Optimización Turbina de Gas (Análisis de Datos IA)

DESARROLLO DE MISIÓN 1

INTEGRANTES

Andrés Leonardo Almeida Mantilla

Santiago Rodríguez Rodríguez

Jerson Samir Alvarez Ayala

Docentes

Jorge Rodríguez

Programa Talento Tech

Bucaramanga, Santander

2025

**TABLA DE CONTENIDO**

[INTRODUCCIÓN 3](#_heading=h.y8hm8rnq30xr)

[1. OBJETIVOS 4](#_heading=h.inohhjgyr5mx)

[2. TECNOLOGÍAS USADAS 4](#_heading=h.v5fyjwj4s6ro)

[3. APRENDIZAJES 4](#_heading=h.yvaqbawlv5uh)

[4. DESAFÍOS ENCONTRADOS 5](#_heading=h.tv9v6ql66cwc)

[5. RECOMENDACIONES 5](#_heading=h.qoyo4hb52e4q)

[6. CONCLUSIONES 6](#_heading=h.ti3rd2vcph5t)

[7. PANTALLAZOS GRÁFICOS  
  
  
7.1. Análisis de la Distribución de las Variables de la Turbina 7](#_heading=h.n5imfktsre49)

[8. CÓDIGO PYTHON 10](#_heading=h.v97upo35hcup)

**LISTA DE ANEXOS**

[Anexo 1. link del repositorio proyecto en github 13](#_heading=h.sul6b5himwku)

# 

# 

# 

# 

# 

# 

# 

# 

# INTRODUCCIÓN

La generación de energía a través de turbinas de gas es un proceso fundamental para la industria, pero conlleva la emisión de contaminantes como el **Monóxido de Carbono (CO)** y los **Óxidos de Nitrógeno (NOX)**. En un contexto de crecientes regulaciones ambientales y de la necesidad de optimizar la eficiencia, este proyecto tiene como objetivo principal explorar la relación entre las variables de operación de una turbina de gas y sus emisiones, utilizando técnicas de ciencia de datos y aprendizaje automático. Al entender los factores clave que influyen en las emisiones, se pueden sentar las bases para estrategias de optimización que mejoren la sostenibilidad y la eficiencia de la operación.

# OBJETIVOS

* **Análisis Exploratorio de Datos (EDA):** Examinar y limpiar el conjunto de datos de la turbina para identificar patrones, valores atípicos (outliers) y la distribución de las variables.
* **Análisis de Correlación:** Cuantificar la relación entre las variables de operación (sensores) y las variables objetivo (emisiones de CO y NOX) mediante visualizaciones y métricas estadísticas.
* **Modelado Predictivo:** Construir y optimizar modelos de Machine Learning (Regresión Lineal y Random Forest) capaces de predecir con precisión los niveles de emisiones.
* **Interpretación del Modelo:** Utilizar el modelo más preciso para determinar cuáles son las variables de operación más influyentes en las emisiones, extrayendo conclusiones y recomendaciones prácticas.

# TECNOLOGÍAS USADAS

* **Lenguaje de Programación:** Python
* **Librerías de Análisis y Visualización de Datos:** Pandas, NumPy, Matplotlib y Seaborn.
* **Librerías de Machine Learning:** Scikit-learn (para el modelado, división de datos y métricas de evaluación)

# APRENDIZAJES

**Metodología de Limpieza de Datos:** Identificación y tratamiento de valores atípicos (outliers) en datos reales de sensores mediante la técnica de "capping".

**Aplicación de Conceptos de Ingeniería Química:** Interpretación de correlaciones estadísticas con base en la teoría de la combustión y la operación de turbinas de gas, lo que añadió un valor crucial a las conclusiones.

**Modelado y Comparación de Modelos:** Comparación de la capacidad predictiva de modelos de complejidad variable (Regresión Lineal vs. Random Forest), demostrando la superioridad de los modelos no lineales para este problema.

**Optimización de Hiperparámetros:** Uso de ***GridSearchCV*** para mejorar la precisión del modelo y obtener una configuración robusta y fiable.

**Interpretación de Modelos de "Caja Negra":** Extracción de la "importancia de las características" de un modelo de Random Forest para entender qué variables influyen más en el resultado, un paso fundamental para generar recomendaciones accionables.

