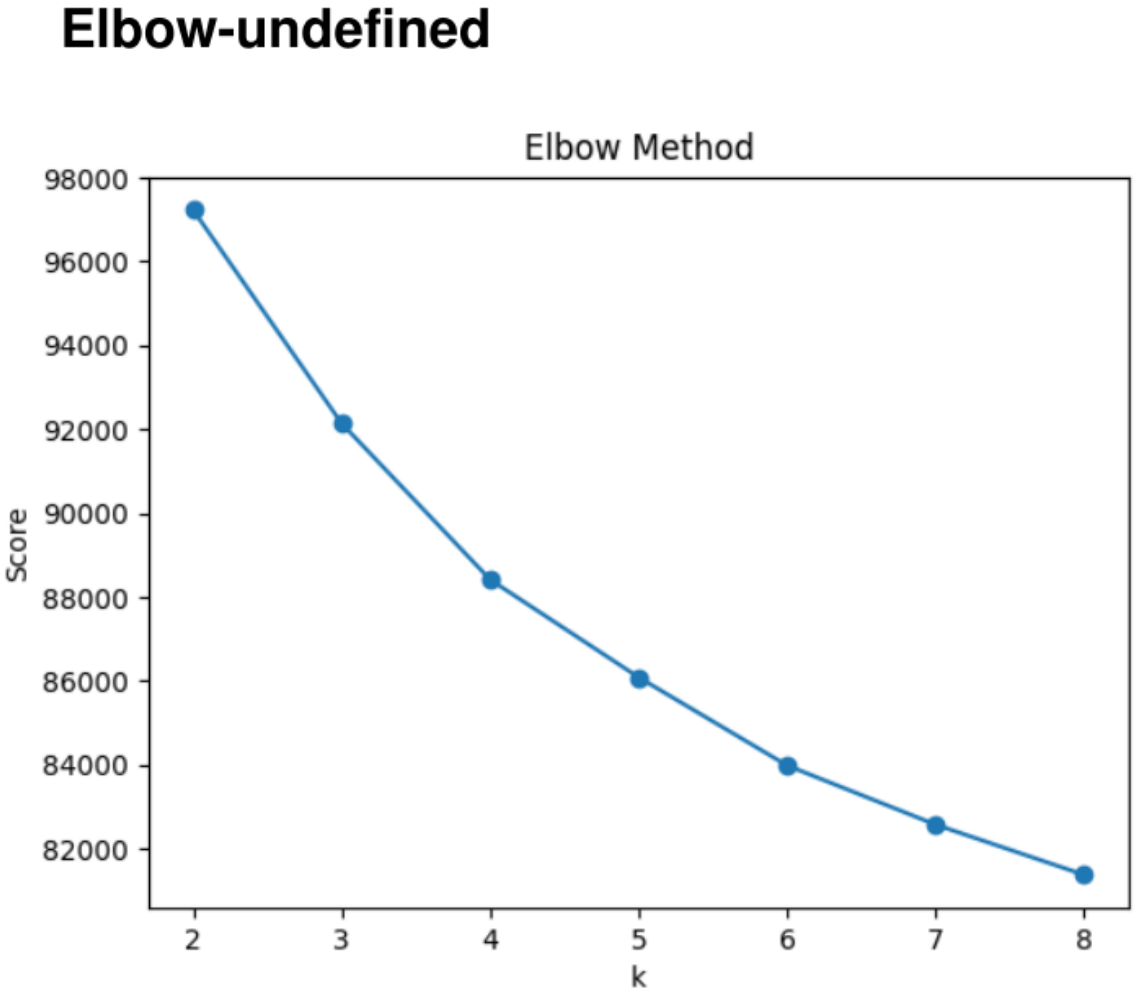


Fig C.6: Resultado de método del codo

TestBarlett-undefined

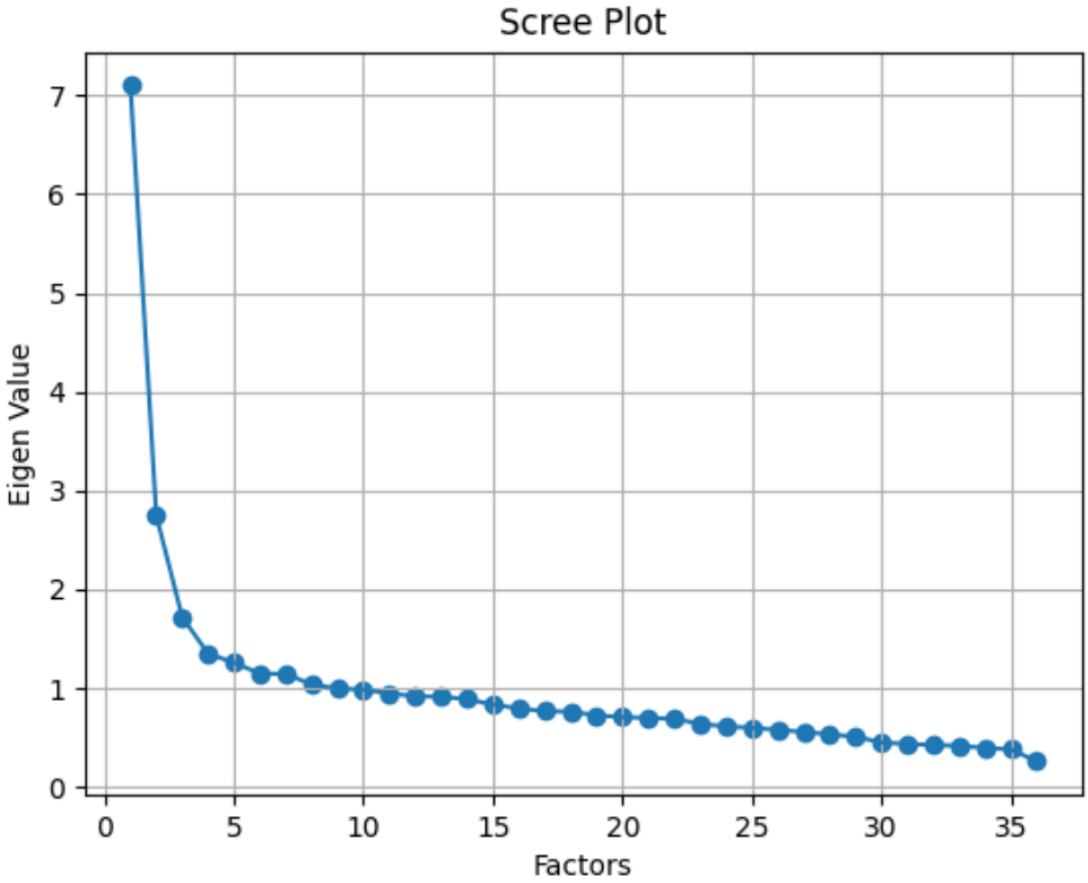


Fig C.7: Resultado de test de barlet