

Departamento de Matemáticas y Física



## **Desarrollo a productivo de un modelo predictivo de monóxido de carbono**

---

**MACHINE LEARNING OPERATIONS BOOTCAMP**

Presenta: **DANIEL MARTINEZ ESCOBOSA**

Guadalajara, Jalisco. mayo de 2024.

# TABLA DE CONTENIDO

TABLA DE CONTENIDO .....	2
LISTA DE FIGURAS.....	3
LISTA DE TABLAS.....	3
<b>1. INTRODUCCIÓN.....</b>	<b>4</b>
1.1. PROBLEMA .....	4
1.2. OBJETIVOS.....	4
1.2.1. Objetivo General .....	4
1.2.2. Objetivos Específicos .....	4
<b>2. MARCO TEÓRICO/CONCEPTUAL .....</b>	<b>5</b>
2.1. TÉCNICAS ESTADÍSTICAS Y MODELOS PREDICTIVOS DE REGRESIÓN .....	5
2.1.1. ANÁLISIS DE CORRELACIÓN .....	5
2.1.2. ANÁLISIS DE COMPONENTES PRINCIPALES (PCA) .....	6
2.1.3. REGRESIÓN LINEAL MÚLTIPLE.....	7
2.1.4. ÁRBOLES DE DECISIÓN .....	8
2.1.4.1. BOLSA DE ÁRBOLES DE DECISIÓN .....	10
2.1.5. RED NEURONAL.....	11
<b>3. DESARROLLO METODOLÓGICO .....</b>	<b>14</b>
3.1. <i>PROCEDENCIA Y MÉTODO DE OBTENCIÓN DE DATOS</i> .....	14
3.2. <i>ANÁLISIS EXPLORATORIO DE DATOS (EDA)</i> .....	14
3.3. <i>DETERMINAR PREGUNTA DEL MODELO DE APRENDIZAJE</i> .....	17
3.4. JUSTIFICACIÓN ESTRATEGIA DE MLOPS PARA DATA SET.....	17
3.5. <i>ARQUITECTURA PIPELINE INICIATIVA DE APRENDIZAJE AUTOMÁTICO</i> .....	19
3.6. <i>EJERCICIO INICIAL: MODELADO DE REFERENCIA</i> .....	20
<b>4. PROCEDIMIENTO Y RESULTADOS .....</b>	<b>21</b>
4.1. DESARROLLO PIPELINE AUTOMATIZADO CON DVC .....	21
4.2. EXPERIMENTACIÓN Y OPTIMIZACIÓN DE MODELOS CON DVC.....	22
4.3. IMPLEMENTACIÓN DE MODELO EN ENTORNO LOCAL CON CONTENEDORES .....	26
4.4. PLANIFICADO DE REENTRENAMIENTO, DRIFT, REDEPLOY, ESCALADO Y MONITOREO .....	29
<b>5. CONCLUSIONES .....</b>	<b>31</b>
5.1. <i>CONCLUSIÓN PERSONAL</i> .....	31
BIBLIOGRAFÍA.....	31
APÉNDICE A. GLOSARIO DE VARIABLES .....	32

## LISTA DE FIGURAS

Figura 1. Componente principal Z1 dirección mayor varianza	6
Figura 2. Esquema del árbol de decisión	9
Figura 3. Esquema del modelo de bolsa de árboles de decisión	11
Figura 4. Representación neurona artificial	11
Figura 5. Capas de una red neuronal	12
Figura 6. Niveles de Monóxido de Carbono CO(GT) en el tiempo	15
Figura 7. Temperatura "T" en grados Celsius durante el año	15
Figura 8. Estudio de distribuciones y presencia de outliers	16
Figura 9. Mapa de calor de correlación entre pares de variables	16
Figura 10. Screenshot de jupyter notebooks (primera versión)	20
Figura 11. Jupyter notebook step_4 modularización	21
Figura 12. Creación de primer stage "data_load.py"	22
Figura 13. Ejemplo de configuración de hiperparámetros en params.yaml	23
Figura 14. Evidencia de haber corrido con éxito pipeline automatizado	23
Figura 15. Evidencia de métricas que guarda evaluate.py y permiten iterar	24
Figura 16. Evidencia de "Ok" en pruebas unitarias en etapa data_load.py del pipeline	25
Figura 17. Evidencia Dockerfile	26
Figura 18. Evidencia Imagen de Docker	27
Figura 19. Evidencia Run de Contenedor Docker	27
Figura 20. Comprobación de creación de contenedor desde Docker desktop	27
Figura 21. Evidencia Predict y comprobación despliegue exitoso	28

## LISTA DE TABLAS

Tabla 1. Modelo Inicial: KPIs Coeficiente de Determinación y RMSE	20
---	----

---

# 1. INTRODUCCIÓN

---

## 1.1. Problema

Abordar un desafío identificando las razones fundamentales que lo justifican. Se resolverá un problema común de aprendizaje automático (problema de regresión) utilizando un data set proporcionado y aplicando lo aprendido en el entrenamiento.

## 1.2. Objetivos

### *1.2.1. Objetivo General*

Los objetivos generales del programa radican en:

- Aplicar conocimientos teóricos en un contexto práctico.
- Demostrar competencia en los temas cubiertos del programa.
- Demostrar habilidades para resolver problemas en escenarios del mundo real utilizando un data set.

### *1.2.2. Objetivos Específicos*

1. Analizar y comprender el data set proporcionado mediante un Análisis Exploratorio de Datos (EDA por sus siglas en inglés).
2. Determinar la pregunta que deseas hacer con un modelo de aprendizaje automático.
3. Identificar por qué se necesita una estrategia de MLOps para este data set (Módulo 1).
4. Diseñar la arquitectura del pipeline para esta nueva iniciativa de aprendizaje automático.
5. Crear un modelo de referencia para resolver tareas asociadas de predicción (clasificación, regresión, etc.) y que responda a la pregunta que te hiciste. Este modelo no necesita tener una precisión, recall o F1-Score altos; sino de crear un modelo rápido para iterar (Módulo 4). Elegir sus características y los hiperparámetros iniciales para este modelo.
6. Configurar la estructura correcta del modelo con la idea de dejarlo listo para implementarse (Módulo 6).
7. Crear nuevas versiones del modelo, que incluyan cambios en las características o ajustes de hiperparámetros para la reproducibilidad y el seguimiento experimental (Módulo 7).
8. Implementar un modelo en su entorno local usando contenedores y planificar el reentrenamiento, drift, redeploy, escalado y monitoreo (Módulo 8).
9. Demostrar estrategias de prueba y versionamiento (Módulos 7 y 8).

---

## 2. MARCO TEÓRICO/CONCEPTUAL

---

### 2.1. Técnicas estadísticas y modelos predictivos de regresión

#### 2.1.1. Análisis de correlación

La correlación, también conocida como coeficiente de correlación lineal (de Pearson), es una medida de regresión que pretende cuantificar el grado de variación conjunta entre dos variables.

Para estudiar la relación lineal existente entre dos variables continuas es necesario disponer de parámetros que permitan cuantificar dicha relación. Uno de estos parámetros es la covarianza, que indica el grado de variación conjunta de dos variables aleatorias:

$$(1) \quad Cov(X, Y) = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{N - 1}$$

siendo  $\bar{x}$  e  $\bar{y}$  las medias de cada variable y  $x_i$  e  $y_i$  el valor de las variables para la observación  $i$ .

La covarianza depende de las escalas en que se miden las variables estudiadas, por lo tanto, no es comparable entre distintos pares de variables. Para poder hacer comparaciones se estandariza la covarianza, generando lo que se conoce como coeficientes de correlación. Existen diferentes tipos, de entre los que destacan el coeficiente de Pearson, Rho de Spearman y Tau de Kendall. Todos ellos varían entre +1 y -1. Siendo +1 una correlación positiva perfecta y -1 una correlación negativa perfecta.

La correlación de Pearson, es la más común. Funciona bien con variables cuantitativas que tienen una distribución normal (e incluso se le considera robusto a pesar de la falta de normalidad), aunque es más sensible a los valores extremos que las otras dos alternativas.

$$(2) \quad r_{xy} = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^n (x_i - \bar{x})^2 \sum_{i=1}^n (y_i - \bar{y})^2}}$$

### 2.1.2. Análisis de Componentes Principales (PCA)

El Análisis de Componentes Principales (PCA por sus siglas en inglés) es un método estadístico de aprendizaje no supervisado que permite simplificar la complejidad de espacios muestrales con muchas dimensiones a la vez que conserva su información.

Las componentes principales se obtienen por combinación lineal de las variables originales. Se pueden entender como nuevas variables obtenidas al combinar de una determinada forma las variables originales. La primera componente principal de un grupo de variables  $(X_1, X_2, \dots, X_p)$  es la combinación lineal normalizada de dichas variables que tiene mayor varianza:

$$(3) \quad Z_1 = \varphi_{11}X_1 + \varphi_{21}X_2 + \dots + \varphi_{p1}X_p$$

Y que entendido de forma gráfica se puede observar la Figura 9.

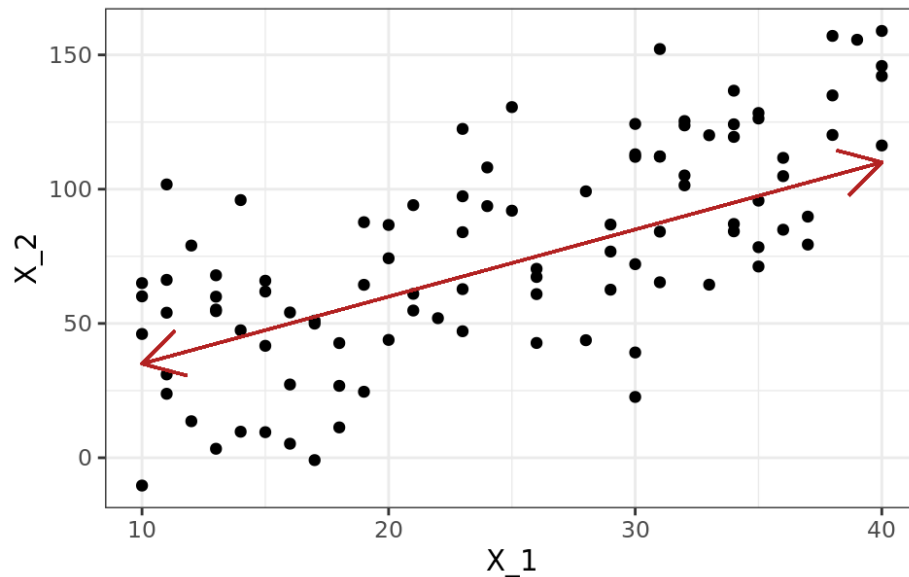


Figura 1. Componente principal Z1 dirección mayor varianza

Que la combinación lineal sea normalizada implica que:

$$(4) \quad \sum_{j=1}^p \varphi_{j1}^2 = 1$$

Los términos  $\varphi_{11}, \dots, \varphi_{1p}$  reciben en el nombre de loadings y son los que definen a la componente. Por ejemplo,  $\varphi_{11}$  es el loading de la variable  $X_1$  de la primera componente principal. Los loadings pueden interpretarse como el peso o importancia que tiene cada variable en cada componente y, por lo tanto, ayudan a conocer la información que recoge cada una de las componentes.