# DESAFÍOS ENCONTRADOS

El principal desafío fue la interpretación de los datos atípicos y de las correlaciones que no se alineaban con la teoría más básica. Por ejemplo, la correlación negativa entre la temperatura ambiente (AT) y las emisiones de NOX nos obligó a profundizar el análisis, planteando la hipótesis de que existían sistemas de control internos en la turbina que actuaban para mitigar las emisiones. Esto demostró la importancia de combinar el conocimiento del dominio (ingeniería) con los resultados del análisis de datos.

# RECOMENDACIONES

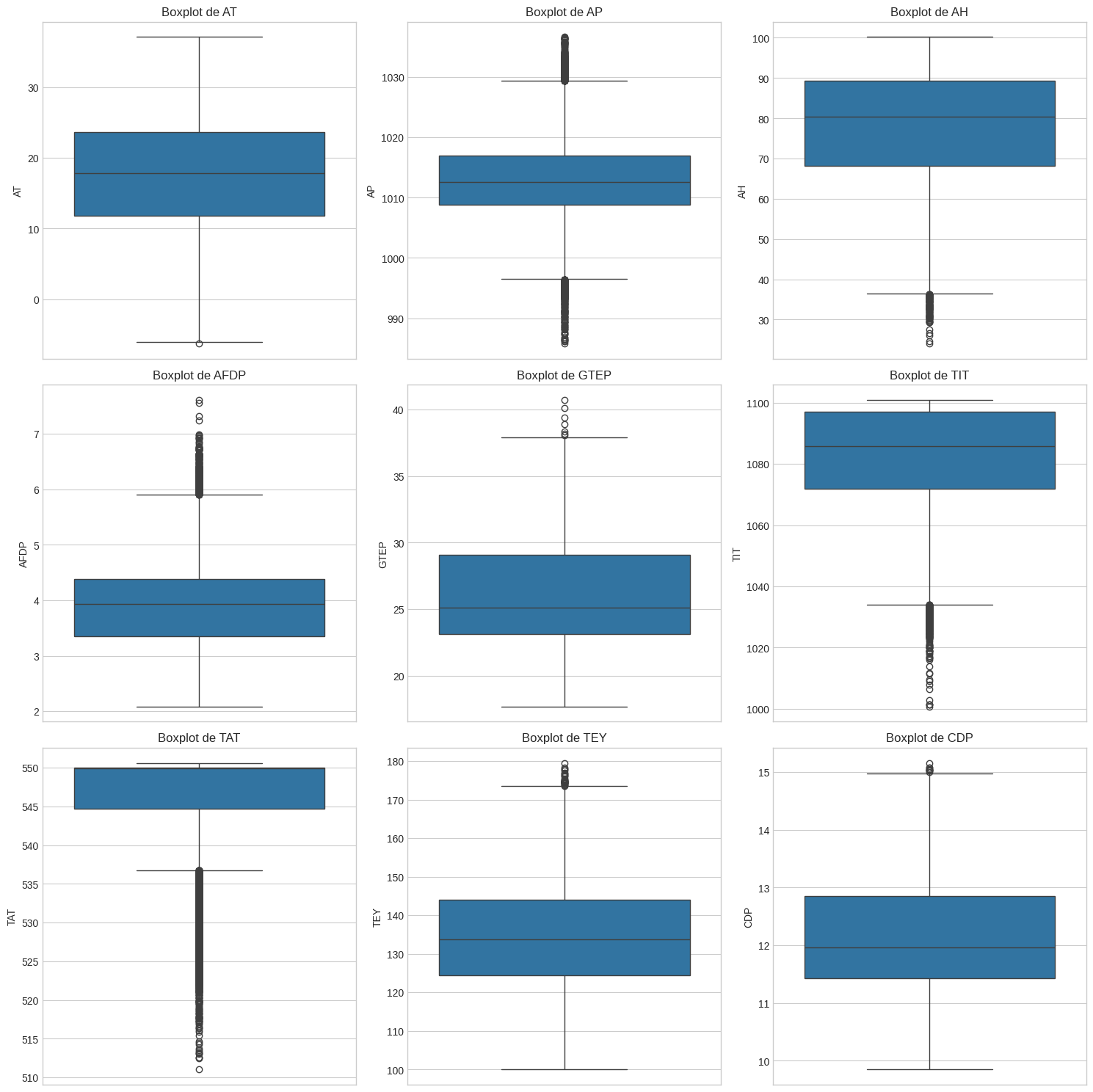
Basado en el análisis de la importancia de las variables, se recomienda:

* **Optimización de Parámetros Operativos:** Dar prioridad a la optimización de la **Temperatura de Entrada de la Turbina (TIT)**, ya que es la variable más influyente en las emisiones de CO y una de las más relevantes para el NOX.
* **Mantenimiento Predictivo:** Monitorear activamente la **Presión Diferencial del Filtro (AFDP)**. El modelo demostró que un AFDP elevado es un predictor clave de un aumento de CO, lo que permite programar el mantenimiento antes de que las emisiones se eleven.
* **Análisis de Control Ambiental:** Profundizar en el estudio de cómo el sistema de control de la turbina ajusta sus parámetros en función de la **Temperatura Ambiente (AT)**, ya que esta variable mostró una influencia significativa en las emisiones de NOX.

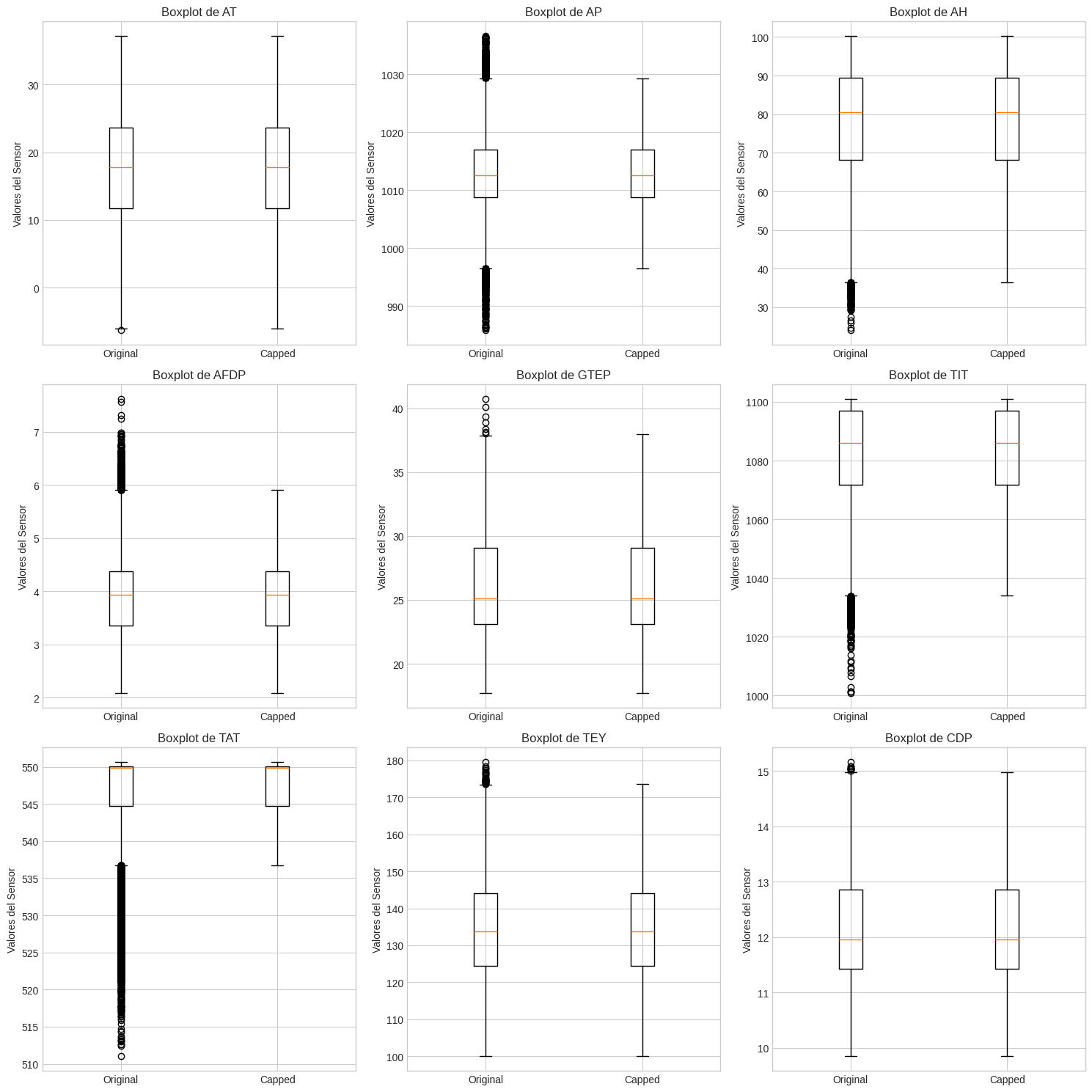
