Instituto Tecnológico y de Estudios Superiores de Monterrey

Maestría en Inteligencia Artificial:

Reporte Final - Operaciones de aprendizaje automático

Equipo 43

Oscar Enrique García García A01016093

Alberto Campos Hernández A01795645

Jessica Giovana García Gómez A01795922

Esteban Sebastián Guerra Espinoza A01795897

Rafael Sánchez Marmolejo A00820345

Profesor Titular: Dr. Gerardo Rodríguez Hernández

Profesor Titular: Maestro Ricardo Valdez Hernández

Noviembre 18, 2025

Índice

1. Objetivo y Alcance

1.1. Objetivo Comercial

El crecimiento acelerado de la demanda eléctrica a nivel mundial, según la IEA [1], la demanda global aumentó en 2024 un 4,3 % en impulsada principalmente por China, las economías emergentes, la electrificación, los vehículos eléctricos y la expansión de centros de datos, está presionando a los sistemas de distribución para operar con mayor precisión y anticipación. En este contexto, las ciudades requieren herramientas que permitan prever su consumo energético con suficiente detalle para evitar sobrecargas, optimizar la gestión operativa y reducir costos asociados a desviaciones o picos inesperados. Tetuán no es la excepción, y contar con modelos de predicción confiables resulta clave para sostener la continuidad del servicio y cumplir con las crecientes exigencias del sistema.

Para abordar esta necesidad, se dispone del conjunto de datos público Power Consumption of Tetouan City, con información horaria o cada 10 minutos del consumo eléctrico de distintas zonas. Este dataset ofrece una base adecuada para estudiar patrones temporales, identificar variaciones entre zonas y construir modelos capaces de anticipar la demanda con suficiente antelación. Su granularidad y extensión permiten capturar ciclos diarios, semanales y estacionales, así como comportamientos anómalos que afectan la operación en tiempo real.

El objetivo central del proyecto es desarrollar una solución que permita predecir el consumo eléctrico en la Zona 2 de Tetuán y convertir esos pronósticos en información útil para la toma de decisiones operativas. Un modelo de este tipo habilita reducir sobrecostos por desviaciones, planificar mejor la carga y los recursos disponibles, y operar dentro de los límites técnicos establecidos. Además, constituye la base para incorporar en el futuro capacidades más avanzadas como gestión de demanda, optimización de activos energéticos o integración con mecanismos de mercado.

La propuesta se estructura en torno a dos componentes fundamentales. El primero es un modelo de predicción de consumo eléctrico, entrenado sobre el dataset de Tetuán, usando técnicas de aprendizaje automático para capturar patrones complejos y mejorar la precisión frente a variaciones inesperadas. Este modelo entregará pronósticos horarios o en ventanas de menor resolución, junto con métricas de calidad como RMSE o MAPE que permitirán evaluar su comportamiento y programar reentrenamientos cuando sea necesario.

El segundo componente es un servicio de pronóstico desplegado como una API que pueda integrarse fácilmente con sistemas operativos existentes como SCADA, EMS o BMS. Esta capa de servicio permitirá consultar el pronóstico en tiempo real, acceder al historial, generar actualizaciones periódicas y alimentar dashboards operativos. De esta manera, el modelo se convierte en una herramienta usable para operadores, ingenieros y equipos de planificación, facilitando una operación más anticipada, estable y eficiente del sistema eléctrico de Tetuán.

1.2. Objetivo Proyecto

El objetivo principal de este proyecto fue que el equipo 43, de la materia Operaciones de Aprendizaje Automático, por parte de la Maestría en Inteligencia Artificial del Tecnológico de Monterrey lograra industrializar un proyecto de Machine Learning mediante la adopción de las mejores prácticas de MLOps, asegurando la reproducibilidad, escalabilidad y mantenibilidad del modelo en todo su ciclo de vida.

Para lograr esto, se establecen los siguientes objetivos específicos:

- Estandarizar la estructura del código y los pipelines de desarrollo utilizando **Programación Orientada a Objetos (POO)** y plantillas de repositorio basadas en **Cookiecutter**.
- Implementar la trazabilidad y el versionamiento de los datasets y artefactos del modelo utilizando **Data Version Control (DVC)**.
- Establecer un sistema de seguimiento de experimentos utilizando **MLFlow** para registrar métricas, parámetros e hiperparámetros de manera centralizada.

- Crear un entorno de ejecución portable y reproducible para el modelo mediante la contenedorización con **Docker**.
- Desplegar el modelo como un **API REST** de alto rendimiento en un entorno serverless utilizando **Cloud Run**, dentro de una infraestructura cloud **(Google Cloud Platform)**.
- Desplegar una interfaz **frontend en Netlify**, que se conecta al servicio en Cloud Run, para facilitar la interacción y prueba del modelo por parte de los usuarios finales.
- Monitorear de forma continua todo el proceso utilizando **evidently** para asegurar el seguimiento integral del desempeño y la detección temprana de desviaciones.

1.3. Alcance

El alcance de este proyecto se define por la implementación y la integración de las siguientes funcionalidades y componentes clave:

1. Ingeniería de Software: Aplicación rigurosa de principios de Programación Orientada a Objetos (POO) en el desarrollo del código del modelo, pipelines de training y módulos auxiliares.
2. Estructura del Repositorio: Uso de CookieCutter para la generación de una estructura de repositorio estandarizada, incluyendo directorios para código fuente, notebooks, datasets, tests, y documentación.
3. Versionamiento de Datos: Configuración de DVC para el versionamiento de los datasets de entrenamiento y prueba, permitiendo la conmutación entre diferentes versiones de datos.
4. Tracking de Experimentos: Integración de MLFlow para registrar y comparar múltiples ejecuciones de entrenamiento, capturando el código fuente, configuración, parámetros y métricas de desempeño.
5. Contenedorización del Servicio: Creación de imágenes Docker optimizadas para la ejecución del modelo y del API de inferencia.
6. Despliegue del Backend: Disponibilización del API REST del modelo (para la inferencia) mediante su despliegue en Google Cloud Run.
7. Despliegue del Frontend: Despliegue de una interfaz web simple (implementada en Netlify) que interactúe con el API desplegado en Cloud Run para demostrar la funcionalidad.
8. Integración Continua y Despliegue continuo (CI/CD): Implementación de un pipeline que actualiza la imagen de Docker, así como el endpoint en Cloud Run.
9. Monitoreo y detección de *Data Drifting*: Implementación de un pipeline que detecta un posible *Data Drifting* en los datos.

Queda fuera del alcance de este proyecto la implementación de algún proceso que realice un reentrenamiento del modelo de forma automática, en consecuencia de la alerta emitida por el pipeline de detección de *Data Drifting* o por algún otro motivo.

2. Introducción

El presente proyecto aborda la necesidad de establecer un flujo de trabajo (pipeline) de Machine Learning (ML) robusto, reproducible y escalable que cumpla con los estándares de la industria en cuanto a las prácticas de MLOps (Machine Learning Operations).

A medida que los modelos de ML pasan de la fase experimental a un ambiente de producción, es crucial asegurar la calidad del código, la trazabilidad de los datos y experimentos, la portabilidad de los componentes, y la facilidad de despliegue y monitorización (así como el reentrenamiento de modelos).

Este proyecto se enfoca en la implementación de una arquitectura que integra las herramientas líderes del ecosistema MLOps (DVC, MLFlow, Cloud Run, Docker, etc.) para lograr los fines básicos de cualquier proyecto de Machine Learning. Se dividió en 3 fases progresivas:

2.1. Fase 1 Construcción y Experimentación

El enfoque principal de la Fase 1 fue el análisis y demostración de la viabilidad del proyecto, crear y establecer una base sólida de los datos y creación de primeros modelos predictivos. En esta fase se observa el uso de notebooks modulares.

Análisis y Datos

- **Análisis de Requerimientos:** Se documentó la problemática (predicción del consumo energético en Tetuán) utilizando el MLCanvas incluido en los anexos.
- **Manipulación y Limpieza de Datos:** Se realizó un EDA riguroso para identificar problemas como timestamps duplicados, valores faltantes y outliers extremos.
- **Preprocesamiento Avanzado:** Se implementaron transformaciones críticas, como la conversión de tipos (object, float64), corrección de outliers con IQR y mediana móvil, e Ingeniería de Características

- **Versionado de Datos:** Se utilizó en una primera etapa Drive para mantener un registro de las versiones del dataset, asegurando la trazabilidad de los datos.

Modelado y Evaluación

- **Construcción de Modelos:** Se evaluaron y compararon cinco algoritmos: *Random Forest, Gradient Boosting, XGBoost, ElasticNet,* y *SVR* .
- **Validación Rigurosa:** Se utilizó *Cross-validation* con *RepeatedKFold* (15 evaluaciones por modelo) y se empleó una División Temporal (80% entrenamiento / 20% test cronológico) para simular la predicción en un ambiente real.
- **Resultados:** Se identificó a *Random Forest* como el mejor modelo, cumpliendo los objetivos de desempeño.

2.2. Fase 2 Gestión Profesional del Proyecto

El enfoque de esta fase fue desarrollar las siguientes etapas de MLOps basándonos en lo desarrollado en la fase 1, el tener una base sólida de los datos y de los roles que se estarían desempeñando permitió que la Fase 2 se desarrollara con mayor agilidad, en esta fase se atendieron puntos importantes como:

Estructuración del código

- **Estructuración con Cookiecutter:** Se adoptó la plantilla Cookiecutter Data Science para estandarizar la organización del proyecto.
- **Refactorización OOP:** El código monolítico de los notebooks se migró y refactorizó a cuatro módulos Python modulares (CargaDatos, Preprocesamiento, Modelo, EvalModelo), aplicando Programación Orientada a Objetos (POO) y principios como Single Responsibility. Se implementaron type hints y docstrings al 100%.
- **Pipeline Scikit-Learn:** Se implementó un Pipeline integrado de Scikit-Learn (Column Transformer + modelo) para automatizar los pasos de preprocesamiento y modelado en un solo objeto serializable, previniendo automáticamente el data leakage.

Seguimiento y versionado

- **MLFlow** Se configuró MLFlow para el registro automático de características importantes del proyecto, capturando métricas, hiperparámetros y artefactos, y permitiendo la comparación visual de cada ejecución de los modelos.
- **Versionado de Datos (DVC):** Se usó DVC para versionar los datasets garantizando la reproducibilidad al asociar cada commit de Git a una versión específica de los datos.

2.3. Fase 3 Industrialización y Despliegue

La Fase 3 completó el ciclo MLOps, llevando el sistema a una fase operativa y monitoreada, implementando la infraestructura necesaria para el despliegue.

Infraestructura y Portabilidad

- **Serving con FastAPI:** Se desarrolló un servicio API REST utilizando FastAPI con un endpoint /predict para exponer el modelo para inferencia.
- **Contenerizacion:** Se creó un Dockerfile para empaquetar el servicio FastAPI, el modelo y todas las dependencias en una imagen reproducible y portable, eliminando problemas de incompatibilidad de entornos.
- **Verificacion de Reproducibilidad:** Se demostró la reproducibilidad del modelo al obtener resultados consistentes al ejecutarse en un entorno limpio/contenedor diferente al de entrenamiento.

Automatización y Calidad

- **Integración Continua (CI/CD):** Se establecieron pipelines de CI/CD (usando GitHub Actions) para automatizar la construcción de la imagen Docker, la actualización del servicio en Cloud Run, y el despliegue del frontend en Netlify.
- **Pruebas Automatizadas:** Se implementaron Pruebas Unitarias y de Integración utilizando pytest para validar componentes clave y el flujo de trabajo extremo a extremo.

Monitoreo Operacional

- **Monitoreo y Detección de Data Drift:** Se implementó una simulación de Data Drift (cambio en la distribución de datos de entrada) y se configuró la librería Evidently para detectarlo, evaluar la pérdida de rendimiento y proponer acciones
- **Despliegue Serverless:** El servicio fue desplegado en Google Cloud Run para un entorno serverless escalable y de alto rendimiento.

En la figura 1 se puede observar el ciclo MLOps, sintetizando en etapas y mostrando de forma gráfica nuestra base de desarrollo.