Para conocer cuanta información es capaz de capturar cada una de las componentes principales obtenidas, se recurre a la proporción de varianza explicada por cada una de ellas. Asumiendo que las variables se han normalizado para tener media cero, la varianza total presente en el set de datos se define como:

$$(5) \quad \sum_{j=1}^p \text{Var}(X_j) = \sum_{j=1}^p \frac{1}{n} \sum_{i=1}^n x_{ij}^2$$

y la varianza explicada por la componente  $m$  es:

$$(6) \quad \frac{1}{n} \sum_{i=1}^n z_{im}^2 = \frac{1}{n} \sum_{i=1}^n \left( \sum_{j=1}^p \varphi_{jm} x_{ij} \right)^2$$

Tanto la proporción de varianza explicada, como la proporción de varianza explicada acumulada son dos valores de gran utilidad a la hora de decidir el número de componentes principales a utilizar para reducción de dimensionalidad.

### 2.1.3. Regresión lineal múltiple

La regresión lineal múltiple permite generar un modelo lineal en el que el valor de la variable dependiente o respuesta ( $Y$ ) se determina a partir de un conjunto de variables independientes llamadas predictores ( $X_1, X_2, X_3, \dots$ ). Los modelos de regresión múltiple pueden emplearse para predecir el valor de la variable dependiente o para evaluar la influencia que tienen los predictores sobre ella.

Los modelos lineales múltiples siguen la siguiente ecuación:

$$(7) \quad Y_i = (\beta_0 + \beta_1 X_{1i} + \beta_2 X_{2i} + \dots + \beta_n X_{ni}) + e_i$$

En donde:

$\beta_0$ : es la ordenada en el origen, el valor de la variable dependiente Y cuando todos los predictores son cero.

$\beta_i$ : se conocen como coeficientes parciales de regresión. Representan el efecto promedio que tiene el incremento en una unidad de la variable predictora  $X_i$  sobre la variable dependiente Y, manteniéndose constantes el resto de las variables.

$e_i$ : es el residuo o error, la diferencia entre el valor observado y el estimado por el modelo.

#### 2.1.4. Árboles de decisión

Un árbol de decisión es un conjunto de oraciones incrustadas de la forma “si, entonces” que dividen el conjunto de datos de los predictores. Las particiones en el conjunto de predictores se utilizan para estimar la salida del modelo como se ejemplifica en la Figura 10.

La estructura de un árbol se basa principalmente en 3 puntos principales:

- El predictor o predictores que se utilizarán y el punto de partición del conjunto de datos.
- La profundidad o complejidad del árbol.
- La ecuación de predicción en los nodos terminales u hojas del árbol.



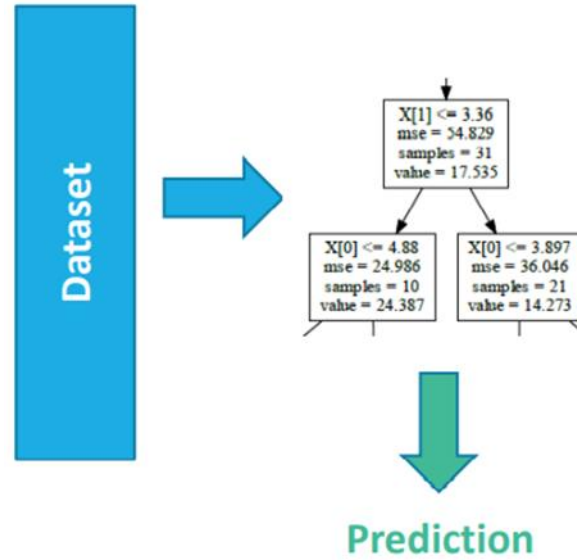


Figura 2. Esquema del árbol de decisión

Para un problema de regresión, se requiere encontrar o proponer un modelo para lograr un mapeo entre dos conjuntos de datos.

( 8 )

$$Y = f(X)$$

donde  $Y$  y  $X$  son el conjunto de variables de salida o a estimar, y el conjunto de predictores o variables de entrada, respectivamente. La función  $f(\cdot)$  será aproximada por el árbol de decisión.

Para la construcción de un árbol de decisión, se considera el conjunto de datos completo  $S$ , donde  $X, Y \in S$ . El conjunto de datos  $S$  se divide mediante una búsqueda exhaustiva de todas las particiones posibles para crear dos subconjuntos de datos  $S_1$  y  $S_2$ , de tal forma que se minimiza un criterio de varianza conjunta,  $L(x_{split})$ .

( 9 )

$$L(x_{pj,split}) = \frac{1}{n_1} \sum_{i \in S_1} (y_i - \bar{y}_1)^2 + \frac{1}{n_2} \sum_{i \in S_2} (y_i - \bar{y}_2)^2$$

Donde,  $x_{pj,split}$  es un valor de partición en el predictor  $pj$  y los parámetros  $\bar{y}_1$  y  $\bar{y}_2$  son los promedios de las muestras de la variable de salida pertenecientes a los respectivos subconjuntos  $S_1$  y  $S_2$ .

El valor de la función  $L(x_{px,split})$ , depende de la mejor  $x_{px,split}$ , que se selecciona para minimizar la varianza en la variable de salida  $Y$ . La búsqueda de la mejor partición debe realizarse de forma exhaustiva en todos los predictores  $x_{pj}$ , y solo se elige una partición de entre todos los predictores.

Este procedimiento se realiza recursivamente en los subconjuntos  $S1$  y  $S2$ .

La función de predicción en cada hoja del árbol se define como el valor promedio de la salida muestras del subconjunto de datos  $S_j$  correspondientes a la  $j$ -ésima rama o nodo final. La predicción  $\hat{y}_i$  correspondiente a una muestra  $x_i$  se define como:

$$(10) \quad \hat{y}_{i,j} = \bar{y}_j$$

#### 2.1.4.1. Bolsa de árboles de decisión

El ensamblaje de modelos es una estrategia que intenta resolver el problema de la solución subóptima de árboles de decisión.

El método "Bagging" es relativamente simple ya que utiliza "bootstrapping" junto con diferentes técnicas de regresión para construir un modelo ensamblado. Cada modelo en el empaquetado genera una predicción para la misma muestra, y todas las predicciones son promediadas para dar una predicción definitiva (ver Figura 11). De esta forma, al combinar sus resultados, unos errores se compensan con otros, reduciéndose la varianza en la predicción y haciendo la predicción más estable y con poder de generalización.

La construcción de una bolsa de árboles se hace de forma sencilla:

1. Generar un subconjunto de datos a partir de los datos originales.
2. Entrenar un árbol de decisiones no podado basado en el subconjunto de datos.
3. Repetir los pasos 1 y 2 hasta que se formen  $m$  árboles de decisión.

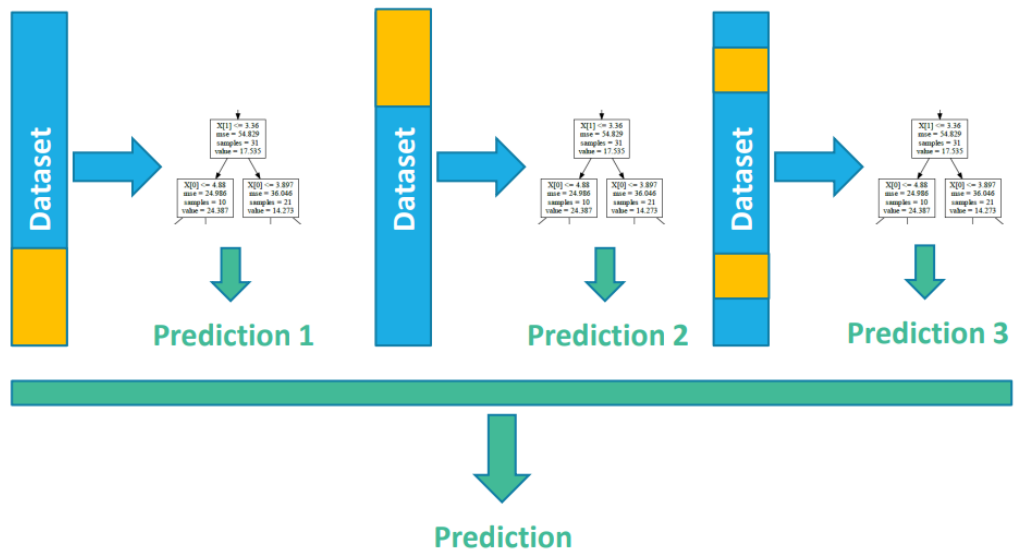


Figura 3. Esquema del modelo de bolsa de árboles de decisión

### 2.1.5. Red Neuronal

Una red neuronal artificial es un modelo altamente no lineal y muy versátil que se usa comúnmente para resolver problemas de regresión o clasificación.

La forma en la que una neurona artificial procesa una señal se representa en dos etapas, tal como se muestra en la Figura 12 y se explica a continuación:

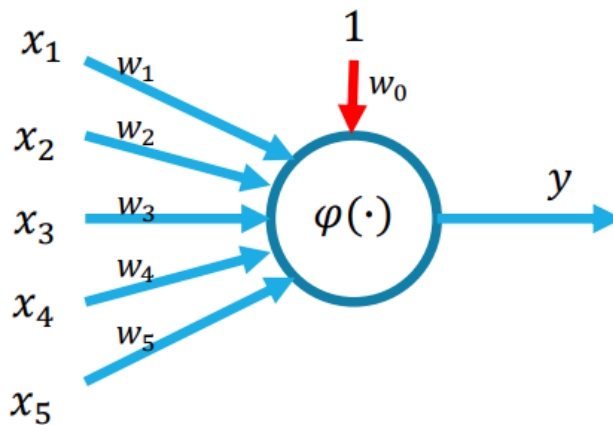


Figura 4. Representación neurona artificial

1. Combinación de las entradas. Todas las señales que recibe la neurona deben ser consideradas para generar una respuesta. Cada señal recibida tiene asociada una

constante  $w$ , que permite manipular la importancia de cada señal recibida o el aporte de información de cada entrada. Su representación matemática se da por la siguiente ecuación:

$$(11) \quad v = w_0 + w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + w_5x_5$$

2. Respuesta neuronal. Considerando todas las señales de entrada, la neurona genera una señal de respuesta  $y = \varphi(v)$  por medio de una función de activación, la cual es diferenciable y ayuda a resolver el problema de optimización en el entrenamiento. Las funciones de activación más comunes son:

$$(12) \quad \text{Lineal: } \varphi(v) = v$$

$$(13) \quad \text{Sigmoidal: } \varphi(v) = \frac{1}{1+e^{-v}}$$

$$(14) \quad \text{Tangente hiperbólica: } \varphi(v) = \tanh(v)$$

Ahora bien, partiendo de la teoría de una neurona artificial, una red neuronal es la interconexión de múltiples neuronas para resolver un problema complejo que una sola neurona no podría. Su organización, como se visualiza en la Figura 13, se realiza en capas para simplificar el análisis: de entrada, ocultas y de salida.