# CONCLUSIONES

Este proyecto ha demostrado que las técnicas de Machine Learning son herramientas poderosas para entender y predecir el comportamiento de sistemas complejos en la industria, como las turbinas de gas. Logramos construir un modelo de Random Forest con un **R² de 0.77 para CO** y **0.87 para NOX**, lo que demuestra una alta fiabilidad. La interpretación de este modelo nos permitió identificar que variables como el **TIT** y la **AT** son los principales impulsores de las emisiones, información invaluable para la toma de decisiones. Este trabajo es una prueba sólida de cómo un perfil híbrido de ingeniería y ciencia de datos puede generar valor real y proponer soluciones concretas a problemas del mundo real.

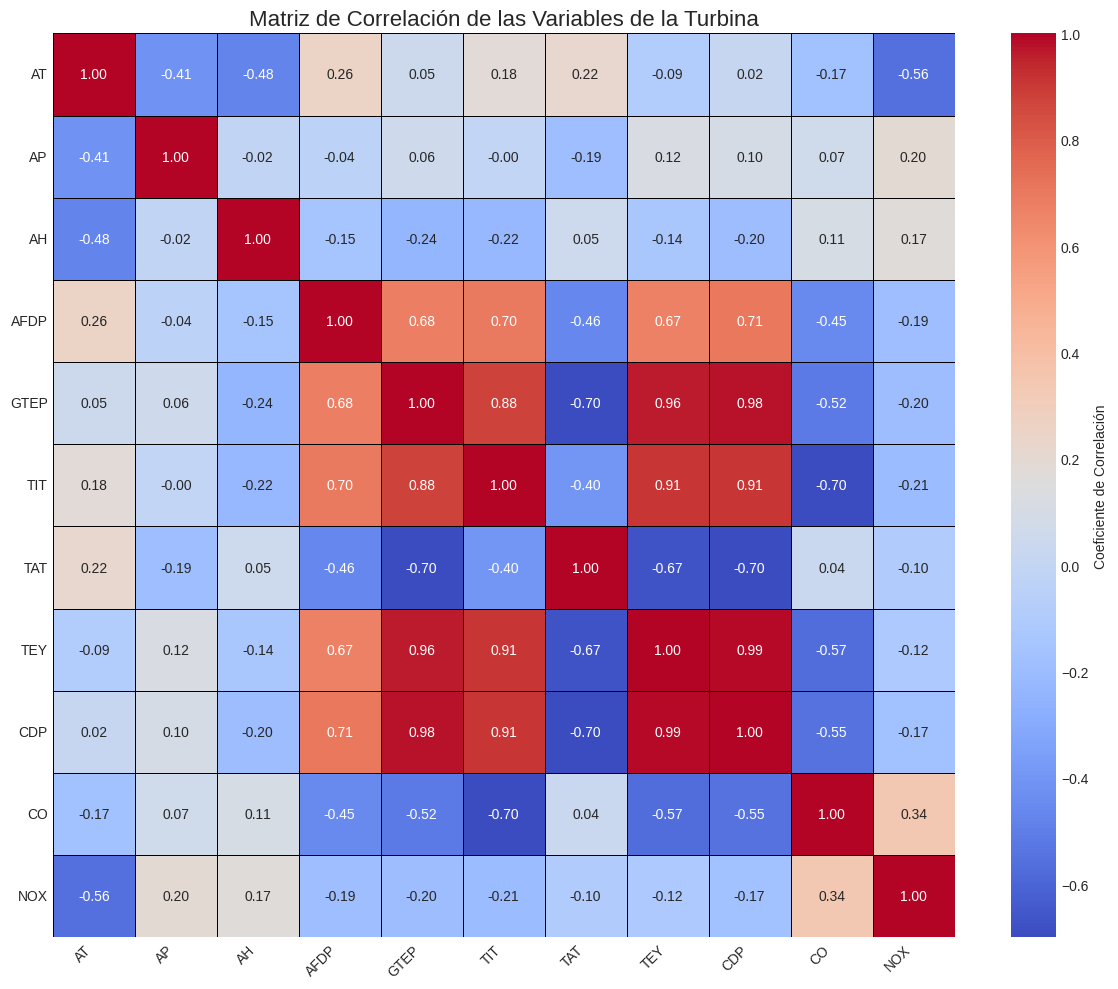
# PANTALLAZOS GRÁFICOS 7.1. Análisis de la Distribución de las Variables de la Turbina