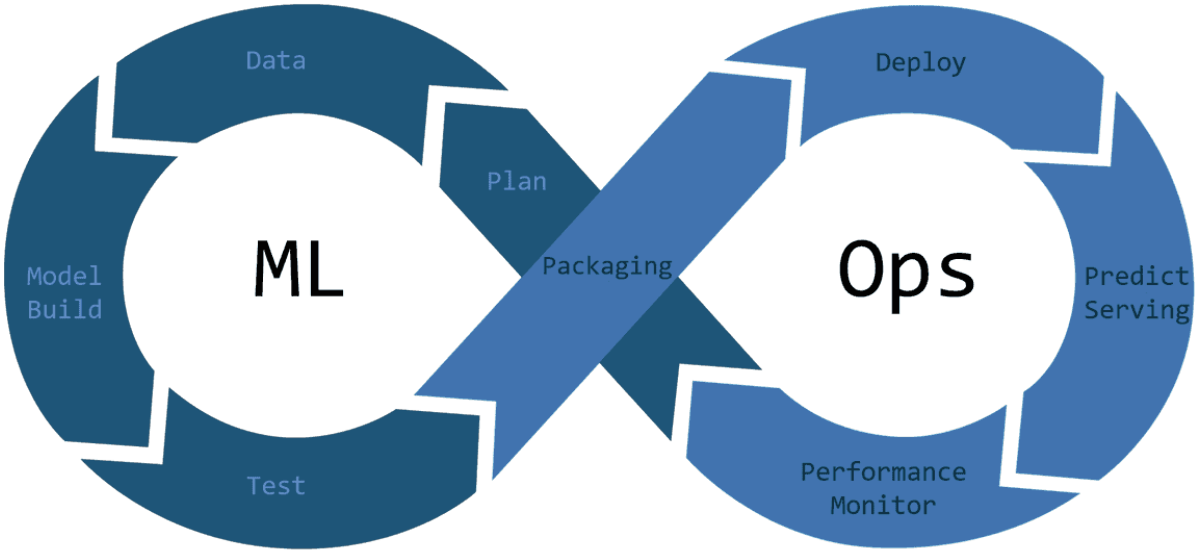

Figura 1: Ciclo MLOps.

3. Justificación Técnica

Herramienta	Rol en el Pipeline	Justificación Técnica
Cookiecutter Data Science	Estructura y Código	Seleccionado por ser el estándar de la industria. Permitió el onboarding de nuevos miembros en horas y facilitó la colaboración sin conflictos mediante la separación clara de responsabilidades.
Refactorización OOP	Código fuente	La migración de notebooks a módulos Python (con 100% de type hints y docstrings) redujo el código monolítico en 59%. Esto hizo que el código fuera reutilizable y testeable para la etapa de CI.
DVC (con Amazon S3)	Versionamiento de Datos	Proporciona Versionado real de datos para datasets grandes. Desacopla los datos pesados de Git, manteniendo el repositorio ligero (< 5MB). El uso de S3 garantiza escalabilidad ilimitada y seguridad IAM
MLFlow	Tracking y Registro	Ofrece un Registro Centralizado de runs, métricas y parámetros, eliminando el «Excel de experimentos». Su Model Registry estandariza el empaquetado y facilita el versionamiento de modelos listos para producción

Docker + Cloud Run	Portabilidad y Serving	Docker asegura la inmutabilidad y portabilidad, Cloud Run fue seleccionado como plataforma serverless para el despliegue del API, ya que ofrece escalabilidad automática a cero y optimización de costos
GitHub Actions	CI/CD	Permitió la automatización completa del proceso build -> push -> deploy. Esto aceleró el ciclo de desarrollo y garantizó que solo el código probado y contenido fuera a producción

4. Desarrollo e implementación del proyecto

A continuación se describe el proceso iterativo y estructurado que se siguió para llevar el modelo de Machine Learning (ML) desde una fase experimental hasta un servicio operativo. Se detalla la aplicación de las mejores prácticas de Programación Orientada a Objetos (POO) para garantizar la modularidad y mantenibilidad del código, así como la secuencia de pasos que integran las herramientas de MLOps.

4.1. Análisis Exploratorio de los Datos (EDA)

Propósito Principal: Exploración, limpieza y análisis exploratorio del dataset

El EDA constituyó la fase inicial y crítica para comprender la estructura general, calidad y características de los datos, así como identificar los tipos de datos de cada variable. Incluyó la visualización de distribuciones, la identificación de valores atípicos y faltantes, la correlación entre variables, detección de patrones y la generación de insights clave. Este proceso fue fundamental para guiar las decisiones de preprocesamiento, calidad de los datos, ingeniería de características, la selección del modelo más adecuado.

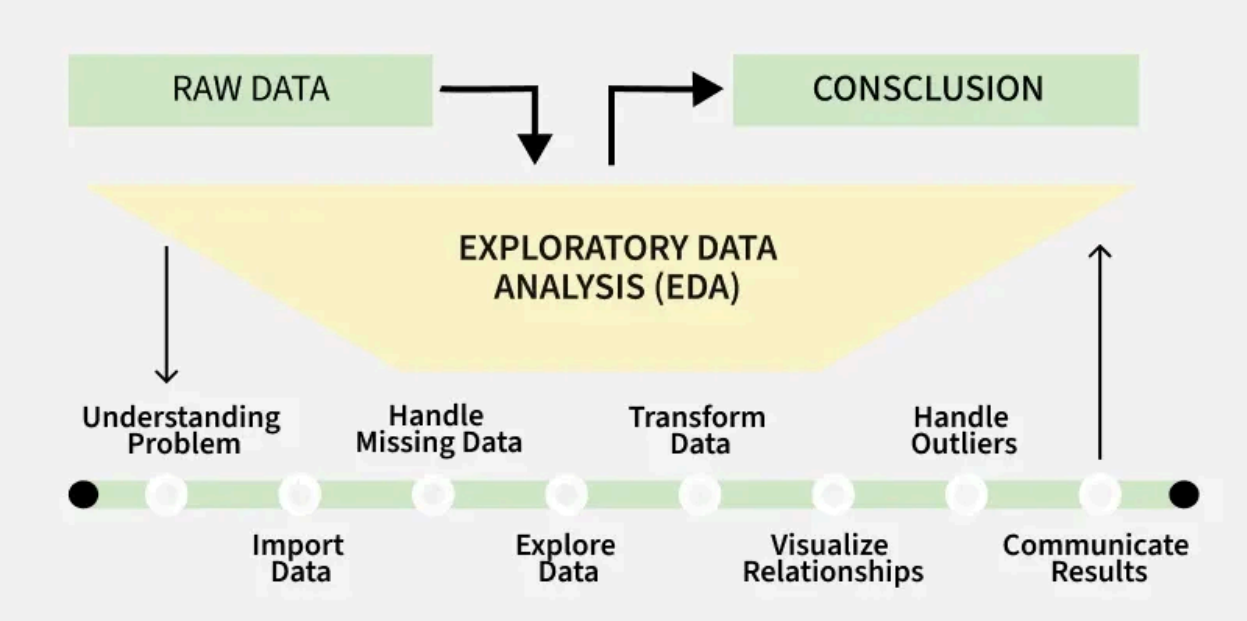

Figura 2: Proceso EDA.

La carga de datos tuvo origen de los datasets *power_tetouan_city_original* y *power_tetouan_city_modified* los cuales son Conjunto de datos público que contiene información horaria de sobre el consumo eléctrico de diferentes zonas en la ciudad de Tetuán.

Las características del dataset se detallan en la siguiente tabla:

Variable	**Tipo**	**Descripción**	**Tipo de Variable (Entrada/ Salida)**
Zone 1 Power Consumption	Numérico (kW)	Consumo de energía en Kilovatios para la Zona 1.	Salida
Zone 2 Power Consumption	Numérico (kW)	Consumo de energía en Kilovatios para la Zona 2 **(Variable Objetivo)**.	Salida
Zone 3 Power Consumption	Numérico (kW)	Consumo de energía en Kilovatios para la Zona 3.	Salida
Timestamp	Fecha/Hora	Marca de tiempo de la medición (resolución de 10 minutos).	Entrada

Temperature	Numérico (°C)	Temperatura ambiente en grados Celsius.	Entrada
Humidity	Numérico (%)	Humedad relativa en porcentaje.	Entrada
WindSpeed	Numérico (km/h)	Velocidad del viento en km/h.	Entrada
GeneralDiffuse-Flows	Numérico	Flujos difusos generales de radiación solar.	Entrada
DiffuseFlows	Numérico	Flujos difusos específicos de radiación solar.	Entrada

Para esta fase los Data Engineers asumieron la tarea crítica de Exploración y Procesamiento de Datos fueorn los encargados de transformar las columnas numéricas a tipo float64 y de abordar los datos faltantes en la columna DateTime con imputación por vecinos. Además, identificaron y optaron por eliminar una columna mixed_type_col de tipo confuso, ya que no existía en el dataset original.

Posteriormente, aplicaron una estrategia de detección y reemplazo de outliers en las variables numéricas utilizando el Rango Intercuartílico (IQR) y la mediana móvil para asegurar que los valores extremos no sesgaran el modelo.

Se utilizaron herramientas para identificación de problemas de calidad (outliers, valores duplicados, valores faltantes), visualizaciones exploratorias (histogramas, boxplots, matrices de correlación), y estadísticas descriptivas detalladas, así como herramientas y bibliotecas específicas (Python, Pandas, DVC, Scikilearn, etc.)

4.2. Estructura del proyecto con CookieCutter

Se utilizó una plantilla de CookieCutter para establecer una estructura de repositorio estandarizada y reproducible desde el inicio. Esto garantiza que todos los pipelines de desarrollo sigan una organización lógica, facilitando la colaboración, la navegación y el mantenimiento del código a largo plazo.

La estructura incluye directorios dedicados para código fuente, datasets, notebooks de experimentación, tests, API REST y frontend; esta estructura se logra ejecutando los siguientes comandos:

```
cookiecutter https://github.com/drivendata/cookiecutter-data-science
```

```
#[Configuración]
project_name: MNA_MLOPs
repo_name: MNA_MLOps
author_name:
Equipo 43
description: Predicción del Consumo de Energía en Tetuán, Marruecos
license: MIT
```

Con la plantilla correctamente configurada y adecuada a nuestro proyecto, la estructura que resulta es la siguiente:

```
MNA_MLOps/
│
├── data/
│   ├── external/              # Datos de fuentes externas
│   ├── interim/               # Datos intermedios transformados
│   ├── processed/             # Datasets finales para modelado
│   │   └── power_tetouan_city_processed.csv      (DVC tracked)
│   └── raw/                   # Datos originales inmutables
│       └── power_tetouan_city_modified.csv       (DVC tracked)
│
├── docs/                      # Documentación del proyecto
│   ├── ML_Canvas.md
│   ├── Data_Versioning.md
│   └── class_diagrams.md
│
├── models/                    # Modelos serializados (.joblib)
│
├── notebooks/                 # Jupyter notebooks experimentales
│   ├── Fase 1_Equipo43.ipynb
│   └── Fase 2_Refactorizacion.ipynb
│
├── Project/                   # Código fuente refactorizado
│   ├── __init__.py
│   ├── CargaDatos.py         # Ingesta de datos
│   ├── Preprocesamiento.py   # Pipeline de limpieza
│   ├── Modelo.py             # Entrenamiento + MLflow
│   ├── EvalModelo.py         # Evaluación comparativa
│   └── best_model_pipeline.joblib
│
├── mlruns/                    # Experimentos de MLflow
│   ├── 0/                     # Default experiment
│   ├── 781331707748066146/    # Power_Consumption_Prediction
│   └── models/                # Registered models
│       └── PowerConsumption_Zone2_Pipeline/
│           ├── version-1/
│           └── version-2/
│
├── mlartifacts/               # Artifacts de modelos MLflow
│
├── reports/                   # Reportes generados
│   ├── figures/
│   └── screenshots/           # Screenshots de MLflow UI
│
├── references/                # Diccionarios de datos
│
├── scripts/                   # Scripts de automatización
│   └── extract_mlflow_data.py
│
├── .dvc/                      # Configuración de DVC
├── .gitignore
├── Makefile                   # Automatización de tareas
├── README.md
├── requirements.txt
└── pyproject.toml
```

Figura 3: Estructura CookieCutter Data Science.