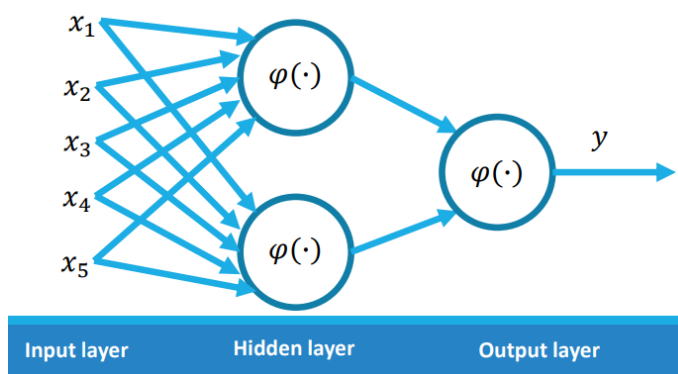


Figura 5. Capas de una red neuronal

Intuitivamente, se puede ver que una red neuronal proporciona un modelo no lineal y esto se puede ajustar por medio de los pesos de cada capa.

Para un problema de regresión, se usa que la función de activación en la capa de salida sea la lineal, así mismo la evaluación de una red neuronal se realiza mediante una función de error o función de costo  $J$ .

$$(15) \quad J = \frac{1}{2n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

Donde,  $y_i$  es la salida deseada,  $\hat{y}_i$  es la salida de la red neuronal para una entrada  $x_i$  y  $n$  es el número de salidas o neuronas en la capa de salida.

Por tanto, la evaluación de la red neuronal para un conjunto de muestras  $m$  se puede escribir como:

$$(16) \quad J = \frac{1}{2nm} \sum_{j=1}^m \sum_{i=1}^n (y_{i,j} - \hat{y}_{i,j})^2$$

Se considera que un modelo neuronal ya tiene aprendido el comportamiento de los datos a ajustar cuando se tiene un valor mínimo de la función  $J$ . Para lograr esto es necesario encontrar los pesos  $W$  que minimicen la evaluación de  $J$  para todas las muestras de un conjunto de datos.

Ahora bien, cuando se aplica la función de costo  $J$  con un modelo neuronal, los pesos  $W$  tienen una relación no lineal a la salida y esto hace que  $J$  sea una función no convexa. Es decir, no hay una única solución  $W$  que minimiza la función de costo  $J$ . Como ayuda ante ello se utiliza un algoritmo de aprendizaje cuya función es la minimización de  $J$  modificando los parámetros  $W$  de la red neuronal.

Durante el proceso de aprendizaje o entrenamiento de una red neuronal, la actualización de los pesos es iterativamente siguiendo la siguiente regla:

$$(17) \quad W_k = W_{k-1} + \Delta W_{k-1}$$

Los algoritmos de entrenamiento de redes neuronales más conocidos son:

- a) Gradiente descendente
- b) Método de Newton
- c) Gauss-Newton
- d) Levenberg-Marquart

---

## 3. DESARROLLO METODOLÓGICO

---

### 3.1. *Procedencia y método de obtención de datos*

El conjunto de datos contiene 9358 casos de respuestas promediadas por hora de una serie de cinco sensores químicos de óxido metálico integrados en un dispositivo multisensor químico de calidad del aire. El dispositivo estaba ubicado en el campo en una zona significativamente contaminada, al nivel de la carretera, dentro de una ciudad italiana. Los datos se registraron desde marzo de 2004 hasta febrero de 2005 (un año), lo que representa las grabaciones más largas disponibles de forma gratuita de las respuestas de dispositivos de sensores químicos de calidad del aire desplegados en el campo. Ground Truth promedió las concentraciones horarias de CO, hidrocarburos no metánicos, benceno, óxidos de nitrógeno total (NO<sub>x</sub>) y dióxido de nitrógeno (NO<sub>2</sub>) y fueron proporcionadas por un analizador certificado de referencia ubicado en el mismo lugar. Están presentes evidencias de sensibilidades cruzadas, así como desviaciones tanto de conceptos como de sensores, como se describe en De Vito et al., Sens. And Act. B, vol. 129,2,2008 (cita requerida) que eventualmente afecta las capacidades de estimación de concentración de los sensores. Los valores faltantes están etiquetados con el valor -200. Este conjunto de datos se puede utilizar exclusivamente con fines de investigación. Los fines comerciales están totalmente excluidos.

### 3.2. *Análisis Exploratorio de Datos (EDA)*

Haciendo un análisis exploratorio de la información presente en el data set, se encontraron los siguientes hallazgos relevantes:

- En términos de análisis de contaminantes en el tiempo, la variable objetivo llamada "CO(GT)" presentó los niveles más altos durante el periodo de tiempo entre noviembre 2004 a enero 2005, en donde coincide con un decrecimiento en la temperatura como se aprecian en las dos figuras siguientes:

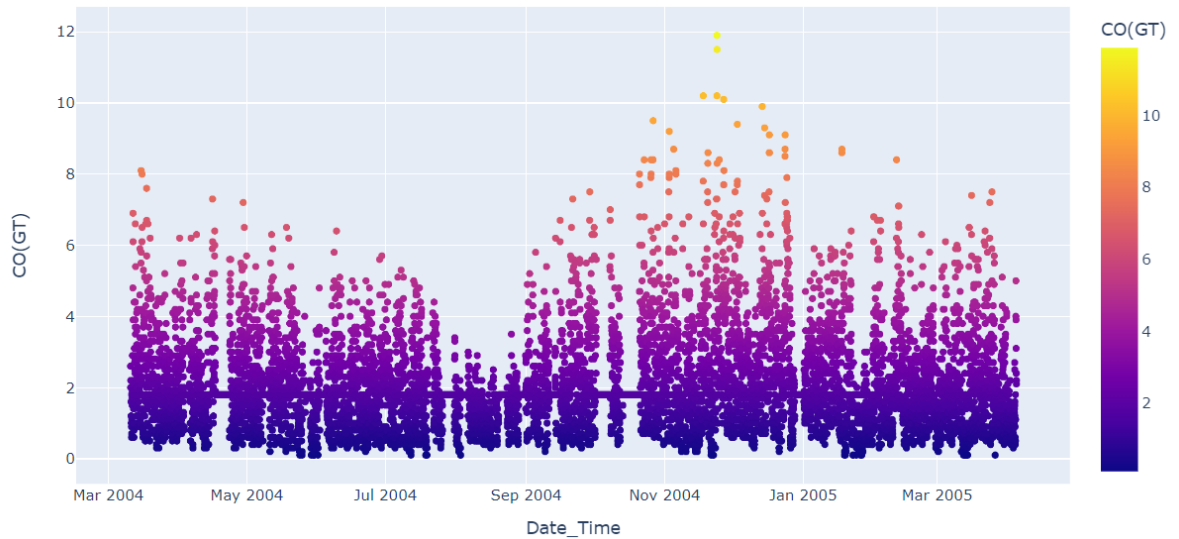


Figura 6. Niveles de Monóxido de Carbono CO(GT) en el tiempo

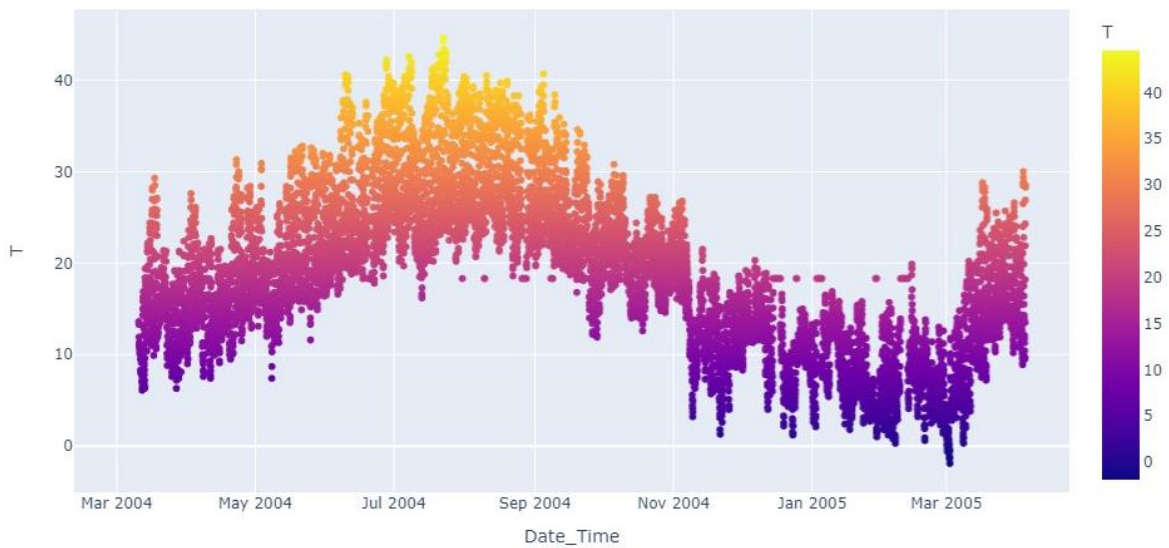


Figura 7. Temperatura "T" en grados Celsius durante el año

- Al estudiar los histogramas previo a tratamiento de datos, se detectó un sesgo moderado en las variables "NMHC(GT)" y "NOx(GT)", cuyos valores de sesgo "Skew" fueron  $>1.5$  pero  $<2$ , o  $<-1.5$  pero  $>-2$  (criterio empírico); mientras que las variables con la distribución más cercana a una normal, fueron "AH", "RH", "T" y "PT08.S4(NO2)" cayendo dentro del rango  $-0.5 < \text{Skew} < 0.5$ .

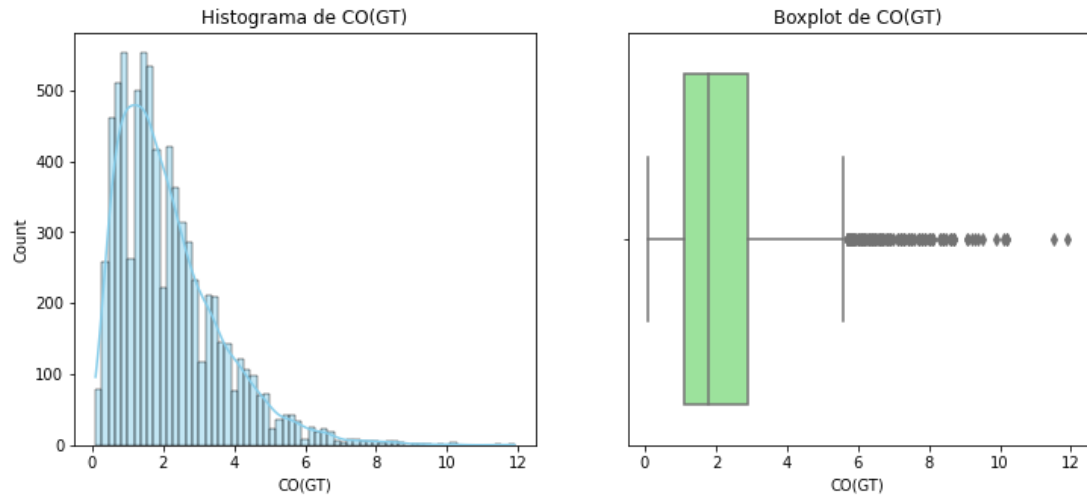


Figura 8. Estudio de distribuciones y presencia de outliers

- Con la ayuda de un análisis de correlación mediante mapas de Seaborn, se detectó que la variable “PT08.S3(NOx)” presentó alto índice de correlación entre sus pares de variables, tal como se aprecia en la siguiente figura:

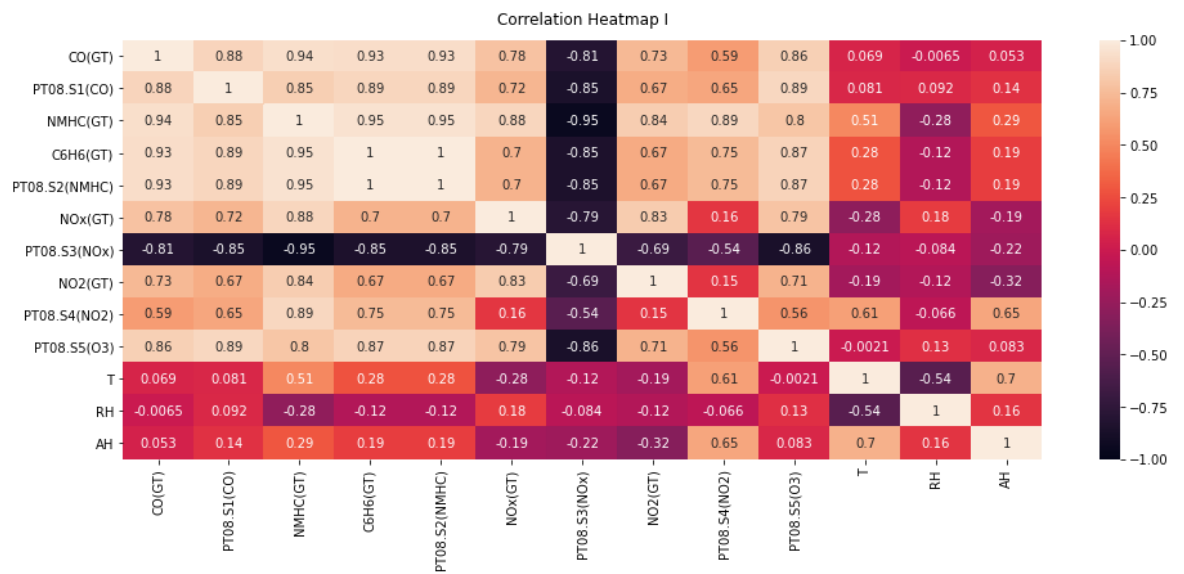


Figura 9. Mapa de calor de correlación entre pares de variables



### 3.3. *Determinar Pregunta del Modelo de Aprendizaje*

El monóxido de carbono (CO(GT)) es conocido como un asesino silencioso debido a su naturaleza inolora, incolora, insípida y no irritante, lo que lo hace especialmente peligroso. Los primeros síntomas de intoxicación por CO son difíciles de detectar, lo que lleva a que se le denomine 'asesino silencioso'. En este proyecto de MLOPs, nos planteamos la pregunta de si es factible estimar los niveles de CO(GT) a partir de la medición de otros gases más fácilmente detectables por sensores. Esta investigación busca desarrollar un modelo de aprendizaje que pueda predecir los niveles de CO(GT) basándose en datos de otros gases, con el objetivo de alertar a la población sobre condiciones ambientales peligrosas.

### 3.4. *Justificación estrategia de MLOps para data set*

El dataset de calidad del aire que se utiliza es una fuente valiosa de datos para comprender y predecir la calidad del aire en función de varias variables ambientales. A continuación, se explican algunos puntos por los que este dataset requiere una estrategia de MLOps:

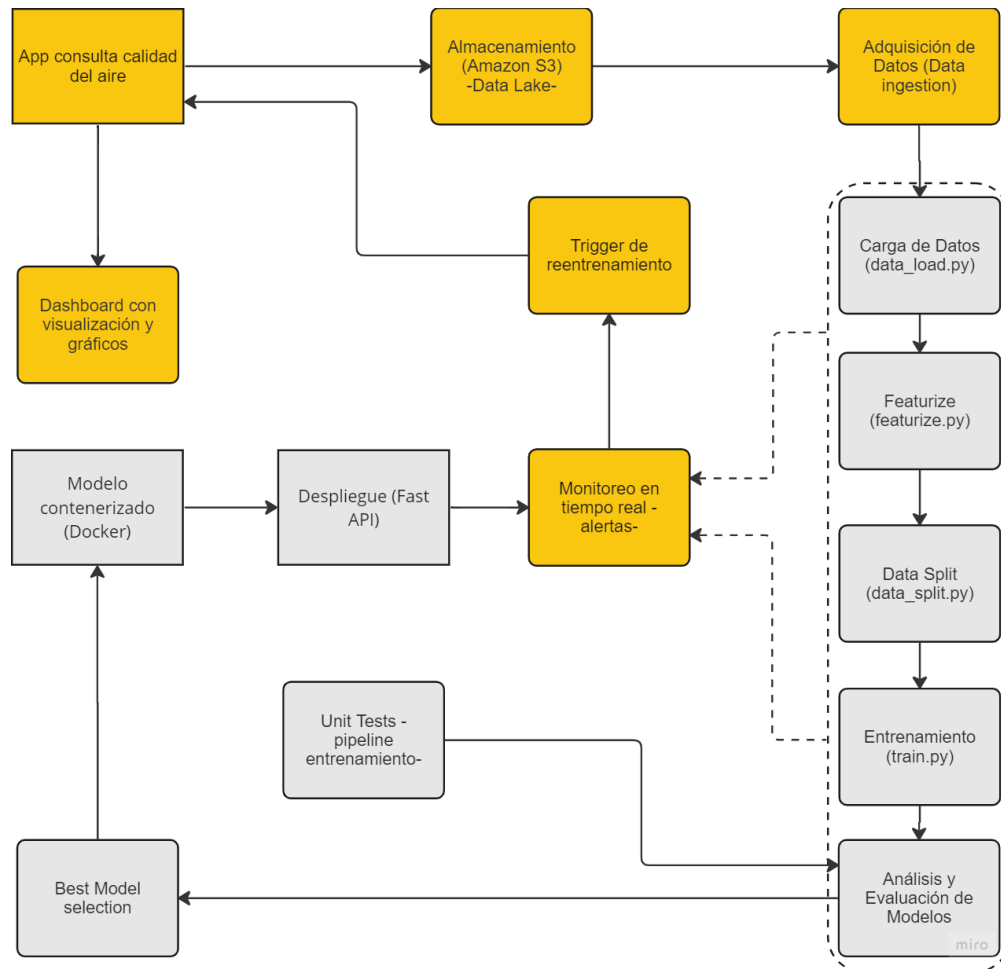
- **Tamaño del dataset:** El dataset contiene registros de sólo un año, pero pensando en múltiples años o incluso en hipotéticas mediciones futuras en menor periodicidad -por ejemplo, mediciones por minuto-, podría manejarse gran volumen de datos. Esto implica desarrollar estrategias de gestión de datos eficientes como técnicas de almacenamiento distribuido, particionamiento de datos y gestión de versiones para asegurar la reproducibilidad de los experimentos.
- **Preprocesamiento de datos:** El dataset de estudio contiene datos incompletos, ruidosos e inconsistentes que necesitan ser preprocesados antes de ser utilizados en modelos de aprendizaje automático. Mediante MLOps se incluiría la automatización de pipelines de preprocesamiento de datos para garantizar la consistencia y la escalabilidad del proceso.
- **Selección de características:** Es necesario identificar las características más relevantes para predecir la calidad del aire es crucial. Con MLOps se puede incluir

la automatización de la selección de características utilizando técnicas como la ingeniería de características automática o la selección de modelos.

- **Entrenamiento de modelos:** Implementar pipelines de entrenamiento de modelos que sean escalables y reproducibles es esencial. Esto implica la configuración de infraestructura de cómputo escalable, la implementación de pipelines de entrenamiento utilizando herramientas como TensorFlow Extended (TFX) o Apache Airflow, y la gestión de recursos para garantizar la eficiencia y la calidad de los modelos.
- **Evaluación y monitorización de modelos en producción:** Una vez que el modelo esté en producción, es importante monitorear su rendimiento y calidad de predicción. Con MLOps se podría incluir en la implementación sistemas de monitorización en tiempo real, alertas automáticas para detectar anomalías en los datos o en el rendimiento del modelo, y la retroalimentación continua para mejorar el modelo con nuevos datos.
- **Despliegue y gestión de modelos en producción:** Implementar estrategias de despliegue de modelos robustas y seguras es fundamental. Esto implica la automatización del proceso de despliegue utilizando contenedores como Docker, la gestión de versiones de modelos, la configuración de infraestructura de producción escalable y la implementación de prácticas de seguridad para proteger los modelos y los datos.

En resumen, la aplicación de estrategias de MLOps en este proyecto ayuda a gestionar eficientemente todas las etapas del ciclo de vida del modelo de aprendizaje automático, desde la gestión de datos hasta el despliegue y monitorización en producción, garantizando así la escalabilidad, reproducibilidad y calidad del modelo.

### 3.5. Arquitectura Pipeline Iniciativa de Aprendizaje Automático



### Descripción del flujo:

El pipeline comienza con la ingesta de datos, donde se cargan datos crudos desde Amazon S3 a un entorno local o en la nube. Luego, estos datos son transformados y preprocesados en la etapa de carga de datos, preparándolos para el procesamiento posterior. A continuación, se generan características en la fase de featurize, optimizando los datos para el entrenamiento del modelo. Los datos se dividen en conjuntos de entrenamiento y prueba en la etapa de división de datos, seguida por el entrenamiento del modelo utilizando los datos de entrenamiento. Posteriormente, el modelo se evalúa con los datos de prueba y se selecciona el mejor modelo basado en métricas de rendimiento. El modelo seleccionado se conteneriza usando Docker y se despliega mediante FASTAPI para permitir consultas de predicción. El pipeline incluye monitoreo en tiempo real de los datos y el rendimiento del modelo, evaluando continuamente métricas como R2 y media

de las variables que entrenan el modelo para entender cambio en el comportamiento de los datos, disparando reentrenamientos si el rendimiento cae por debajo de un umbral de aceptación. Finalmente, los usuarios pueden consultar predicciones diarias a través de una aplicación y visualizar datos y métricas en un tablero interactivo.

**Nota Pipeline:** etapas en color naranja no pertenecen al alcance y objetivos de este proyecto, por lo que se tienen bosquejadas como parte de la solución aunque no se encuentren elementos de su desarrollo en el repositorio remoto Github compartido.

### 3.6. *Ejercicio Inicial: Modelado de Referencia*

Se pretenden desarrollar múltiples modelos, sin embargo, el ejercicio inicial corrido en este proyecto consistió en un modelo de regresión multivariado en el que los KPIs propuestos fueron el coeficiente de determinación ( $R^2$ ) y el RMSE. Nota: para más detalle dirigirse dentro de la carpeta del proyecto a: *airquality\_bootcamp\notebooks\0\_prototype\_1ra\_entrega\_capstonepw\_daniel\_martinez\_e.ipynb*.