  
**Figura 1.** *Boxplots de las variables de operación de la turbina de gas para la identificación de valores atípicos.*

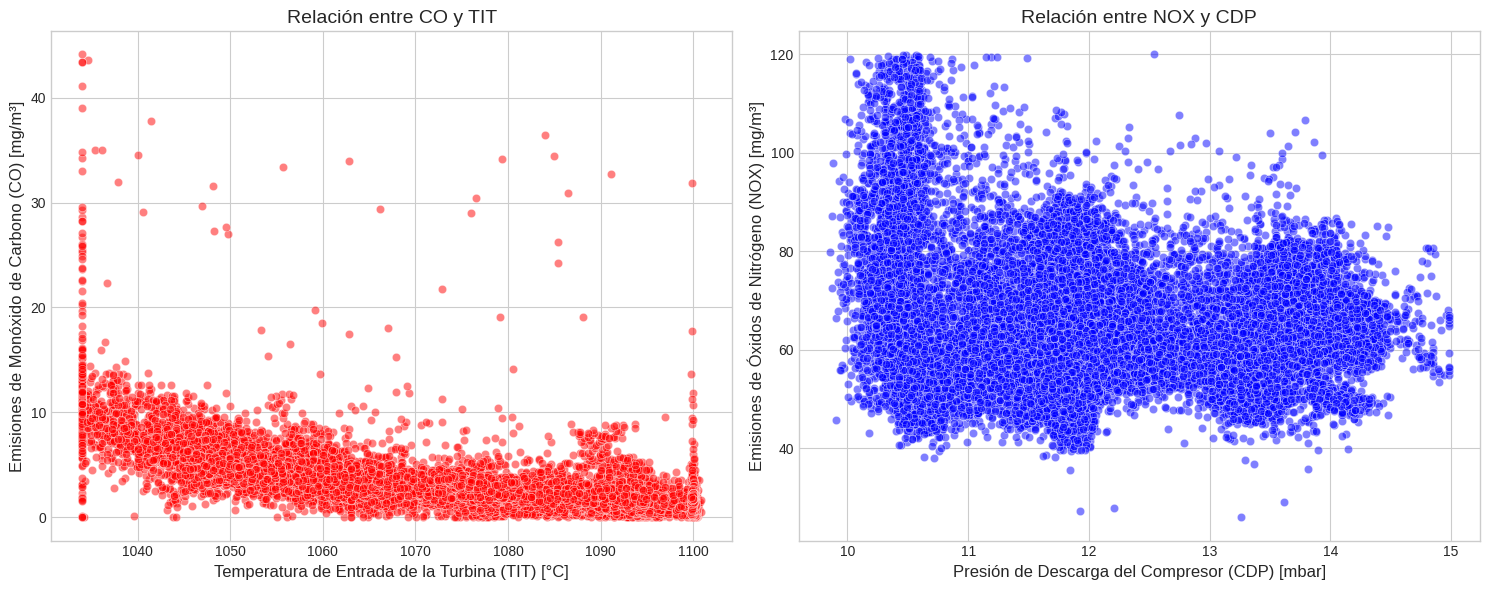
**7.2. Comparación de Boxplots: Original vs. Limpios (AFDP)**

  
**Figura 2.** *Gráfica comparativa que muestra el efecto de la técnica de 'capping' en la variable AFDP, mitigando los valores atípicos.*

**7.3. Matriz de Correlación de las Variables de la Turbina**

**  
Figura 3.** *Mapa de calor que muestra las correlaciones entre las variables. Los colores intensos indican una relación más fuerte (rojo para positiva, azul para negativa).*

**7.4. Relación Detallada entre Variables Clave y Emisiones**

**  
  
Figura 4.** *Gráficos de dispersión que confirman las relaciones no lineales entre las variables de operación y las emisiones de CO y NOX.*