Para dar mayor explicación a los puntos anteriores:

data/: Contiene los datos originales, intermedios y procesados, organizados por etapas del pipeline.

Project/: Directorio para el código fuente modularizado (módulos Python para features, entrenamiento, etc.), resultado directo de la refactorización.

notebooks/: Para el trabajo exploratorio de los Data Scientists, separado del código de producción.

models/: Para almacenar los artefactos del modelo antes de su registro en MLFlow.

reports/: Para almacenar visualizaciones y documentación de resultados.

Adicionalmente, se incluyeron carpetas para pruebas unitarias e integrales (tests), la aplicación desplegada en Cloud Run y el Frontend desplegado en Netlify.

4.3. Versionamiento de Datos (DVC)

Se implementó Data Version Control (DVC) para gestionar el versionamiento de los datasets, para cada transformación que se le aplicaba a los datos. Esto asegura la trazabilidad y reproducibilidad de los experimentos, permitiendo a los desarrolladores moverse entre diferentes versiones de los datos y asociar cada modelo entrenado a un conjunto de datos específico.

Para esta parte, en un inicio se configuró el DVC con Google Drive; sin embargo, debido a las limitantes de que este repositorio se «limitaba» a la cuenta de algún usuario del equipo, para la fase 2 se implementó con S3. Para esto, el equipo docente nos proporcionó credenciales de AWS, así como el nombre de los buckets asignados por equipo.

Algunos de los comandos que se utilizaron para poder implementar el DVC en S3 fueron los siguientes:

- Configuración del archivo de perfil (credenciales)

```
aws configure --profile equipo43


#[Configuración]
AWS Access Key ID: KEY ID en archivo
AWS Secret Access Key: Secret Access Key en archivo
Default region name: us-east-2
Default output format: json
```

- Configuración del contenedor remoto (S3)

```
dvc remote add -d team_remote s3://itesm-mna/202502-equipo43
dvc remote modify team_remote region us-east-2
dvc remote modify team_remote profile equipo43
```

- Agregando el archivo para versionamiento

```
git add .
git commit -m "feat: Initializing DVC and setting up the remote storage in S3"
python -m dvc add data/processed/power_tetouan_city_modified.csv
```

```
git add data/processed/power_tetouan_city_modified.csv.dvc data/raw/.gitignore
git commit -m "Track power_tetouan_city_modified.csv with DVC"
python -m dvc push
```

- Comprobando que el archivo se encuentra en el bucket designado

```
aws s3 ls s3://itesm-mna/202502-equipo43 --recursive --profile equipo43 | head
```

Figura 4: Listado de archivos en AWS S3.

Beneficios del versionamiento en S3

Integrar Data Version Control (DVC) con Amazon S3 proporciona una solución robusta y escalable para gestionar conjuntos de datos de Machine Learning (ML), especialmente cuando estos son grandes.

1. Escalabilidad y Almacenamiento Rentable
   - Escalabilidad Ilimitada: Amazon S3 ofrece una capacidad de almacenamiento prácticamente ilimitada, ideal para proyectos de ML que manejan grandes volúmenes de datos.
   - Costo-Efectividad: S3 es un servicio de almacenamiento de objetos muy económico. Almacenar grandes datasets de esta manera es mucho más rentable que intentar mantenerlos replicados en varios servidores Git.
2. Desacoplamiento de Datos y Código (Git)
   - Repositorios Ligeros: DVC mantiene tu código y los metadatos de los datos (pequeños archivos .dvc que apuntan a S3) en tu repositorio GitHub. Esto mantiene a Git rápido y ligero, ya que no almacena directamente los archivos de datos pesados.
3. Trazabilidad y Reproducibilidad Completa
   - Trazabilidad Garantizada: DVC utiliza hashes (sumas de verificación) para identificar la versión exacta de un archivo de datos en S3. Se pueden versionar grandes

volúmenes de datos sin sobrecargar Git. Cada commit de Git se asocia a una versión específica de tus datos en S3 y se puede «ir y venir» de una versión de datos, de forma precisa.

- Reproducibilidad Sencilla: Cualquier miembro del equipo puede hacer checkout de un commit de Git antiguo y usar el comando dvc pull para descargar automáticamente la versión exacta de los datos asociados desde S3.

4. Seguridad y Colaboración
   - Control de Acceso (IAM): S3 se integra con AWS IAM, permitiéndote establecer políticas de acceso detalladas. Con esto, se tiene control sobre qué usuarios o roles tienen permiso para leer o escribir datos en el bucket de DVC.
   - Colaboración Global: S3 es accesible a nivel mundial. Los miembros de un equipo distribuidos geográficamente pueden acceder a los mismos conjuntos de datos versionados de forma eficiente.

4.4. Seguimiento de experimentos con MLFlow

MLFlow se utilizó como plataforma central para el seguimiento y gestión de los experimentos de Machine Learning. Se registró automáticamente el código fuente, los parámetros de entrenamiento, los hiperparámetros, y las métricas de rendimiento (como RMSE o r2 score). Esto permitió una comparación objetiva y eficiente de los resultados de múltiples ejecuciones, facilitando la selección del mejor modelo.

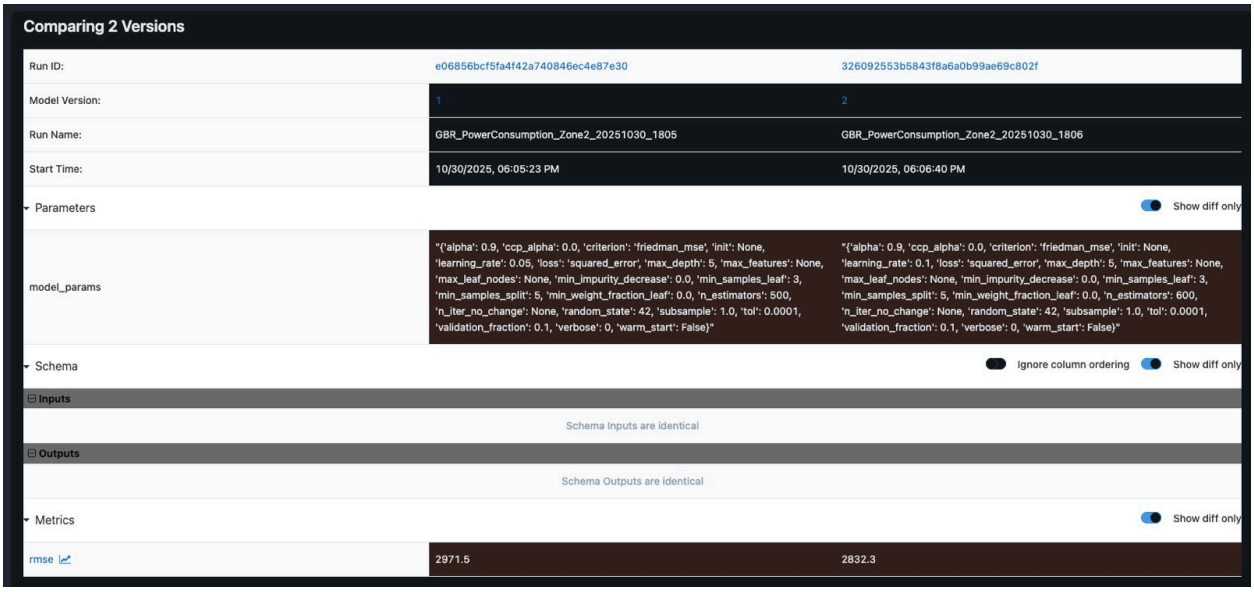

Figura 5: Comparación de versiones en MLFlow.

Para la última fase, se desplegó el mismo servicio de MLFlow en una herramienta web (DagsHub), que permitió que todos los integrantes del equipo tuvieran acceso a los experimentos, siguiendo el principio de accesibilidad y colaboración.

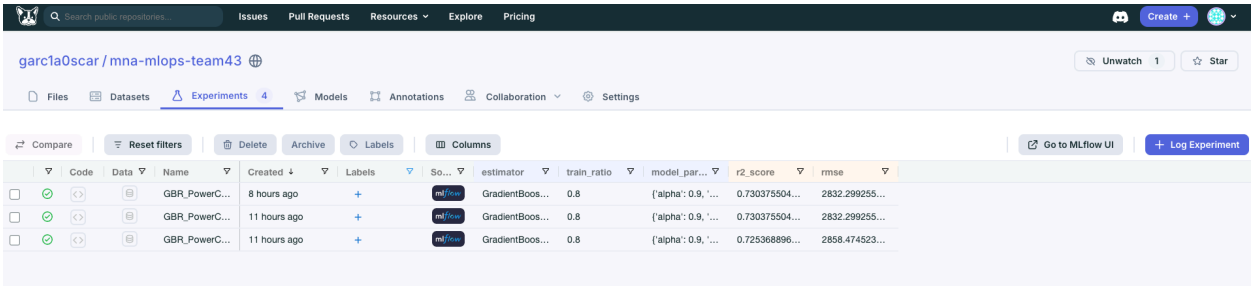

Figura 6: Comparación de versiones en MLFlow.

Beneficios de Utilizar MLflow en Proyectos ML

MLflow es una plataforma de código abierto diseñada para gestionar el ciclo de vida completo del Machine Learning, incluyendo el seguimiento de experimentos, la reproducibilidad y el despliegue de modelos.

1. Trazabilidad y Gestión Centralizada de Experimentos
   - MLflow Tracking: Proporciona un **registro centralizado** de todas las métricas, parámetros, artefactos y códigos fuente utilizados en cada ejecución experimental (Run). Esto elimina la confusión sobre qué **notebook** o script generó un resultado específico.
   - Reproducibilidad Sencilla: Se puede **comparar y analizar** diferentes «Runs» lado a lado, facilitando la identificación de las mejores combinaciones de hiperparámetros y modelos.
2. Estandarización de Modelos
   - MLflow Models: Ofrece un **formato estándar** para empaquetar modelos de ML (Python, R, Java, etc.). Esto permite que cualquier modelo se pueda desplegar en una variedad de plataformas como **Docker**, **Azure ML**, **AWS SageMaker** o **Kubernetes** sin reescribir código.
   - Interfaz Uniforme: El formato estándar asegura que la forma de cargar y hacer predicciones con un modelo sea la misma, independientemente del **framework** original (TensorFlow, PyTorch, Scikit-learn, etc.).
3. Gestión del Ciclo de Vida y Despliegue (MLOps)
   - MLflow Model Registry: Actúa como un **repositorio centralizado** para gestionar la transición de modelos a través de diferentes etapas. Esto proporciona una única fuente de verdad sobre el estado del modelo.
   - Versionamiento y Rollbacks: Facilita el **versionamiento** de los modelos que están listos para producción y simplifica las **operaciones de rollback** (revertir a una versión anterior) si un modelo desplegado presenta problemas.

4.5. Contenerización de aplicaciones con Docker

La contenerización del modelo y su API de inferencia se llevó a cabo utilizando Docker. Este proceso empaquetó el código, las dependencias y el entorno de ejecución en una imagen portable y aislada. Esto eliminó los problemas de incompatibilidad de entornos, asegurando que el servicio se comporte de manera idéntica en el desarrollo local y en el entorno de cloud de producción.

Dentro del proyecto, construimos nuestro empaquetado, a partir de un archivo *Dockerfile* que aplica los siguientes pasos:

- Instalación de la imagen de sistema operativo a usar (python:3.10-slim).
- Definir el directorio de trabajo (WORKDIR)
- Copiar los contenidos de la carpeta a la imagen creada.
- Instalar las dependencias, a partir del archivo requirements.txt
- Inicializar la aplicación con uvicorn.