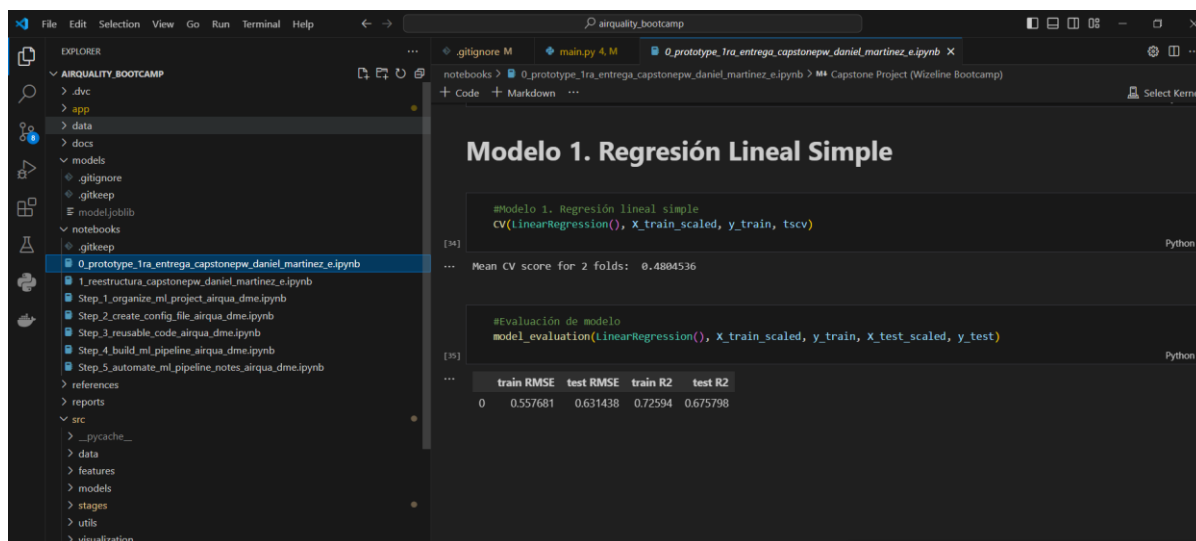


Figura 10. Screenshot de jupyter notebooks (primera versión)

A continuación, los resultados arrojados del ejercicio:

Modelo LR	R2	RMSE
Train	0.7260	0.5517
Test	0.6758	0.6314

Tabla 1. Modelo Inicial: KPIs Coeficiente de Determinación y RMSE

---

## 4. PROCEDIMIENTO Y RESULTADOS

---

### 4.1. Desarrollo Pipeline Automatizado con DVC

En el contexto del Bootcamp de MLOps, se trabajó en convertir un notebook de Jupyter en un pipeline modular utilizando DVC (Data Version Control) en Visual Studio Code. Este procedimiento fue necesario para permitir un flujo de trabajo más organizado y reproducible, esencial para el despliegue en entornos de producción.

A alto nivel de como se explica en la carátula README del repositorio remoto del proyecto, el primer paso consistió en crear un entorno virtual para gestionar las dependencias del proyecto sin interferencias con otros proyectos. Una vez logrado esto se instalaron las dependencias y librerías necesarias para trabajar en el proyecto (incluidas versiones de DVC y DVCLive) y se agregó el entorno virtual a Jupyter notebook para continuar trabajando con otros notebooks en el que cada nueva versión supondría un acercamiento hacia la modularización del código mediante una transición ordenada hacia un flujo de trabajo robusto y escalable.

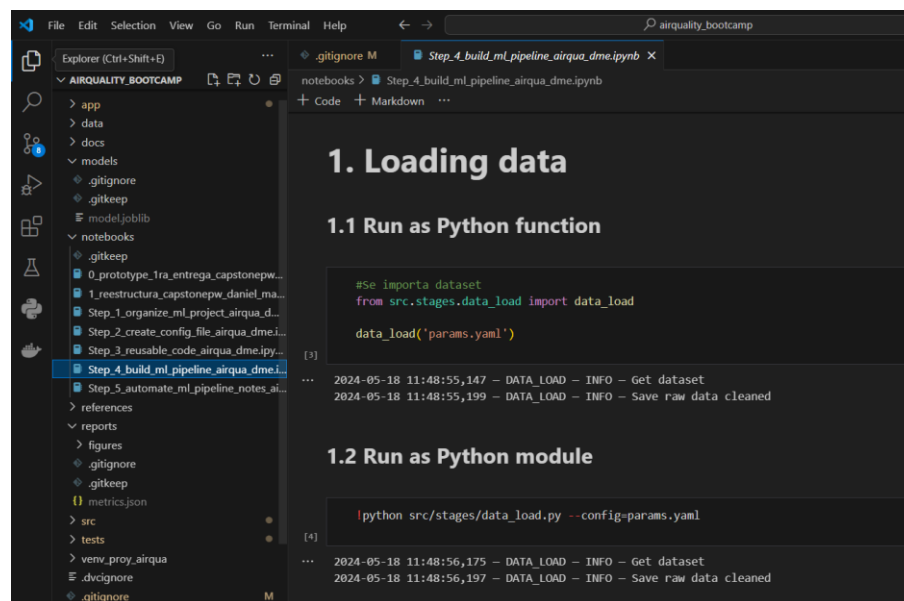
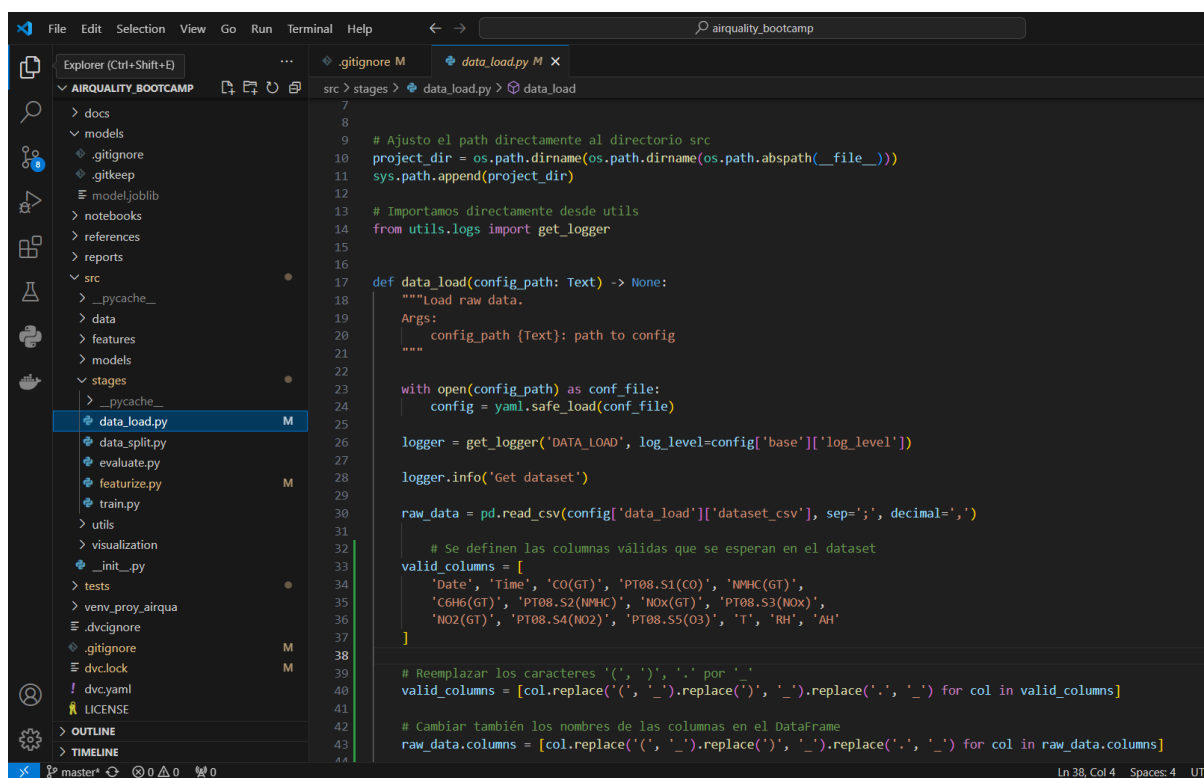


Figura 11. Jupyter notebook step\_4 modularización

A partir del notebook "Step\_4\_build\_ml\_pipeline\_airqua\_dme.ipynb", se realizó la migración del código de cada etapa hacia scripts Python independientes con extensión .py, organizados dentro de un folder con ruta relativa "src/stages" bajo una estructura de directorios basada en la plantilla CookieCutter.

El primer archivo creado fue “data\_load.py” el cual se convirtió en el primer stage del pipeline. De manera similar, se añadieron otros stages, asegurando que cada script dependiera de las salidas del stage anterior y configurando las dependencias y salidas adecuadamente en el archivo dvc.yaml. Cada stage fue probado y reproducido con el comando “dvc repro” hasta finalmente correr con éxito el flujo completo automatizado de inicio a fin como se muestra en la siguiente figura. En añadidura y como parte de las buenas prácticas, se realizaron constantes commits en Git para mantener la integridad del proyecto.



```
7
8
9 # Ajusto el path directamente al directorio src
10 project_dir = os.path.dirname(os.path.abspath(__file__))
11 sys.path.append(project_dir)
12
13 # Importamos directamente desde utils
14 from utils.logs import get_logger
15
16
17 def data_load(config_path: Text) -> None:
18     """Load raw data.
19     Args:
20         config_path (Text): path to config
21     """
22
23     with open(config_path) as conf_file:
24         config = yaml.safe_load(conf_file)
25
26     logger = get_logger('DATA_LOAD', log_level=config['base']['log_level'])
27
28     logger.info('Get dataset')
29
30     raw_data = pd.read_csv(config['data_load']['dataset_csv'], sep=';', decimal=',')
31
32     # Se definen las columnas válidas que se esperan en el dataset
33     valid_columns = [
34         'Date', 'Time', 'CO(GT)', 'PT08.S1(CO)', 'NH4H(GT)',
35         'COH6(GT)', 'PT08.S2(NH4H)', 'NOX(GT)', 'PT08.S3(NOx)',
36         'NO2(GT)', 'PT08.S4(NO2)', 'PT08.S5(O3)', 'T', 'RH', 'AH'
37     ]
38
39     # Reemplazar los caracteres '(', ')', '.' por '_'
40     valid_columns = [col.replace('(', '_').replace(')', '_').replace('.', '_') for col in valid_columns]
41
42     # Cambiar también los nombres de las columnas en el DataFrame
43     raw_data.columns = [col.replace('(', '_').replace(')', '_').replace('.', '_') for col in raw_data.columns]
```

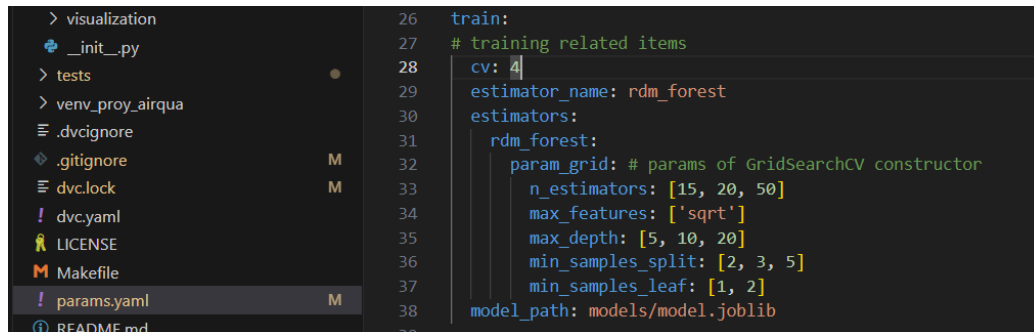
Figura 12. Creación de primer stage “data\_load.py”

## 4.2. Experimentación y Optimización de Modelos con DVC

Una vez que el pipeline de Machine Learning fue automatizado utilizando DVC (Data Version Control), se consideró realizar diversas experimentaciones para optimizar el modelo de Random Forest propuesto como alternativa más sofisticada al ejercicio inicial donde se planteó una regresión lineal multivariada. Este proceso incluyó la modificación de hiperparámetros y la re-ejecución eficiente del pipeline para evaluar distintas configuraciones. A continuación, se detalla el método de experimentación utilizado:

## 1. Modificación de Hiperparámetros:

Para optimizar el modelo de Random Forest, se utilizó el archivo de configuración “params.yaml”, el cual contiene los parámetros necesarios para cada etapa del pipeline. Los hiperparámetros específicos fueron ajustados directamente en este archivo.