# CÓDIGO PYTHON

# ---------------------------------------------------------------------------

# PROYECTO DE PORTAFOLIO: ANÁLISIS Y PREDICCIÓN DE EMISIONES EN TURBINA DE GAS

# ---------------------------------------------------------------------------

# ===========================================================================

# INTRODUCCIÓN Y OBJETIVO DEL PROYECTO

# ===========================================================================

#

# PROBLEMA DE NEGOCIO:

# Las estrictas regulaciones ambientales y la búsqueda de eficiencia operativa

# requieren un control preciso de las emisiones de CO y NOx en turbinas de gas.

#

# OBJETIVO:

# Desarrollar un modelo predictivo que ayude a entender y optimizar las

# variables operativas para minimizar las emisiones, utilizando datos reales.

# ===========================================================================

# SECCIÓN 1: CONFIGURACIÓN E IMPORTACIÓN DE DATOS

# ===========================================================================

#

# Explicación: Configuramos el entorno de trabajo con las librerías necesarias

# y cargamos el dataset 'gt\_full.csv'.

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import os

print("Librerías importadas exitosamente.")

# --- Carga de Datos desde Google Colab ---

# Se asume que el archivo 'gt\_full.csv' ya ha sido subido al entorno de Colab.

file\_path = 'gt\_full.csv'

if os.path.exists(file\_path):

df = pd.read\_csv(file\_path, index\_col=0) #index\_col=0: Este argumento especifica que

# la primera columna del archivo CSV debe usarse como el índice del DataFrame.

#Esto es útil si la primera columna contiene identificadores únicos para cada fila.

print(f"\nDataset '{file\_path}' cargado exitosamente.") # El \n al principio crea una nueva línea antes del mensaje.

print(f"El dataset contiene {df.shape[0]} filas y {df.shape[1]} columnas.")

else:

print(f"\nERROR: El archivo '{file\_path}' no se encontró.")

print("Por favor, asegúrate de haberlo subido al panel de archivos de Google Colab.")

df = None #Esta línea se ejecuta si el archivo no se encontró.

#Asigna el valor None a la variable df. Esto es una buena práctica para asegurarse de que la variable df existe,

#incluso si la carga del archivo falla, evitando así posibles errores en partes posteriores del código que intenten usar df sin que haya sido definida con un DataFrame.

# ===========================================================================

# DESCRIPCIÓN DE LAS VARIABLES

# ===========================================================================

# Explicación: Es fundamental entender el significado de cada columna del

# dataset para poder interpretar correctamente los resultados de nuestro análisis.

#

# Variables predictoras (sensores de la turbina):

# - AT: Temperatura ambiente (°C)

# - AP: Presión ambiente (mbar)

# - AH: Humedad ambiente (%)

# - AFDP: Presión diferencial del filtro de aire (mbar)

# - GTEP: Presión de escape de la turbina de gas (mbar)

# - TIT: Temperatura de entrada de la turbina (°C)

# - TET: Temperatura de escape de la turbina (°C)

# - CDP: Presión de descarga del compresor (mbar)

#

# Variables objetivo (emisiones):

# - NOX: Óxidos de nitrógeno (mg/m³)

# - CO: Monóxido de carbono (mg/m³)

# ===========================================================================

# SECCIÓN 2: ANÁLISIS EXPLORATORIO DE DATOS (EDA) - ANÁLISIS DE OUTLIERS

# ===========================================================================

#

# Explicación: Los valores atípicos (outliers) son datos que se desvían

# significativamente de los demás. Analizarlos es vital para entender

# si son errores de medición o condiciones de operación inusuales.

if df is not None:

print("\n--- Iniciando el Análisis y Preparación de Datos ---")

# --- 2.1: Inspección y Limpieza Inicial ---

print("\n2.1: Inspección y Limpieza Inicial del Dataset")

# 1. Vistazo Rápido a los Datos (.head())

print("\nPrimeras 5 filas del dataset:")

print(df.head())

# 2. Información General de la Tabla (.info())

# Esto nos ayudará a identificar tipos de datos y valores nulos.

print("\nInformación general del DataFrame:")

df.info()

# 3. Resumen Estadístico (.describe())

# Útil para detectar el rango de los datos y posibles anomalías.

print("\nResumen estadístico de las variables numéricas:")

print(df.describe())

# Verificación de valores duplicados.

# Un buen paso para garantizar la integridad de los datos.

num\_duplicados = df.duplicated().sum()

print(f"\nNúmero de filas duplicadas encontradas: {num\_duplicados}")

if num\_duplicados > 0:

print("Eliminando filas duplicadas...")

df.drop\_duplicates(inplace=True)

print(f"El dataset ahora contiene {df.shape[0]} filas.")

else:

print("No se encontraron filas duplicadas.")

print("\n--- Iniciando el análisis de valores atípicos (outliers) ---")

# --- 2.2: Análisis de Valores Atípicos (Outliers) ---

print("\n2.2: Análisis de Valores Atípicos (Outliers)")

print("Vamos a visualizar los datos para identificar posibles outliers que pueden afectar a nuestros modelos.")