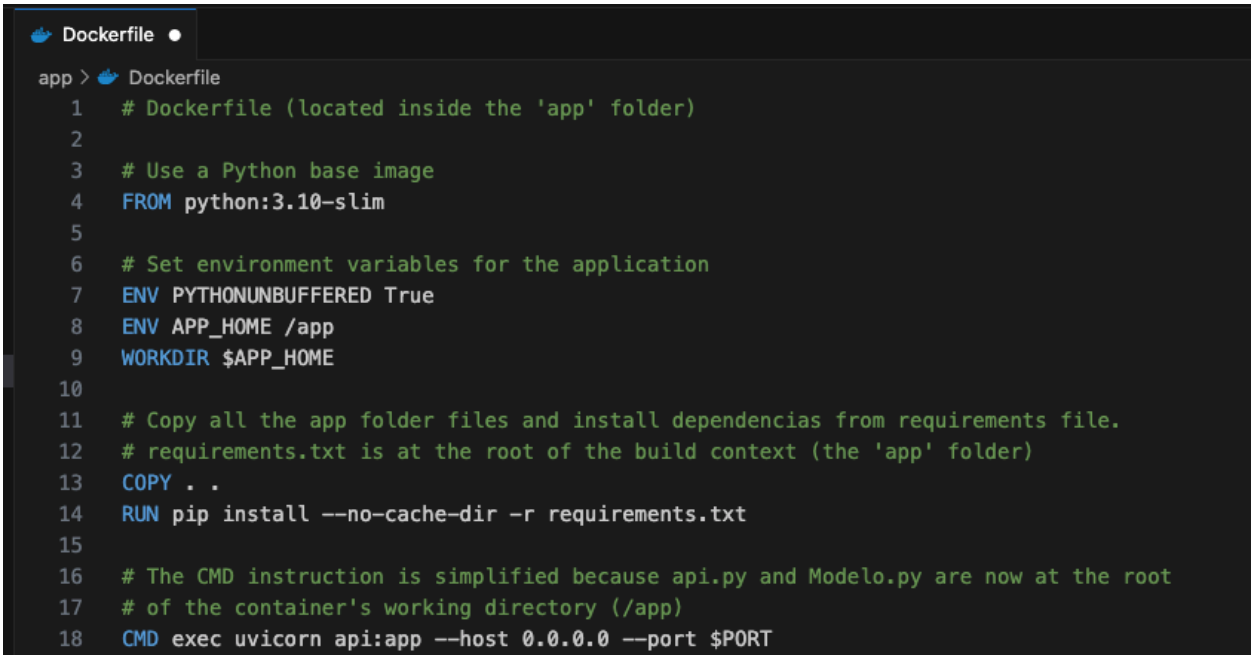

Figura 7: Estructura de archivo Dockerfile.

Una vez creado nuestro archivo Dockerfile, podemos ejecutar los comandos correspondientes para poder desplegar nuestro contenedor en un servicio como Artifact Registry, dentro

de la nube de Google Cloud Platform, previamente configuradas nuestras credenciales para acceder al proyecto de GCP por medio de nuestra terminal (GCLOUD CLI)

Nota: Cabe mencionar que, aunque este comando lo podemos ejecutar de forma manual, en el proyecto lo integramos dentro de nuestro pipeline de CI/CD para hacer este proceso de forma automática cada que se realice un cambio en alguno de los componentes de la aplicación (código, librerías, lógica, versión de imagen, etc.)

Beneficios de la Contenerización con Docker

Docker permite empaquetar una aplicación y todas sus dependencias (bibliotecas, configuraciones, scripts, etc.) en una unidad estandarizada llamada contenedor. Esto garantiza que la aplicación se ejecute de manera rápida y fiable en cualquier entorno de computación.

1. Estandarización y Reproducibilidad
   - Elimina el «funciona en mi máquina»: Docker resuelve el problema de la diferencia de entornos al garantizar que el **entorno de ejecución** sea siempre el mismo, desde el desarrollo hasta la producción.
   - Inmutabilidad: Una vez que se construye una imagen de Docker, esta es inmutable. Si necesitas un cambio, construyes una **nueva imagen**, lo que mejora la consistencia y la trazabilidad.
2. Aislamiento y Seguridad
   - Aislamiento de Procesos: Los contenedores están **aislados** del sistema operativo anfitrión y entre sí. Esto significa que si falla una aplicación o componente, no afectará a otros contenedores o al **host**.
   - Dependencias Limpias: Cada contenedor tiene sus propias dependencias. Esto evita conflictos de versiones de bibliotecas y mantiene el sistema operativo base del servidor **limpio y ligero**.
3. Eficiencia y Portabilidad

- Rápida Inicialización: Los contenedores son mucho más **ligeros y rápidos** que las máquinas virtuales (VMs). Pueden iniciarse en segundos, lo que acelera el ciclo de desarrollo y despliegue continuo (CI/CD).
- Portabilidad en la Nube: Un contenedor de Docker se ejecuta sin modificaciones en cualquier plataforma que soporte contenedores (GCP, AWS, Azure, Kubernetes, etc.), facilitando la **migración** y el despliegue multi-nube.

4. Optimización de Recursos
   - Densidad: Permite ejecutar **múltiples contenedores** en un único servidor host, utilizando los recursos del sistema operativo de manera más eficiente que las máquinas virtuales.

4.6. Disponibilización del servicio

El servicio de inferencia, encapsulado en un contenedor Docker, se desplegó como un API REST serverless utilizando Google Cloud Run. Esta estrategia permite la escalabilidad automática basada en la demanda, minimiza los costos operativos y facilita la gestión de la infraestructura, haciendo que el modelo esté disponible bajo demanda para aplicaciones externas (en nuestro caso, nuestro frontend).

Tal y como se menciono en la sección anterior, una vez que tenemos configurado correctamente nuestro archivo Dockerfile, podemos realizar un despliegue (registro: push) hacia nuestro repositorio en Artifact Registry y, posteriormente, hacia Cloud Run.

- Comandos para registrar el contenedor de Docker en Artifact Registry

```
docker build -t <REGION>-docker.pkg.dev/<PROJECT-ID>/<REPOSITORY-NAME>/<IMAGE-NAME>:<TAG> .
docker push <REGION>-docker.pkg.dev/<PROJECT-ID>/<REPOSITORY-NAME>/<IMAGE-NAME>:<TAG>
```

- Comandos para crear nuestra aplicación en Cloud Run (GCP CLI)

```
gcloud run deploy $SERVICE_NAME --image <REGION>-docker.pkg.dev/<PROJECT-ID>/<REPOSITORY-NAME>/<IMAGE-NAME>:<TAG>
--region $REGION --allow-unauthenticated --platform managed --min-instances 0
```

Nota: nuevamente, aunque estos comandos permiten realizar el despliegue de forma manual, dentro del proyecto se implementaron en Github Actions, como parte del CI/CD.

Beneficios de Desplegar Servicios en Google Cloud Run

Google Cloud Run es una plataforma de cómputo serverless que te permite ejecutar contenedores directamente, sin preocuparse por la complejidad de la infraestructura subyacente. Es ideal para microservicios y APIs.

1. Escalabilidad Automática y Serverless

- Escalado a Cero: La principal ventaja es la capacidad de **escalar hasta cero instancias** cuando no hay tráfico.
- Escalado Rápido: La plataforma gestiona automáticamente el escalado de uno a n contenedores de forma rápida y eficiente para manejar picos de tráfico.

2. Simplicidad y Foco en el Código
   - Menos DevOps: Al ser una plataforma **serverless**, Cloud Run **elimina la necesidad de administrar clústeres** de Kubernetes, máquinas virtuales, o parches. Los equipos pueden concentrarse únicamente en el desarrollo de la aplicación.
   - Despliegue de Contenedores: Acepta cualquier contenedor de Docker estándar. Esto permite a los desarrolladores usar cualquier lenguaje de programación, librería o dependencia sin restricciones de **runtime**.
3. Optimización de Costos
   - Modelo de Pago por Uso: Gracias al escalado a cero y la facturación por milisegundos de uso, Cloud Run ofrece un modelo de costos **muy predecible y optimizado**, siendo significativamente más barato que mantener VMs 24/7.
4. Integración con Google Cloud
   - Entorno Gestionado: Se integra nativamente con otros servicios de Google Cloud como **Artifact Registry** (para las imágenes), **Cloud Logging** y **Cloud Monitoring**, simplificando la observabilidad y el ciclo de MLOps/DevOps.
   - Dominios Personalizados: Permite asignar dominios personalizados y gestionar certificados SSL/TLS automáticamente.

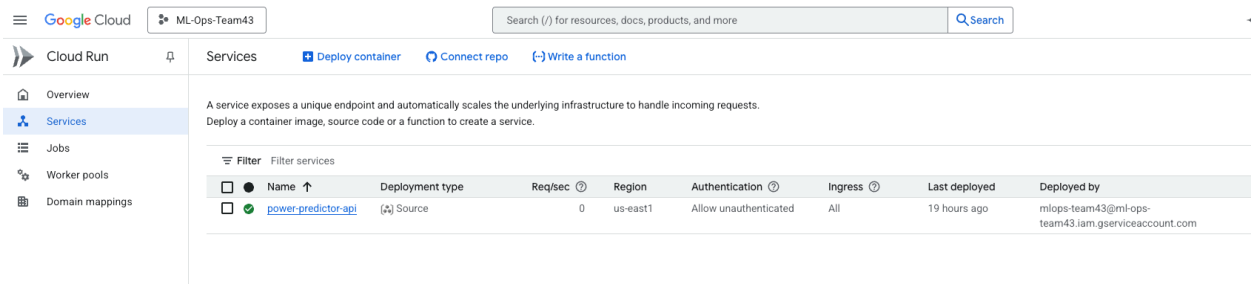

Figura 8: Aplicación desplegada en Cloud Run (GCP).

Después de haber desplegado nuestra aplicación en Cloud Run, se hizo una prueba con un método POST, desde la aplicación Postman, que se muestra a continuación. Los datos son ficticios y están destinados únicamente para la prueba realizada.

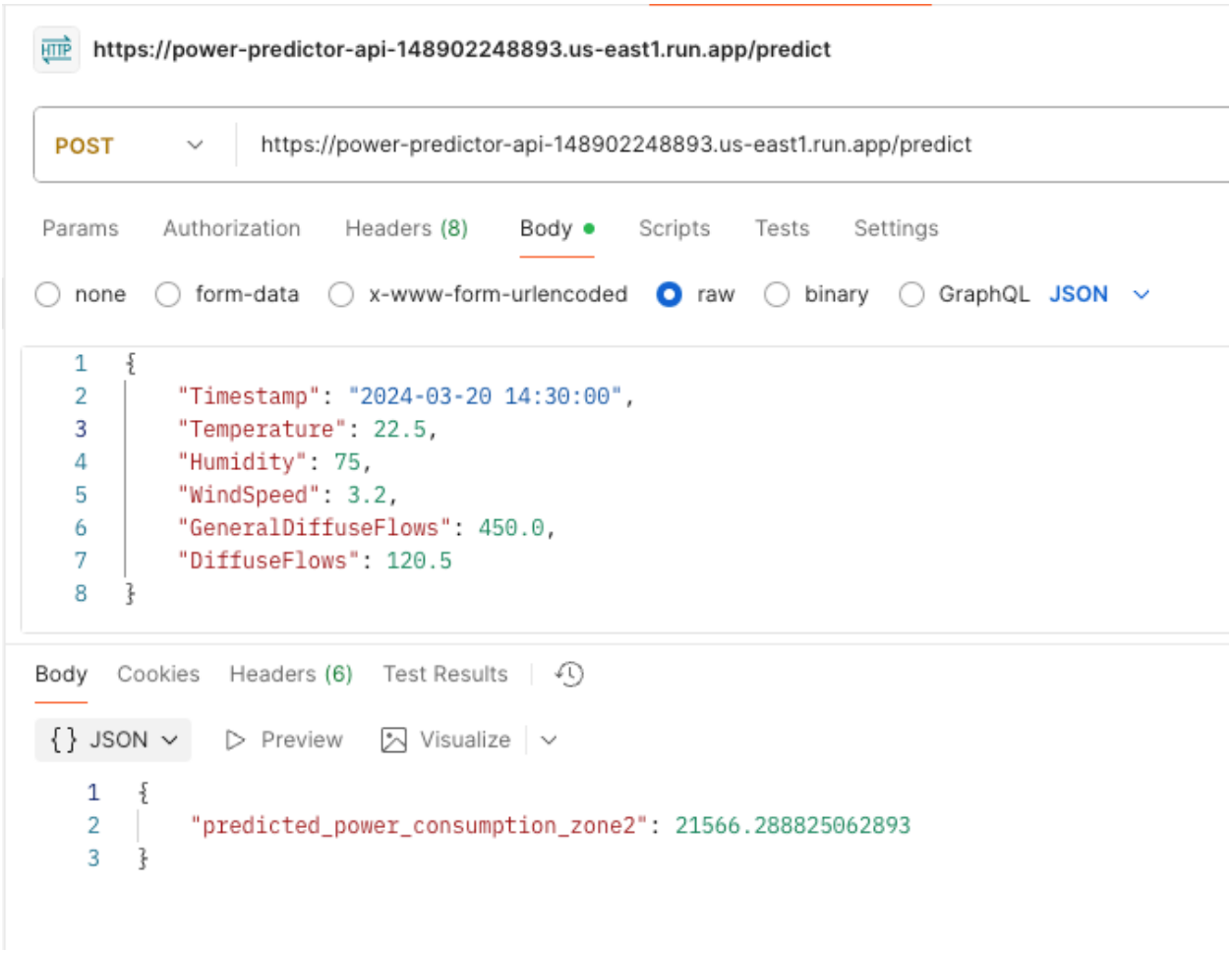

Figura 9: Prueba del API Endpoint con Postman.