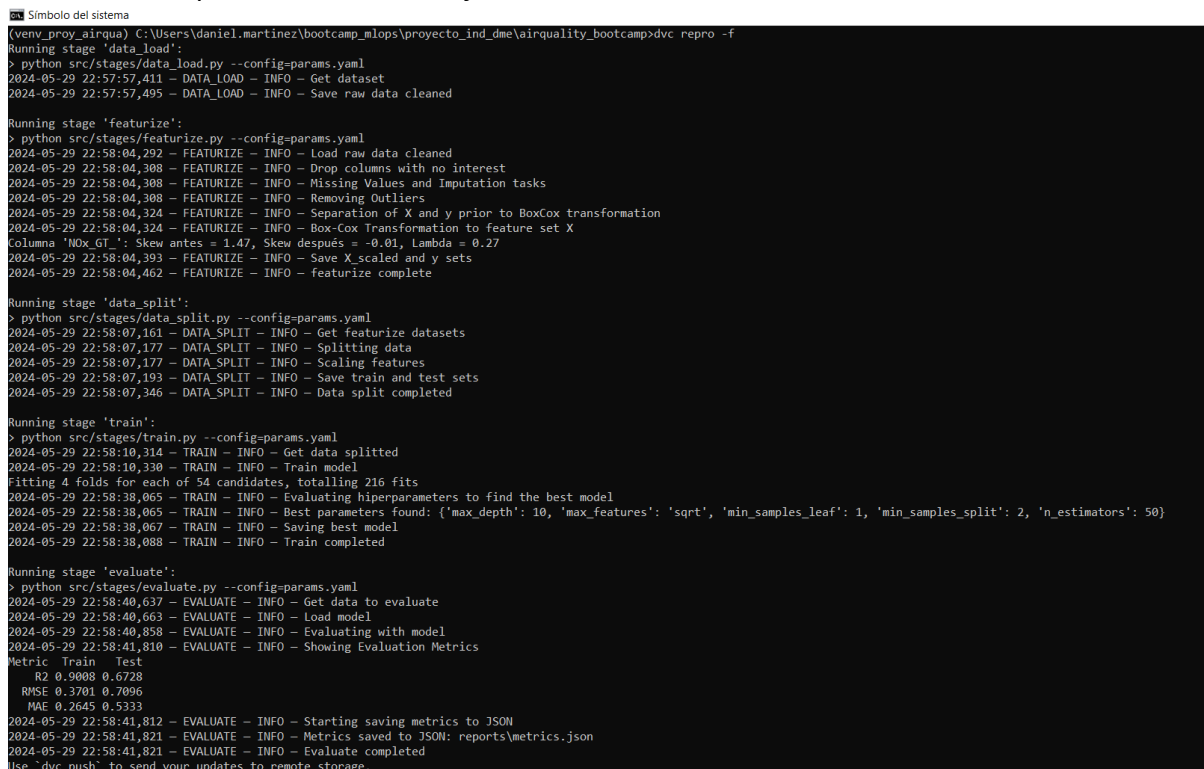


```
26 train:
27 # training related items
28 cv: 4
29 estimator_name: rdm_forest
30 estimators:
31   rdm_forest:
32     param_grid: # params of GridSearchCV constructor
33       n_estimators: [15, 20, 50]
34       max_features: ['sqrt']
35       max_depth: [5, 10, 20]
36       min_samples_split: [2, 3, 5]
37       min_samples_leaf: [1, 2]
38 model_path: models/model.joblib
```

Figura 13. Ejemplo de configuración de hiperparámetros en params.yaml

## 2. Ejecución del Pipeline Completo:

Para evaluar el impacto de los cambios en los hiperparámetros, se ejecutó el pipeline completo utilizando el comando “dvc repro -f” directamente desde el terminal. Este comando fuerza la re-ejecución de todas las etapas del pipeline, garantizando que cada cambio en los parámetros sea reflejado en el modelo final.



```

[venv_proy_airsta] C:\Users\daniel.martinez\bootcamp_mlops\proyecto_ind_dme\airquality_bootcamp>dvc repro -f
Running stage 'data_load':
> python src/stages/data_load.py --config=params.yaml
2024-05-29 22:57:57,411 - DATA_LOAD - INFO - Get dataset
2024-05-29 22:57:57,495 - DATA_LOAD - INFO - Save raw data cleaned

Running stage 'featurize':
> python src/stages/featurize.py --config=params.yaml
2024-05-29 22:58:04,292 - FEATURIZE - INFO - Load raw data cleaned
2024-05-29 22:58:04,308 - FEATURIZE - INFO - Drop columns with no interest
2024-05-29 22:58:04,308 - FEATURIZE - INFO - Missing Values and Imputation tasks
2024-05-29 22:58:04,308 - FEATURIZE - INFO - Removing Outliers
2024-05-29 22:58:04,324 - FEATURIZE - INFO - Separation of X and y prior to BoxCox transformation
2024-05-29 22:58:04,324 - FEATURIZE - INFO - Box-Cox Transformation to feature set X
Columna 'NOx_GT': Skew antes = 1.47, Skew después = -0.01, Lambda = 0.27
2024-05-29 22:58:04,393 - FEATURIZE - INFO - Save X scaled and y sets
2024-05-29 22:58:04,462 - FEATURIZE - INFO - featurize complete

Running stage 'data_split':
> python src/stages/data_split.py --config=params.yaml
2024-05-29 22:58:07,161 - DATA_SPLIT - INFO - Get featurize datasets
2024-05-29 22:58:07,177 - DATA_SPLIT - INFO - Splitting data
2024-05-29 22:58:07,177 - DATA_SPLIT - INFO - Scaling features
2024-05-29 22:58:07,193 - DATA_SPLIT - INFO - Save train and test sets
2024-05-29 22:58:07,346 - DATA_SPLIT - INFO - Data split completed

Running stage 'train':
> python src/stages/train.py --config=params.yaml
2024-05-29 22:58:10,314 - TRAIN - INFO - Get data splitted
2024-05-29 22:58:10,330 - TRAIN - INFO - Train model
Fitting 4 folds for each of 54 candidates, totalling 216 fits
2024-05-29 22:58:38,065 - TRAIN - INFO - Evaluating hyperparameters to find the best model
2024-05-29 22:58:38,065 - TRAIN - INFO - Best parameters found: {'max_depth': 10, 'max_features': 'sqrt', 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 50}
2024-05-29 22:58:38,067 - TRAIN - INFO - Saving best model
2024-05-29 22:58:38,088 - TRAIN - INFO - Train completed

Running stage 'evaluate':
> python src/stages/evaluate.py --config=params.yaml
2024-05-29 22:58:40,637 - EVALUATE - INFO - Get data to evaluate
2024-05-29 22:58:40,663 - EVALUATE - INFO - Load model
2024-05-29 22:58:40,858 - EVALUATE - INFO - Evaluating with model
2024-05-29 22:58:41,810 - EVALUATE - INFO - Showing Evaluation Metrics
Metric Train Test
R2 0.9008 0.6728
RMSE 0.3701 0.7096
MAE 0.2645 0.5333
2024-05-29 22:58:41,812 - EVALUATE - INFO - Starting saving metrics to JSON
2024-05-29 22:58:41,821 - EVALUATE - INFO - Metrics saved to JSON: reports\metrics.json
2024-05-29 22:58:41,821 - EVALUATE - INFO - Evaluate completed
Use 'dvc push' to send your updates to remote storage.
```

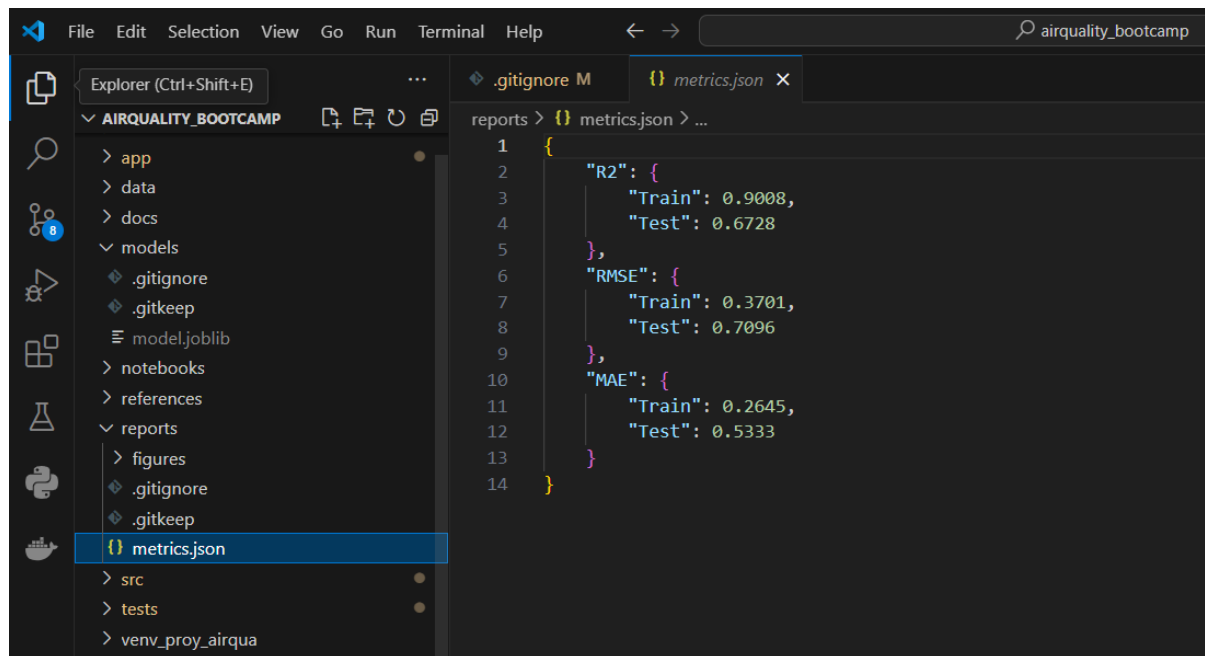
Figura 14. Evidencia de haber corrido con éxito pipeline automatizado

### 3. Ejecución Selectiva de Etapas del Pipeline:

Para hacer el proceso más eficiente, especialmente cuando sólo se modificaban los hiperparámetros del modelo, se ejecutaron únicamente las etapas de entrenamiento (train.py) y evaluación (evaluate.py). Esto se logró especificando estas etapas en el comando `dvc repro`, evitando así la re-ejecución de etapas anteriores que no fueron modificadas.