4.7. Despliegue del Frontend

Se desarrolló un interfaz de usuario simple (frontend) que se desplegó en el servicio Netlify. El objetivo de este componente es demostrar la funcionalidad del modelo en un entorno de usuario final. Este frontend consume el API de inferencia desplegado en Cloud Run, cerrando el ciclo de vida del desarrollo y facilitando las pruebas de interacción.

Beneficios del Despliegue de Frontend (React en Netlify)

Netlify es una plataforma de hosting moderna y de alto rendimiento que se especializa en la Arquitectura JAMstack, lo que la hace ideal para aplicaciones de frontend construidas con frameworks modernos como React.

1. Flujo de Trabajo y CI/CD Integrado
   - Despliegue Continuo (CI/CD): Ofrece integración directa con repositorios Git (GitHub, GitLab, Bitbucket). Cada vez que haces un **push** a la rama principal, Netlify es capaz de **automáticamente construir** nuestra aplicación React y desplegarla.
   - Despliegues Instantáneos: Los despliegues son rápidos y se propagan globalmente a través de su Red de Distribución de Contenido (CDN) casi al instante.
2. Optimización de Rendimiento
   - Red de Distribución de Contenido (CDN): El sitio web se distribuye globalmente, lo que significa que los usuarios siempre obtienen contenido desde el **servidor más cercano**, resultando en **tiempos de carga ultra rápidos**.
   - Auto-Configuración: Netlify optimiza automáticamente los activos de tu aplicación React (compresión, minificación, **hashing** de archivos) sin configuración manual.
3. Facilidad de Uso y Colaboración
   - Previsualizaciones de Despliegue: Para cada **Pull Request** (PR), Netlify genera una **URL de previsualización única**. Esto permite al equipo revisar la funcionalidad y el **look and feel** de los cambios antes de fusionarlos a la rama principal.
   - Configuración Sencilla de Dominios: Conexión simple de dominios personalizados y gestión gratuita y automática de **certificados SSL/TLS** (HTTPS) con Let's Encrypt.

4. Despliegues Atómicos
    - Cero Tiempo de Inactividad (**Downtime**): Netlify garantiza que tu sitio está completamente desplegado y funcional **antes** de cambiar el tráfico a la nueva versión. Si el despliegue falla, la versión anterior permanece activa.
5. Experiencia de Usuario y Separación de Responsabilidades
    - Interfaz Intuitiva: El frontend, construido con React, permite crear una **interfaz de usuario rica, dinámica e intuitiva**. Esto mejora significativamente la satisfacción del usuario y reduce la curva de aprendizaje.
    - Separación Lógica: Al separar el frontend (presentación y lógica de UI) del backend (lógica de negocio y datos), se logra un **desacoplamiento**. Esto permite que los equipos trabajen de forma independiente y que cada capa pueda evolucionar o escalar sin afectar a la otra.

4.8. Integración Continua y Despliegue Continuo (CI/CD)

Se establecieron pipelines de CI/CD básicos para automatizar la construcción (CI) y el despliegue (CD) de los artefactos principales. Esto incluye laa creación/actualización de la imagen Docker del servicio, la actualización del servicio en Cloud Run y el despliegue del frontend en Netlify tras cada cambio significativo en el código, garantizando entregas rápidas y consistentes.

Beneficios de la Integración y Despliegue Continuo (CI/CD)

Implementar un pipeline CI/CD automatiza los pasos de construir, probar y desplegar tu código. Esta práctica acelera el ciclo de desarrollo, reduce errores humanos y asegura que el software de alta calidad esté siempre listo para el usuario final.

1. Detección Temprana de Errores (Integración Continua - CI)
   - Feedback Rápido: Cada vez que un desarrollador integra código (**push** a Git), el sistema ejecuta pruebas automatizadas. Esto proporciona **feedback inmediato** sobre si el cambio rompió algo (pruebas unitarias, integración), haciendo que la corrección sea rápida y menos costosa.
   - Reducción de Riesgos: Al integrar cambios pequeños y frecuentes, se evita el riesgo de introducir grandes errores al final de un ciclo de desarrollo extenso.
2. Automatización y Eficiencia (Despliegue Continuo - CD)
   - Despliegues Frecuentes y Atómicos: La capacidad de desplegar de forma **automática y frecuente** permite a los equipos entregar valor a los usuarios rápidamente. Los despliegues son «atómicos» (o funcionan o se revierte), minimizando el tiempo de inactividad (**downtime**).
   - Eliminación de Tareas Manuales: Se eliminan las tareas manuales y repetitivas de **build** y despliegue, **liberando tiempo** valioso de los desarrolladores para centrarse en la creación de nuevas funcionalidades.
3. Mejora de la Calidad y Estandarización

- Calidad Constante: La ejecución automática de **linters** y verificaciones de seguridad en cada **build** garantiza un **estándar de calidad de código** consistente a lo largo del tiempo.
- Trazabilidad Completa: Los **pipelines** de CI/CD, especialmente con herramientas como GitHub Actions, dejan un registro de **quién, qué y cuándo** se desplegó un cambio específico, mejorando la auditoría y la trazabilidad.

deploy_app.yml

.github > workflows > deploy_app.yml

```yaml
name: CI/CD Cloud Run con Artifact Registry

on:
  push:
    branches:
      - main
      - oscar
    paths:
      - 'app/**'

jobs:
  deploy:
    runs-on: ubuntu-latest
    environment: production

    # Permisos requeridos para la autenticación OIDC
    permissions:
      contents: 'read'
      id-token: 'write'

    steps:
      - name: Checkout del Repositorio
        uses: actions/checkout@v4

      # 1. Autenticación en Google Cloud (Usando WIF)
      - name: Autenticación en Google Cloud
        id: auth
        uses: google-github-actions/auth@v2
        with:
          # Usa la Variable de Repositorio para la ruta del proveedor WIF
          workload_identity_provider: ${{ vars.GCP_WIF_PROVIDER }}
          # Usa el Secreto de Repositorio para la cuenta de servicio
          service_account: ${{ secrets.GCP_SA_EMAIL }}

      # 2. Configurar gcloud CLI
      - name: Configurar gcloud CLI
        uses: google-github-actions/setup-gcloud@v2
        with:
          project_id: ${{ vars.GCP_PROJECT_ID }}
          export_default_credentials: true

      # 3. CONECTAR DOCKER A ARTIFACT REGISTRY
      - name: Configurar Docker para Artifact Registry
        run: gcloud auth configure-docker ${{ vars.GCP_REGION }}-docker.pkg.dev --quiet
```

Figura 10: Worfklow CI/CD para aplicación en Cloud Run Pt.1

```yaml
# 3. CONECTAR DOCKER A ARTIFACT REGISTRY
- name: Configurar Docker para Artifact Registry
  run: gcloud auth configure-docker ${{ vars.GCP_REGION }}-docker.pkg.dev --quiet

# 4. Construir la Imagen de Docker
- name: Construir Imagen
  run: |
    IMAGE_TAG=${{ vars.GCP_REGION }}-docker.pkg.dev/${{ vars.GCP_PROJECT_ID }}/${{ vars.GCP_REPOSITORY_NAME }}/${{ vars.GCP_SERVICE_NAME }}:${{ github.sha }}
    docker build -t $IMAGE_TAG ./app

# 5. Enviar la Imagen a Artifact Registry
- name: Enviar a Artifact Registry
  run: |
    IMAGE_TAG=${{ vars.GCP_REGION }}-docker.pkg.dev/${{ vars.GCP_PROJECT_ID }}/${{ vars.GCP_REPOSITORY_NAME }}/${{ vars.GCP_SERVICE_NAME }}:${{ github.sha }}
    docker push $IMAGE_TAG

# 6. Desplegar la Imagen en Cloud Run
- name: Desplegar a Cloud Run
  uses: google-github-actions/deploy-cloudrun@v2
  with:
    service: ${{ vars.GCP_SERVICE_NAME }}
    region: ${{ vars.GCP_REGION }}
    image: ${{ vars.GCP_REGION }}-docker.pkg.dev/${{ vars.GCP_PROJECT_ID }}/${{ vars.GCP_REPOSITORY_NAME }}/${{ vars.GCP_SERVICE_NAME }}:${{ github.sha }}
```

Figura 11: Worfklow CI/CD para aplicación en Cloud Run Pt.2

deploy_frontend.yml M

.github > workflows > deploy_frontend.yml

```yaml

on:
  push:
    branches:
      - main
      - oscar
    paths:
      - 'frontend/**'

jobs:
  deploy:
    runs-on: ubuntu-latest

    environment: production
    steps:
      - name: Checkout del Código
        uses: actions/checkout@v4

      - name: Configurar Node.js 20
        uses: actions/setup-node@v4
        with:
          node-version: 20

      - name: Instalar Dependencias y Construir la Aplicación (Build)
        run: |
          npm install
          npm run build
        working-directory: ./frontend/prediction-frontend

      - name: Desplegar a Netlify
        uses: nwtgck/actions-netlify@v2
        env:
          NETLIFY_AUTH_TOKEN: ${{ secrets.NETLIFY_AUTH_TOKEN }}
          NETLIFY_SITE_ID: ${{ secrets.NETLIFY_SITE_ID }}
        with:
          # Carpeta donde Vite/React generó los archivos finales
          publish-dir: './frontend/prediction-frontend/dist'

          # La branch 'main' será la producción (puedes cambiar esto si 'oscar' es la de producción)
          production-branch: main

          # Crea una URL de previsualización para cualquier branch que no sea 'main'
          deploy-preview: true
```

Figura 12: Worfklow CI/CD para frontend

4.9. Monitoreo y Detección de Data Drift

Esta sección describe la implementación de herramientas y métricas para el monitoreo continuo del modelo en producción. Un enfoque clave es la detección de Data Drift, que ocurre cuando las estadísticas de los datos de entrada en producción divergen significativamente de las usadas durante el entrenamiento. Esto permite alertar a tiempo sobre posibles degradaciones en el rendimiento del modelo y accionar posibles soluciones como lo es un reentrenamiento del modelo y mantenerlo vigente.

Beneficios del Monitoreo y Detección de Data Drift

El monitoreo continuo es esencial para mantener la precisión y el rendimiento de un modelo de Machine Learning en producción. La detección de Data Drift asegura que la distribución de los datos de entrada no haya cambiado significativamente con el tiempo.

1. Mantenimiento de la Precisión del Modelo
   - Detección de *Drift*: Permite identificar cuándo la **distribución de los datos de producción** (los datos que el modelo ve ahora) se ha separado significativamente de la **distribución de los datos de entrenamiento**. Esta separación es la causa principal de la caída del rendimiento del modelo (**model decay**).
   - Alertas Proactivas: Al detectar el *drift* de datos de entrada o el *drift* de concepto (cuando la relación entre las **features** y el **target** cambia), el equipo puede recibir **alertas automáticas** para reentrenar o actualizar el modelo antes de que el rendimiento caiga a niveles inaceptables.
2. Trazabilidad y Análisis de Causa Raíz
   - Análisis de Features: Las herramientas de monitoreo permiten ver qué **features** específicas han experimentado el cambio de distribución más drástico (por ejemplo, si una unidad de medida cambió, o si un sensor se dañó), facilitando el **diagnóstico rápido** de la causa raíz.

- Métricas de Negocio: Además de las métricas técnicas (precisión, AUC), se pueden monitorear **métricas de negocio** (ingresos, tasas de clics) para correlacionar el *drift* del modelo con el impacto real en el negocio.

3. Optimización del Ciclo MLOps
   - Reentrenamiento Inteligente: En lugar de reentrenar el modelo en un horario fijo (ej. cada mes), el monitoreo permite adoptar un enfoque basado en eventos (**event--based**). El modelo solo se reentrena cuando el drift **supera un umbral** definido, optimizando los recursos de cómputo.

Detección de Data Drift

Implementación con Evidently.ai

Para este proyecto se implementó un sistema robusto de monitoreo y detección de data drift utilizando la librería Evidently.ai versión 0.7.14. Esta herramienta de código abierto permite detectar cambios en las distribuciones de datos y evaluar el rendimiento del modelo de manera automatizada.

Script principal desarrollado: `scripts/monitor_data_drift_evidently.py`

Componentes clave del sistema de monitoreo:

1. **Detección de Drift Estadístico**
   - Tests de Kolmogorov-Smirnov para features numéricos
   - Tests de Chi-cuadrado para features categóricos
   - Umbral de p-value < 0.05 para indicar drift significativo

2. **Evaluación de Performance del Modelo**
   - Comparación baseline vs. datos con drift
   - Métricas monitoreadas: RMSE, MAE, R^2, MAPE
   - Umbrales de degradación definidos:
     - RMSE: >10% degradación
     - MAE: >10% degradación

- R^2: >5% degradación

3. **Reportes Generados**

Archivo	Descripción	Uso
data_drift_report.html	Reporte visual interactivo con distribuciones y tests estadísticos	Revisión manual por Data Scientists
data_drift_report.json	Métricas en formato JSON legible por máquina	Integración con sistemas de alertas
comparison.json	Comparación baseline vs drift con degradación porcentual	Decisiones de reentrenamiento

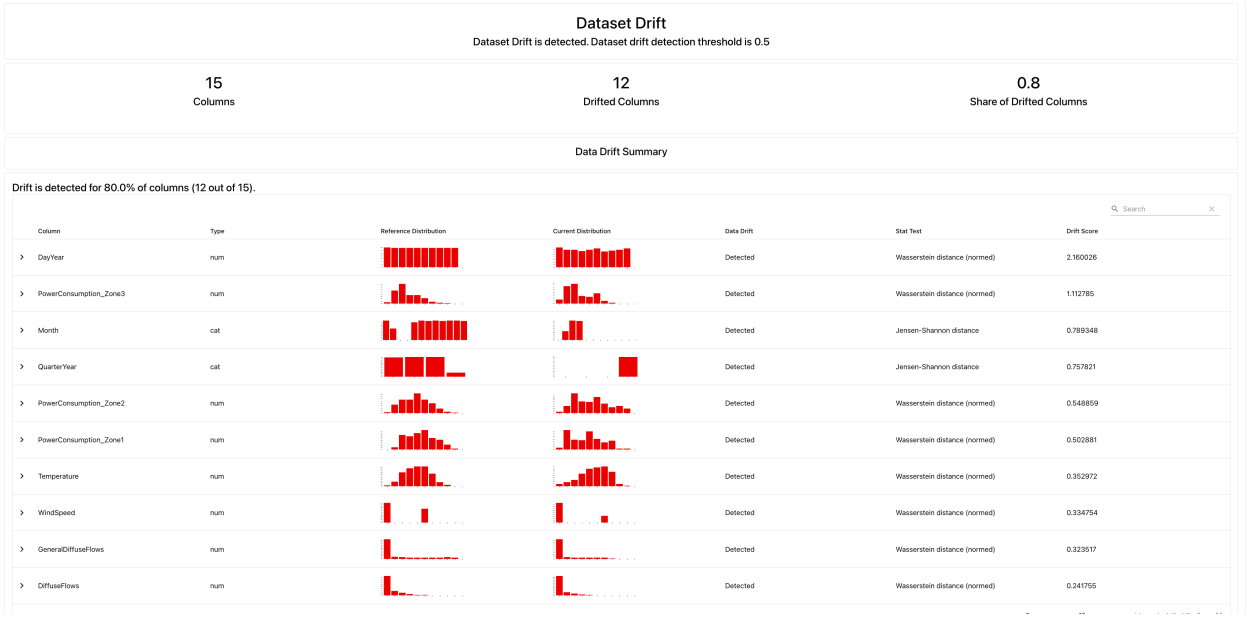

Figura 13: Dashboard de Evidently.ai mostrando detección de drift en 12 de 15 columnas

Resultados de la implementación:

En las pruebas realizadas con drift simulado (incremento de +5°C en temperatura), el sistema detectó correctamente:

- 12 de 15 columnas (80%) con drift estadístico
- Degradación del modelo: −0.37% en RMSE (dentro de umbrales aceptables)
- No se excedieron los umbrales críticos de performance

Comando de ejecución:

```
source venv/bin/activate && python
  scripts/monitor_data_drift_evidently.py
```

Integración en el pipeline MLOps:

El script de monitoreo se integró en el pipeline completo (`run_mlops_complete.py`) permitiendo:

- Ejecución automatizada post-entrenamiento
- Generación de alertas cuando se exceden umbrales
- Documentación de tendencias a lo largo del tiempo

Esta implementación asegura que el equipo pueda detectar proactivamente degradación del modelo antes de que impacte al usuario final, permitiendo tomar decisiones informadas sobre cuándo reentrenar.

5. Roles y participación del equipo

5.1. Metodología de Trabajo

El equipo adoptó una metodología ágil con roles especializados siguiendo principios de MLOps. Cada miembro asumió un rol específico con responsabilidades claras, permitiendo trabajo paralelo y minimizando dependencias.

Roles definidos:

1. Data Engineer (DE): Esteban Guerra, Alberto Campos
2. ML Engineer/Data Scientist (ML/DS): Esteban Guerra, Oscar García
3. MLOps Engineer (MLOpsE): Oscar García , Jessica García
4. Software Engineer (SWE): Alberto Campos, Jessica García
5. DevOps / Project Manager (PM): Rafael Sánchez

5.2. Responsabilidades y tareas por rol

5.2.1. Data Engineer

Responsabilidades:

- Diseño e implementación de pipelines de datos.
- Limpieza y transformación de datasets.
- Versionado de datos con DVC.
- Validación de la calidad de los datos.
- Implementación de herramientas para monitorear data quality y colaborar en la detección de data drift por medio de la librería evidently.

Código desarrollado:

```
Project/
├── CargaDatos.py (31 líneas)
└── Preprocesamiento.py (164 líneas)
```

Funciones clave implementadas:

1. **_limpiar_parsear_datetime()**
   - Manejo de múltiples formatos de fecha
   - Imputación inteligente de gaps temporales
   - ~50 líneas de código
2. **_outliers_meadiana_rodante()**
   - Detección con IQR (Q1-1.5IQR, Q3+1.5IQR)
   - Corrección con mediana rodante (ventana=25)
   - Preservación de tendencias temporales
3. **_features_tiempo()**
   - Extracción de Day, Month, Hour, Minute
   - Cálculo de Day of Week, Quarter, Day of Year
   - ~30 líneas

Herramientas utilizadas:

- Pandas
- Numpy
- Python dataclasses
- DVC
- Jupyter notebooks

5.2.2. ML Engineer/Data Scientist

Responsabilidades:

- Implementación de modelos de machine learning
- Experimentación con hiperparámetros
- Integración con MLflow para tracking
- Evaluación comparativa de algoritmos
- Diseñar los métodos de detección de data drift a nivel estadístico o por performance del modelo

Código desarrollado:

```
Project/
├── Modelo.py             (148 líneas)
└── EvalModelo.py         (145 líneas)
```

Clases implementadas:

1. **ModeloEspecial**
   - Entrenamiento con tracking MLflow automático
   - Serialización de pipelines Scikit-Learn
   - Model registry integration
2. **Evaluador**
   - Comparación de 5 algoritmos (RF, XGB, GB, ElasticNet, SVR)
   - Cross-validation con RepeatedKFold
   - Selección automática del mejor modelo

Herramientas utilizadas:

- Scikit-Learn
- XGBoost
- MLflow
- Joblib
- NumPy
- Pandas

5.2.3. MLOps Engineer

Responsabilidades:

- Configuración de infraestructura MLflow
- Setup de DVC con Google Drive y luego con Amazon S3
- Documentación de workflows MLOps
- Estandarización de logging

- Liderar la dockerización del modelo y de los servicios asociados (API, workers, pipelines).
- Preparar pipelines de CI/CD, incluyendo ejecución automática de pytest.
- Integrar el monitoreo de data drift, performance del modelo y logs operativos.

Configuraciones implementadas:

1. **MLflow:**

mlflow.set_tracking_uri(«http://127.0.0.1:5000»)
mlflow.set_experiment(«Power_Consumption_Prediction»)

2. **DVC:**

Integración en Amazon S3

3. **Google Cloud Platform:**
   - Proyecto: mlops-equipo43
   - API: Google Drive API habilitada
   - OAuth: Client ID configurado para DVC
4. **Docker:**
   - Dockerfile
5. **Monitoreo:**
   - `monitor_data_drift_evidently.py`
6. **Documentación creada:**
   - docs/Data_Versioning.md
   - Guía completa de DVC
   - docs/MLflow_Setup.md
   - Instrucciones de configuración
   - Troubleshooting en README.md

Herramientas:

- MLflow

- DVC
- Google Cloud Platform
- Amazon S3
- Bash scripting
- Docker

5.2.4. Software Engineer

Responsabilidades:

- Refactorización de código notebook → módulos
- Aplicación de principios SOLID
- Code review - Mejora de calidad (type hints, docstrings)
- Implementar el frontend (web o interfaz de usuario), conectándolo con la API del modelo.

Refactorizaciones realizadas:

Métrica	Antes	Después	Mejora
Type hints	0%	100%	+100%
Docstrings	20%	100%	+400%
Funciones reutilizables	3	22	+633%
Complejidad ciclomática	15	6	−60%

Principios aplicados:

- Single Responsibility (un módulo, una responsabilidad)
- Dataclasses para reducir boilerplate
- Type hints para documentación - Métodos privados vs públicos

Herramientas:

- VSCode
- Pylance
- Ruff (linter)
- Git

- Netlify

5.2.5. DevOps / Project Manager

Responsabilidades:

- Estructuración con Cookiecutter
- Gestión de repositorio Git
- Automatización con Makefile
- Coordinación del equipo
- Supervisar que la ejecución de CI/CD, monitoreo y logging esté correctamente integrada.