### 4. Evaluación de Resultados:

Después de cada ejecución, los resultados del modelo, incluyendo las métricas de rendimiento como el coeficiente de determinación (R2), el error cuadrático medio (RMSE) y el error absoluto medio (MAE), fueron revisados. Estas métricas se almacenaron en un archivo JSON, facilitando la comparación de resultados entre diferentes configuraciones de hiperparámetros.



```
1 {
2   "R2": {
3     "Train": 0.9008,
4     "Test": 0.6728
5   },
6   "RMSE": {
7     "Train": 0.3701,
8     "Test": 0.7096
9   },
10  "MAE": {
11    "Train": 0.2645,
12    "Test": 0.5333
13  }
14 }
```

Figura 15. Evidencia de métricas que guarda evaluate.py y permiten iterar

### 5. Iteración y Optimización:

Basándose en las métricas obtenidas, se ajustaron iterativamente los hiperparámetros en `params.yaml` para encontrar la combinación que ofreciera el mejor rendimiento. Este proceso de iteración y evaluación permitió optimizar el modelo de Random Forest de manera sistemática.



## 6. Verificación de Pruebas Unitarias:

Para finalizar la evaluación de un modelo potencial, siempre se debe asegurar que las pruebas unitarias propuestas y creadas funcionan adecuadamente para cada una de las etapas. Si bien lo ideal es que se corran automáticamente en alguna etapa, de momento se proponen como pruebas aisladas que deben ejecutarse manualmente desde el terminal, por ejemplo, utilizando el comando `"python -m unittest tests/test_data_load.py"`, entre otros, garantizando así la integridad y fiabilidad del pipeline y antes de dar el modelo por definitivo.

```
(venv_proy_aigua) C:\Users\daniel.martinez\bootcamp_mlops\proyecto_ind_dme\airquality_bootcamp>python -m unittest tests/test_data_load.py
2024-05-30 14:47:13,477 - DATA_LOAD - INFO - Get dataset
2024-05-30 14:47:13,595 - DATA_LOAD - INFO - Save raw data cleaned
2024-05-30 14:47:13,718 - DATA_LOAD - INFO - Get dataset
2024-05-30 14:47:13,742 - DATA_LOAD - INFO - Save raw data cleaned
2024-05-30 14:47:13,852 - DATA_LOAD - INFO - Get dataset
2024-05-30 14:47:13,871 - DATA_LOAD - INFO - Save raw data cleaned
.
-----
Ran 3 tests in 0.512s
OK
```

Figura 16. Evidencia de "Ok" en pruebas unitarias en etapa data\_load.py del pipeline

## Beneficios del Enfoque:

- **Reproducibilidad:** La capacidad de registrar y versionar los datos y parámetros con DVC asegura que cada experimento sea reproducible. Cualquier cambio en los parámetros es fácilmente rastreable y replicable.
- **Eficiencia:** La ejecución selectiva de etapas del pipeline ahorra tiempo y recursos, permitiendo una iteración más rápida y eficiente.
- **Facilidad de Comparación:** El almacenamiento de métricas en un formato estructurado (JSON) facilita la comparación entre diferentes configuraciones, ayudando a identificar rápidamente la configuración óptima.

En conclusión de este bloque, la combinación de DVC con una estructura de pipeline modular permitió realizar experimentaciones de manera eficiente y reproducible, contribuyendo significativamente al desarrollo y optimización del modelo de Machine Learning.

### 4.3. Implementación de modelo en entorno local con contenedores

Se trató el despliegue de un modelo de Machine Learning en un entorno local mediante un contenedor de Docker, implicando varios pasos clave. A continuación, se detalla el método:

#### 1.Preparación del Entorno y Archivos Necesarios

Se creó una carpeta llamada “app” en el directorio del proyecto donde se construyeron o colocaron los siguientes archivos; cada uno de ellos necesario para crear el contenedor de Docker. Estos archivos son:

- Un Dockerfile que define cómo se construirá la imagen de Docker.
- Un archivo requirements.txt que lista todas las dependencias necesarias para ejecutar la aplicación.
- El archivo main.py que contiene el código de la aplicación FastAPI y el modelo preentrenado guardado en model.joblib.

#### 2. Creación de la Imagen de Docker

El primer paso en el proceso de despliegue fue crear una imagen de Docker a partir del Dockerfile. Este archivo especifica que la imagen se basó en Python 3.12, establece el directorio de trabajo en “/app”, copia los archivos necesarios al contenedor, instala las dependencias listadas en “requirements.txt”, expone el puerto 8000 y finalmente define el comando para ejecutar la aplicación utilizando uvicorn.

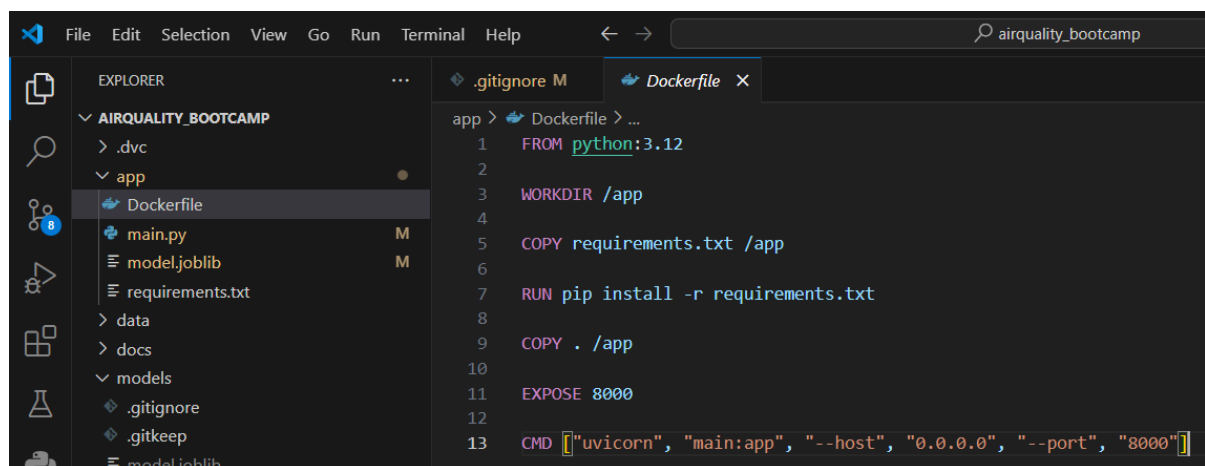


Figura 17. Evidencia Dockerfile

Para construir la imagen de Docker, se utilizó el siguiente comando en el terminal:

```
(venv_proy_airqua) C:\Users\daniel.martinez\bootcamp_mlops\proyecto_ind_dme\airquality_bootcamp>docker build -f app\Dockerfile --tag ml_model_img_dme .
[+] Building 0.0s (0/0) docker:default
[+] Building 53.7s (11/11) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 240B
=> [internal] load metadata for docker.io/library/python:3.12
=> [auth] library/python:pull token for registry-1.docker.io
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [1/5] FROM docker.io/library/python:3.12@sha256:3966b81808d864099f802080d897cef36c01550472ab3955fdd716d1c665acd6
=> [internal] load build context
=> => transferring context: 748.14MB
=> CACHED [2/5] WORKDIR /app
=> CACHED [3/5] COPY requirements.txt /app
=> CACHED [4/5] RUN pip install -r requirements.txt
=> [5/5] COPY . /app
=> exporting to image
=> => exporting layers
=> => writing image sha256:1bf84a2bd71c9af5e57e163305c5f957ebb3b8fe0a6b66219a07ca076fbff76
=> => naming to docker.io/library/ml_model_img_dme

View build details: docker-desktop://dashboard/build/default/default/ra9c5oo784xavy9qbqp7do2p

What's Next?
  View a summary of image vulnerabilities and recommendations → docker scout quickview

(venv_proy_airqua) C:\Users\daniel.martinez\bootcamp_mlops\proyecto_ind_dme\airquality_bootcamp>
```

Figura 18. Evidencia Imagen de Docker

### 3. Ejecución del Contenedor de Docker

Una vez creada la imagen, se ejecutó el contenedor a partir de esta imagen. Este paso se realizó y se aseguró como se muestra a continuación:

```
(venv_proy_airqua) C:\Users\daniel.martinez\bootcamp_mlops\proyecto_ind_dme\airquality_bootcamp\app>docker run --name contenedor_ml_app_v1 --rm --publish 8000:8000 ml_model_img_dme:v0
INFO: Started server process [1]
INFO: Waiting for application startup.
INFO: Application startup complete.
INFO: Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)
172.17.0.1:49812 - "GET / HTTP/1.1" 200 OK
172.17.0.1:49812 - "GET /docs HTTP/1.1" 200 OK
172.17.0.1:49812 - "GET /openapi.json HTTP/1.1" 200 OK
172.17.0.1:49814 - "POST /predict HTTP/1.1" 200 OK
172.17.0.1:52954 - "POST /predict HTTP/1.1" 200 OK
```

Figura 19. Evidencia Run de Contenedor Docker

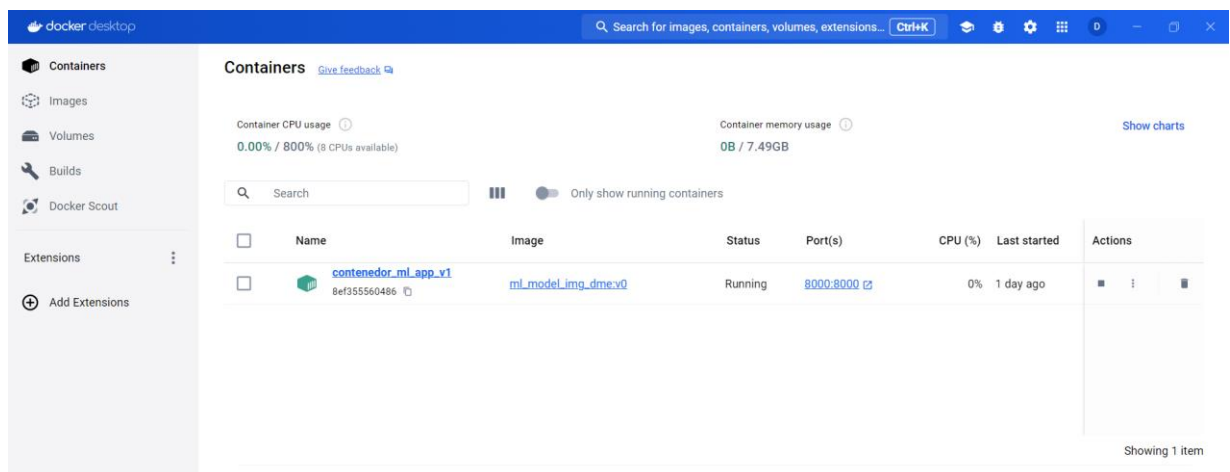


Figura 20. Comprobación de creación de contenedor desde Docker desktop

**Nota:** este comando ejecuta el contenedor y mapea el puerto 8000 del contenedor al puerto 8000 del host local, permitiendo el acceso a la aplicación a través de `http://localhost:8000`.