Implementaciones:

Cookiecutter:

```
cookiecutter https://github.com/drivendata/cookiecutter-data-science
```

Makefile:

- install: `pip install -r requirements.txt`
- mlflow-ui: `mlflow ui --port 5000`
- dvc-pull: `dvc pull`
- pipeline completo: `python Project/run_mlops_complete.py`

Convenciones Git:

- Branch naming: feature/nombre-descriptivo
- Commits: feat:, fix:, docs:, refactor:
- PR review: Mínimo 1 aprobación

6. Anexos

6.1. Anexo A: Ejecución de flujo completo

Para este proyecto, también se creó un *script* que hace una ejecución completa del flujo, para la generación de un modelo. Para la ejecución del flujo, basta con ejecutar:

```
python3 Project/run_mlops_complete.py
```

Dentro de este flujo, se ejecutan las siguientes acciones:

- **Validación del ambiente (versión de Python, librerías, estructura del repositorio, etc.)**

Para esta sección, se hace una validación de la versión de Python, que estén instaladas las librerías necesarias para que funcione todo el código y que además, se cuente con la estructura de CookieCutter. El resultado mostrado es el siguiente:

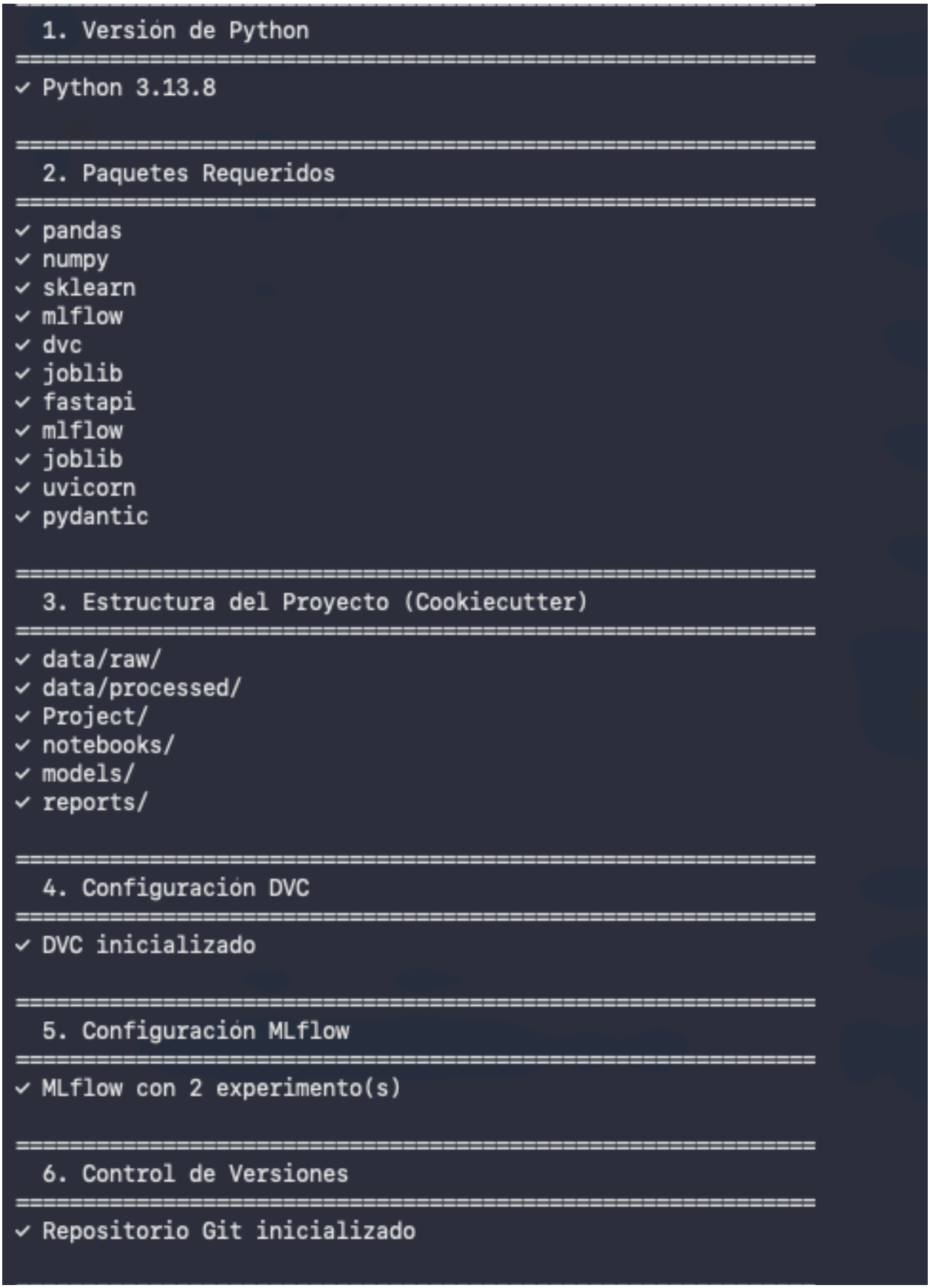

Figura 14: Validación de ambiente.

- **Pull de datasets desde DVC**

Esta sección lo único que hace es recuperar la última versión de los datasets versionados.

```
>>> Step 2/6: DVC Data Pull
--------------------------------------------------------------------------------
[INFO] Checking DVC status...
[INFO] Running: Checking DVC data status
  Command: /Users/oscar.garcia/Documents/Venvs/mlops_venv/bin/python3 -m dvc status
[SUCCESS] DVC data is already up to date
```

Figura 15: Recuperación de Datasets con DVC.

- **Ejecución del flujo de entrenamiento del modelo**

En esta sección se aplica el flujo completo, partiendo desde el dataset original (modificado), pasando por las etapas de preprocesamiento, entrenamiento, registro del modelo (MLFlow) y evaluación del mismo.

```
>>> Step 3/6: Model Training Pipeline
------------------------------------------------------------------------------
[INFO] Attempting full pipeline with MLflow...
[INFO] Running: Executing full ML pipeline (load, preprocess, train, evaluate)
  Command: /Users/oscar.garcia/Documents/Venvs/mlops_venv/bin/python3 /Users/oscar.garcia/Library/Mobile Documents/com~apple~CloudDocs/Education/MNA_IT
ESM/5TrimSep2025Dic2025/MLOps/Proyecto_MLOps/MNA_MLOps_clean/Project/run_full_pipeline.py
[SUCCESS] Training pipeline completed successfully

============================================================
                    MLOps Pipeline - Equipo 43
============================================================

Pipeline started at: 2025-11-17 23:41:19

>>> Step 1: Loading Raw Data
------------------------------------------------------------
[OK] Loaded dataset: 53,464 rows × 10 columns
[OK] Source: /Users/oscar.garcia/Library/Mobile Documents/com~apple~CloudDocs/Education/MNA_ITESM/5TrimSep2025Dic2025/MLOps/Proyecto_MLOps/MNA_MLOps_cl
ean/data/raw/power_tetouan_city_modified.csv

>>> Step 2: Preprocessing Data
------------------------------------------------------------
  -> Executing preprocessing pipeline...

[OK] Preprocessing complete: 52,418 rows × 15 columns
[OK] Saved to: /Users/oscar.garcia/Library/Mobile Documents/com~apple~CloudDocs/Education/MNA_ITESM/5TrimSep2025Dic2025/MLOps/Proyecto_MLOps/MNA_MLOps_
clean/data/processed/power_tetouan_city_processed.csv

  Missing values after preprocessing:
  [OK] No missing values!

>>> Step 3: Training Machine Learning Model
------------------------------------------------------------
  -> Using Gradient Boost Regressor
     - n_estimators: 600
     - learning_rate: 0.1
     - max_depth: 5
     - min_samples_split: 5
     - min_samples_leaf: 3
     - random_state: 42

  -> Training model...
🌐 Using remote MLflow tracking on DagsHub
Accessing as garc1a0scar
Initialized MLflow to track repo "garc1a0scar/mna-mlops-team43"
Repository garc1a0scar/mna-mlops-team43 initialized!
Starting model training...
Training complete.
Model successfully saved to: /Users/oscar.garcia/Library/Mobile Documents/com~apple~CloudDocs/Education/MNA_ITESM/5TrimSep2025Dic2025/MLOps/Proyecto_ML
Ops/MNA_MLOps_clean/models/best_model_pipeline.joblib
Uploading artifacts...
MLFlow Run ID: 7931f0d1d1a54dbd81918dce1a35b429
MLFLOW_TEST_RUN_ID:7931f0d1d1a54dbd81918dce1a35b429

Model performance on the x_test dataset:
Test RMSE: 2832.299
🏃 View run AutoRun_20251117_234120_PowerConsumption_Zone2_20251117_2341 at: https://dagshub.com/garc1a0scar/mna-mlops-team43.mlflow/#/experiments/3/ru
ns/7931f0d1d1a54dbd81918dce1a35b429
🧪 View experiment at: https://dagshub.com/garc1a0scar/mna-mlops-team43.mlflow/#/experiments/3

[OK] Model training complete!
[OK] Model saved to: /Users/oscar.garcia/Library/Mobile Documents/com~apple~CloudDocs/Education/MNA_ITESM/5TrimSep2025Dic2025/MLOps/Proyecto_MLOps/MNA_
MLOps_clean/models/best_model_pipeline.joblib
[OK] Test set size: 10,484 samples

>>> Step 4: Evaluating Model Performance
------------------------------------------------------------

  Model Performance Metrics:
  ------------------------------------------------
  RMSE (Root Mean Squared Error):       2832.30 kW
  MAE  (Mean Absolute Error):           2176.59 kW
  MAPE (Mean Absolute % Error):            8.67 %
  R²   (Coefficient of Determination):   0.7304
  ------------------------------------------------

  Performance vs. Target Metrics:
  RMSE < 4000 kW:     [PASS]
  MAPE < 12%:         [PASS]
  R² > 0.9:           [FAIL]
✔ Métricas del modelo ACTUAL guardadas en: outputs/actual_metrics.json
✔ Parametros del modelo ACTUAL guardados en: outputs/actual_params.json

------------------------------------------------------------------------
                    Pipeline Execution Summary
------------------------------------------------------------------------

[SUCCESS] All steps completed successfully!

Pipeline outputs:
  1. Processed dataset: /Users/oscar.garcia/Library/Mobile Documents/com~apple~CloudDocs/Education/MNA_ITESM/5TrimSep2025Dic2025/MLOps/Proyecto_MLOps/M
NA_MLOps_clean/data/processed/power_tetouan_city_processed.csv
  2. Trained model:     /Users/oscar.garcia/Library/Mobile Documents/com~apple~CloudDocs/Education/MNA_ITESM/5TrimSep2025Dic2025/MLOps/Proyecto_MLOps/M
NA_MLOps_clean/models/best_model_pipeline.joblib
  3. Model metrics:     Logged to MLflow
```

Figura 16: Flujo de entrenamiento de modelo.

- **Ejecución de pruebas con pytest**

Una vez generado el modelo, se procede a realizar pruebas unitarias y de integración con el modelo y data sintética. Dentro de estas pruebas se encuentra: validación de preprocesamiento, extracción de características, creación del pipeline, pruebas de entrenamiento y predicción, entre otras.

Para esta ejecución, bastó con integrar el siguiente comando, dentro de nuestro pipeline completo:

```
pytest tests/ -v --tb=short
```

Este comando toma toda la carpeta tests, donde tenemos nuestro código para la realización de cada una de las pruebas mencionadas anteriormente.

Figura 17: Output de pruebas con pytest.

- **Generación de dashboard de monitoreo y Data Drifting**

Finalmente, el flujo genera el dashboard montado en Evidently, que se mostró en la sección anterior, a partir de datos simulados.

Aunque esta sección se realiza con datos simulados, esto podría aplicarse a nuevos datasets con data «real».

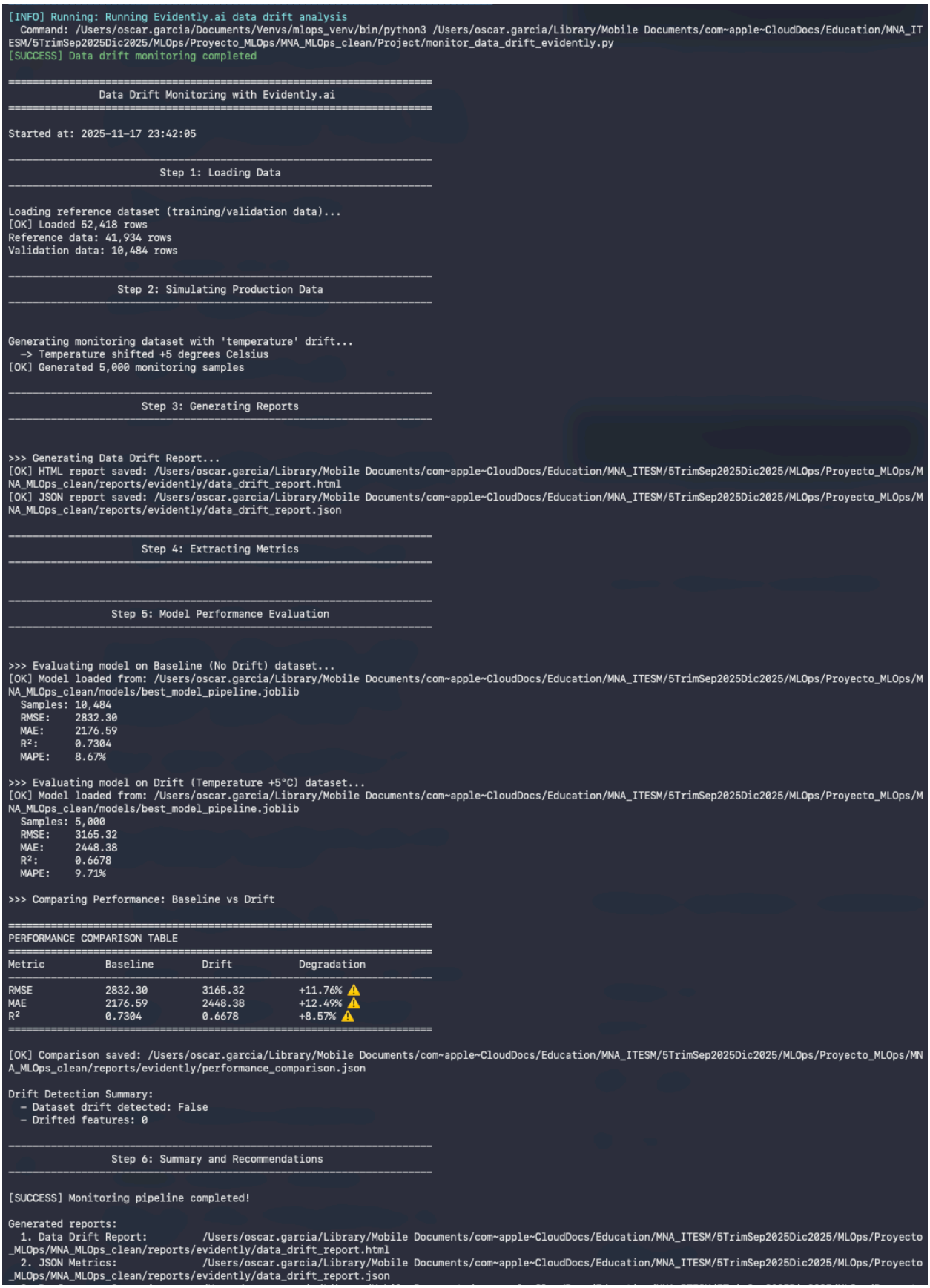

Figura 18: Monitoreo y data drifting en flujo completo.

6.2. Anexo B: Validación de reproducibilidad

Para verificar la reproducibilidad del modelo, el equipo decidió crear un contenedor de docker, instalar las mismas librerías utilizadas en el proyecto base y ejecutar nuestro pipeline completo, descrito en la sección anterior.

Una vez ejecutado el pipeline completo, se ejecutan unos scripts adicionales que comparan los parámetros, el tamaño del modelo y las métricas guardadas en MLFlow vs las mismas características para el modelo ejecutado dentro del contenedor.

Finalmente, si se detecta alguna diferencia el script la imprimirá y fallará la validación global de reproducibilidad.

reproducibility > Dockerfile

```
1   FROM python:3.13-slim
2
3   WORKDIR /app
4   ENV PYTHONUNBUFFERED=1
5
6   RUN apt-get update && \
7       apt-get install -y --no-install-recommends \
8           build-essential \
9           libffi-dev \
10          git && \
11      rm -rf /var/lib/apt/lists/*
12
13  COPY requirements.txt /app/requirements.txt
14  RUN pip install --no-cache-dir -r requirements.txt
15
16  # Copiar carpetas (basado en su ubicación real dentro de reproducibility)
17  COPY Project /app/Project
18  COPY data /app/data
19  COPY models /app/models
20  COPY .dvc /app/.dvc
21
22  ENTRYPOINT ["python"]
23
```

Figura 19: Dockerfile de contenedor para reproducibilidad

reproducibility > Makefile

```
IMAGE_NAME = reproducibility_image
MLFLOW_TRACKING_URI = https://dagshub.com/garc1a0scar/mna-mlops-team43.mlflow
OUTPUT_DIR = outputs

.PHONY: build extract train compare full

build:
    docker build -t $(IMAGE_NAME) .

# 1) Extraer métricas desde MLflow (reference) → outputs/reference_*.json
extract: build
    mkdir -p $(OUTPUT_DIR)
    docker run --rm \
        -e MLFLOW_TRACKING_URI=$(MLFLOW_TRACKING_URI) \
        -v $(OUTPUT_DIR):/app/outputs \
        $(IMAGE_NAME) \
        Project/extract_reference_from_mflow.py

# 2) Entrenar pipeline y producir test model + test metrics
train: build
    docker run --rm \
        -v $(OUTPUT_DIR):/app/outputs \
        -v models:/app/models \
        $(IMAGE_NAME) \
        Project/run_full_pipeline.py --model_path_override "/app/models/test_model_pipeline.joblib"

# 3) Comparar reference vs test
compare: build
    docker run --rm \
        -v $(OUTPUT_DIR):/app/outputs \
        -v models:/app/models \
        $(IMAGE_NAME) \
        Project/compare_models.py

full: extract train compare

```

Figura 20: Makefile para verificación de reproducibilidad

Para ejecutar el pipeline de verificación de reproducibilidad, falta con ejecutar el siguiente comando, una vez que se tiene el *Dockerfile* y el *Makefile* creados como en las imágenes anteriores:

```
make full
```

7. Hallazgos, aprendizajes y conclusiones

Durante estos 3 meses, estuvimos trabajando en este proyecto, que nos llevó a los siguientes puntos clave y aprendizajes.

7.1. Hallazgos clave

- **Industrialización exitosa**: se logró industrializar un proyecto de Machine Learning mediante la adopción de las mejores prácticas de MLOps, asegurando reproducibilidad, escalabilidad y mantenibilidad del modelo a lo largo de su ciclo de vida.
- **Adopción de estándares**: la utilización de la plantilla CookieCutter estableció una estructura de repositorio estandarizada desde el inicio, facilitando la colaboración dentro del equipo y el mantenimiento del código.
- **Mejores prácticas**: Se implementaron herramientas como *Liners* y *Formatters* para asegurar un estilo de código consistente en todo el repositorio, mejorando el formato y legibilidad, *Type Hinting* para definir las entradas y salidas de todas las funciones del pipeline, esto redujo errores de ejecución y a incluir *Docstrings* en todas las funciones permitió describir su propósito, parámetros de entrada y valores de retorno facilitando la comprensión y el onboarding.
- **Trazabilidad garantizada**: al configurar Data Version Control (DVC), inicialmente con Google Drive y después con Amazon S3, se implementó una solución robusta y escalable para el versionamiento de nuestros datasets, manteniendo los repositorios de Github más ligeros y asegurando la trazabilidad del código con la versión exacta de los datos.
- **Gestión de experimentos:** La integración y configuración con MLFlow permitió el seguimiento centralizado de métricas, parámetros e hiperparámetros de múltiples ejecuciones, facilitando la comparación entre resultados para la selección del mejor modelo.
- **Portabilidad y Despliegue Serverless:** La contenerización con Docker, eliminó los problemas comunes de incompatibilidad de entornos, permitiendo un desplieuge de alto rendimiento del API REST en Google Cloud Run.

- **Implementación de CI/CD**: Se establecieron pipelines de integración Continua y Despliegue Continuo (CI/CD), utilizando Github Actions, para automatizar la creación de la imagen Docker de la aplicación y la actualización de los servicios en Cloud Run y Netlify.

7.2. Aprendizajes clave por rol

Rol	Lección aprendida
Data Engineer	El preprocesamiento para un ambiente productivo debe ser automatizable y documentado; para esto, las dataclasses de Python ayudaron a modularizar los objetos, tener un código más limpio, hacerlo reutilizable y escalable.
ML Engineer/Data Scientist	MLFlow transformó la experimentación, pasando de un registro manual a un tracking automático con comparación visual, lo que resultó en un inmenso ahorro de tiempo. De igual forma, al tener scripts propios de cada etapa, se pudieron hacer los flujos completos una y otra vez, sin caer en errores humanos.
MLOps Engineer	La infraestructura de MLOps es una parte muy importante en la productivización de un proyecto de Machine Learning. Una vez que DVC y MLFlow se configuraron correctamente, el equipo pudo trabajar y experimentar mucho más rápido. De igual forma, nos dimos cuenta que implementar un pipeline de CI/CD puede ahorrar mucho tiempo, esfuerzo y error humano dentro de nuestros despliegues.
Software Engineer	Refactorizar ML requiere el uso intensivo de Type Hints y Docstrings para compensar la falta de tipos estrictos en DataFrames, mejorando significativamente la calidad y documentación del código.
DevOps	La estructura inicial, así como el uso de scripts de pipelines y Makefiles ahorra días de setup, y coordinación entre los miembros de todo el equipo. Aunque no lo creamos, la organización inicial ayudó a que todo el equipo pudiera trabajar sin necesidad de explicaciones adicionales.

El equipo concluye que la adopción de un flujo de trabajo estructurado y automatizado, basado en herramientas especializadas como DVC, MLFlow, Docker, Cloud Run y CI/CD, es fundamental para llevar los modelos de Machine Learning de una fase experimental a un servicio operativo robusto, reproducible y escalable.

Aunque el proyecto cumplió con todos los objetivos propuestos (incluido el despliegue del backend y el frontend y la detección de Data Drift) , la implementación de un proceso para realizar un reentrenamiento automático del modelo en respuesta a una alerta de Data Drifting quedó específicamente fuera del alcance definido. Este aspecto se identifica como una potencial mejora futura, que sabemos que es esencial en un ambiente productivo.

8. Anexos

Repositorio de github: https://github.com/oscargarciatec/MNA_MLOps

Cloud Run URL: https://power-predictor-api-148902248893.us-east1.run.app

Frontend de Predicción: https://powerconsumption-pred.netlify.app/

MLCanvas:https://drive.google.com/file/d/1lkaTHG8GBurP9wN4clxWVNWuDca6bT9p/view

9. Referencias

[1] International Energy Agency (IEA), «Electricity 2025 – Demand: Entering the Age of Electricity». Accedido: 18 de noviembre de 2025. [En línea]. Disponible en: https://www.iea.org/reports/electricity-2025/demand

[2] DrivenData, «Cookiecutter Data Science: A logical, reasonably standardized, but flexible project structure for doing and sharing data science work». Accedido: 1 de septiembre de 2025. [En línea]. Disponible en: https://drivendata.github.io/cookiecutter-data-science/

[3] M. Zaharia, «MLflow: A Machine Learning Lifecycle Platform». Accedido: 1 de octubre de 2025. [En línea]. Disponible en: https://mlflow.org/docs/latest/

[4] D. Petrov, «DVC: Data Version Control». Accedido: 1 de septiembre de 2025. [En línea]. Disponible en: https://dvc.org/doc

[5] F. Pedregosa, «Scikit-learn: Machine Learning in Python». Accedido: 1 de octubre de 2025. [En línea]. Disponible en: https://scikit-learn.org/

[6] A. Salam y A. El Hibaoui, «Power Consumption of Tetouan City». Accedido: 1 de septiembre de 2025. [En línea]. Disponible en: https://archive.ics.uci.edu/dataset/849/

[7] GitHub, «How to build a CI/CD pipeline with GitHub Actions in four steps». Accedido: 11 de noviembre de 2025. [En línea]. Disponible en: https://github.blog/enterprise-software/ci-cd/build-ci-cd-pipeline-github-actions-four-steps/