#### 4. Interacción con la API Desplegada y Uso del Modelo

Con el contenedor en ejecución, la API de FastAPI está disponible localmente. La interfaz interactiva de documentación de la API, generada automáticamente por FastAPI, se puede acceder en `http://localhost:8000/docs`. Esta interfaz permite realizar consultas a la API, facilitando la prueba y verificación del modelo desplegado.

Para realizar una predicción, se puede utilizar el endpoint `/predict` a través de la interfaz de documentación en “`http://localhost:8000/docs#/default/predict_predict_post`”. Aquí, se pueden ingresar los valores de las características requeridas por el modelo y obtener la predicción correspondiente.

A continuación, se muestra un ejemplo donde se genera una predicción del CO(GT) a partir de los valores de los otros gases en el aire.

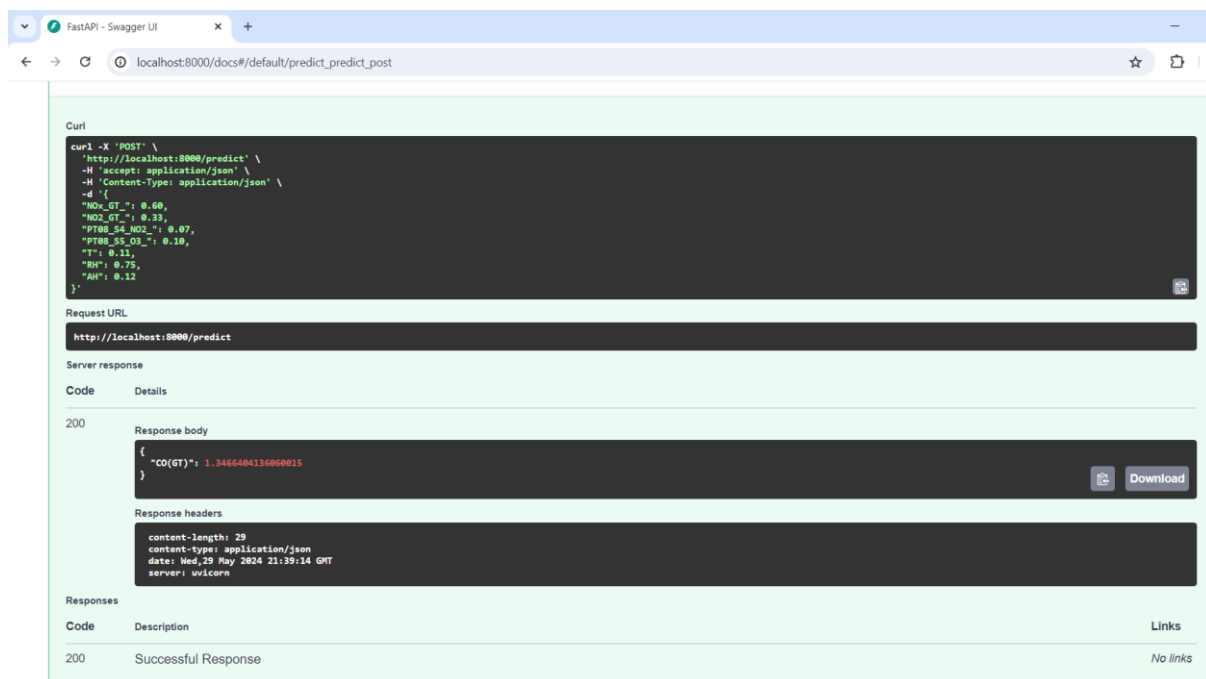


Figura 21. Evidencia Predict y comprobación despliegue exitoso

## 5. Estrategias de versionamiento y prueba de modelo

En lugar de utilizar un registro de modelos (model registry) proporcionado por DVC, se puede implementar una estrategia de prueba y versionamiento creando diferentes imágenes de Docker, cada una con sus propias etiquetas (tags) para reflejar distintas versiones y configuraciones del modelo. Este enfoque permite realizar experimentos y pruebas de manera aislada, asegurando que cada versión del modelo pueda ser evaluada y comparada de forma independiente.

Para cada experimentación y versión del modelo, se puede crear una imagen de Docker con una etiqueta específica que identifique la versión. Por ejemplo, al ajustar hiperparámetros o actualizar el modelo, se puede construir una nueva imagen con un tag distintivo:

```
docker build -t my-fastapi-app:v1 .  
docker build -t my-fastapi-app:v2 .  
docker build -t my-fastapi-app:experiment-hyperparam .
```

Estas etiquetas permiten identificar y desplegar versiones específicas del modelo, facilitando el proceso de pruebas A/B y comparación de rendimiento entre versiones.

Para probar y comparar distintas versiones del modelo, se pueden ejecutar múltiples contenedores, cada uno basado en una imagen con un tag específico. Por ejemplo:

```
docker run -d -p 8001:8000 my-fastapi-app:v1  
docker run -d -p 8002:8000 my-fastapi-app:v2
```

De esta manera, se pueden realizar predicciones y evaluar el rendimiento de cada versión del modelo de forma paralela, utilizando diferentes puertos.

### 4.4. Planificado de reentrenamiento, drift, redeploy, escalado y monitoreo

Implementar una estrategia robusta para el manejo del ciclo de vida del modelo es crucial para asegurar su rendimiento y relevancia continua. A continuación, se plantea una propuesta basada en las buenas prácticas de MLOps y los elementos disponibles en el proyecto.

**Nota:** Si bien estos elementos no fueron desarrollados (programados) al no formar parte del alcance de objetivos de este proyecto, se abordan en esta sección a manera de propuesta para llevarse a cabo en un ejercicio posterior al proyecto:

## 1. Reentrenamiento del Modelo:

**Frecuencia de Reentrenamiento:** Establecer un cronograma de reentrenamiento basado en la acumulación de nuevos datos. Por ejemplo, reentrenar el modelo mensualmente o cuando se detecten cambios significativos en los datos de entrada (features).

**Alternativa automatizada:** Utilizar DVC para automatizar el reentrenamiento mediante pipelines que se ejecuten automáticamente cuando se incorporen nuevos datos al sistema.

## 2. Detección de Drift:

**Monitoreo de Datos en Tiempo Real:** Implementar un sistema de monitoreo para detectar el drift en las características de entrada y las predicciones del modelo. Esto puede incluir el cálculo de métricas como la media de las variables y la métrica de coeficiente de determinación  $R^2$ .

**Alertas, Umbrales, Trigger-retrain:** Configurar umbrales para estas métricas (por ejemplo cuando el  $R^2$  sea menor de 0.68 y la media de al menos una variable cambie  $\pm 10\%$ ) generar una alerta que al ser superada indique la necesidad de reentrenamiento.

## 3. Escalado:

**Autoescalado:** Configurar el autoescalado de contenedores utilizando herramientas como Kubernetes para gestionar automáticamente el escalado horizontal basado en la carga del servicio. Esto garantiza que el servicio pueda manejar incrementos en el volumen de consultas sin degradar el rendimiento.

## 4. Planificación de Mantenimiento:

**Actualizaciones Regulares:** Planificar ventanas de mantenimiento regulares para actualizar dependencias, aplicar parches de seguridad y realizar tareas de limpieza en el entorno de despliegue.

**Documentación y Backups:** Mantener una documentación detallada del pipeline, configuraciones y procedimientos de despliegue. Realizar backups regulares de los modelos y datos críticos.

---

## 5. CONCLUSIONES

---

### 5.1. *Conclusión Personal*

Este trabajo de bootcamp representó el escenario ideal para desarrollar competencias que ayudaran a la implementación de modelos de machine learning en la industria. Desde mi punto de vista fue retador armar cada uno de los bloques porque involucró aprender y desarrollar cosas que las veía muy ajenas al rol de data scientist pero que sin lugar a duda me vienen a sumar mucho a mi perfil profesional. Agradezco a Wizeline y el conjunto de profesores que facilitaron el aprendizaje en los distintos módulos.

## BIBLIOGRAFÍA

- [1] J. Amat, "Correlación lineal y regresión lineal" [Online document] Jun. 2016, Available at HTTP: [https://www.cienciadedatos.net/documentos/24\\_correlacion\\_y\\_regresion\\_lineal](https://www.cienciadedatos.net/documentos/24_correlacion_y_regresion_lineal)
- [2] J. Amat, "Análisis de componentes principales (Principal Component Analysis, PCA) y t-SNE" [Online document] Jun. 2017, Available at HTTP: [https://www.cienciadedatos.net/documentos/35\\_principal\\_component\\_analysis](https://www.cienciadedatos.net/documentos/35_principal_component_analysis)
- [3] J. Amat, "Introducción a la regresión lineal múltiple" [Online document] Jul. 2016, Available at HTTP: [https://www.cienciadedatos.net/documentos/25\\_regresion\\_lineal\\_multiple.html](https://www.cienciadedatos.net/documentos/25_regresion_lineal_multiple.html)
- [4] M. Kuhn y K. Johnson, "Decision Trees and Neural Networks" en Applied Predictive Modeling, NY: Springer New York, 2013, chapter 14.

## APÉNDICE A. GLOSARIO DE VARIABLES

A continuación, se presentan las variables contenidas en el dataset estudiado:

Variable Name	Role	Type	Description	Units	Missing Values
Date	Feature	Date			no
Time	Feature	Categorical			no
CO(GT)	Feature	Integer	True hourly averaged concentration CO in mg/m <sup>3</sup> (reference analyzer)	mg/m <sup>3</sup>	no
PT08.S1(CO)	Feature	Categorical	hourly averaged sensor response (nominally CO targeted)		no
NMHC(GT)	Feature	Integer	True hourly averaged overall Non Metanic HydroCarbons concentration in microg/m <sup>3</sup> (reference analyzer)	microg/m <sup>3</sup>	no
C6H6(GT)	Feature	Continuous	True hourly averaged Benzene concentration in microg/m <sup>3</sup> (reference analyzer)	microg/m <sup>3</sup>	no
PT08.S2(NMHC)	Feature	Categorical	hourly averaged sensor response (nominally NMHC targeted)		no
NOx(GT)	Feature	Integer	True hourly averaged NOx concentration in ppb (reference analyzer)	ppb	no
PT08.S3(NOx)	Feature	Categorical	hourly averaged sensor response (nominally NOx targeted)		no
NO2(GT)	Feature	Integer	True hourly averaged NO2 concentration in microg/m <sup>3</sup> (reference analyzer)	microg/m <sup>3</sup>	no
PT08.S4(NO2)	Feature	Categorical	hourly averaged sensor response (nominally NO2 targeted)		no
PT08.S5(O3)	Feature	Categorical	hourly averaged sensor response (nominally O3 targeted)		no
T	Feature	Continuous	Temperature	°C	no
RH	Feature	Continuous	Relative Humidity	%	no
AH	Feature	Continuous	Absolute Humidity		no