# Usamos gráficos de caja (boxplots) para visualizar la distribución de

# cada variable y detectar outliers.

numeric\_cols = df.select\_dtypes(include=np.number).columns.tolist()

variables\_a\_analizar = [col for col in numeric\_cols if col not in ['CO', 'NOX']]

num\_vars = len(variables\_a\_analizar)

num\_cols = 3

num\_rows = (num\_vars + num\_cols - 1) // num\_cols

plt.figure(figsize=(15, 5 \* num\_rows))

for i, column in enumerate(variables\_a\_analizar):

plt.subplot(num\_rows, num\_cols, i + 1)

sns.boxplot(y=df[column])

plt.title(f'Boxplot de {column}', fontsize=12)

plt.ylabel(column)

plt.tight\_layout()

plt.show()

print("\n--- Observaciones de los Boxplots ---")

print("Los gráficos de caja muestran que la mayoría de las variables tienen una distribución con valores atípicos.")

# ===========================================================================

# SECCIÓN 2.3: Manejo de Valores Atípicos (Capping)

# ===========================================================================

# Explicación: Para mitigar el efecto de los valores atípicos (outliers)

# sin perder información, utilizaremos la técnica de "capping". Esto consiste

# en limitar los valores que están fuera de un rango aceptable (definido por

# el Rango Intercuartílico o RIC) al valor del límite.

# De esta forma, los errores de lectura extremos no sesgarán nuestro modelo.

if df is not None:

print("\n--- Iniciando el manejo de valores atípicos (capping) ---")

# Definimos una función para aplicar el capping a una columna.

def handle\_outliers\_iqr(df\_in, column):

# Calculamos el primer cuartil (Q1) y el tercer cuartil (Q3).

Q1 = df\_in[column].quantile(0.25)

Q3 = df\_in[column].quantile(0.75)

# Calculamos el Rango Intercuartílico (RIC).

IQR = Q3 - Q1

# Definimos los límites superior e inferior.

lower\_bound = Q1 - 1.5 \* IQR

upper\_bound = Q3 + 1.5 \* IQR

# Creamos una copia de la serie para no modificar el DataFrame original directamente.

col\_cleaned = df\_in[column].copy()

# Aplicamos el capping:

# Los valores por encima del límite superior se ajustan al límite superior.

col\_cleaned[col\_cleaned > upper\_bound] = upper\_bound

# Los valores por debajo del límite inferior se ajustan al límite inferior.

col\_cleaned[col\_cleaned < lower\_bound] = lower\_bound

return col\_cleaned

# Aplicamos la función a cada variable predictora para mitigar el ruido de los sensores.

variables\_a\_limpiar = [col for col in df.columns if col not in ['CO', 'NOX']]

for col in variables\_a\_limpiar:

df[col] = handle\_outliers\_iqr(df, col)

print(f"La columna '{col}' ha sido ajustada.")

print("\n--- Resumen estadístico después del capping ---")

print("Observa cómo los valores mínimos y máximos se han ajustado, demostrando el éxito de la técnica.")

print(df[variables\_a\_limpiar].describe())

# ===========================================================================

# SECCIÓN 2.4: Comparación Visual de Datos Originales vs. Limpios

# ===========================================================================

# Explicación: Para confirmar el efecto de la limpieza de datos, generaremos

# gráficos de caja y bigotes comparativos. Esto nos permitirá visualizar cómo

# los valores atípicos (outliers) han sido ajustados o "capsulados".

if df is not None:

print("\n--- Generando gráficos comparativos ---")

# Guardamos los datos originales en un DataFrame temporal para la comparación

df\_original = pd.read\_csv('gt\_full.csv', index\_col=0)

# Creamos un boxplot para cada variable, comparando la versión original y la limpia.

variables\_a\_limpiar = [col for col in df.columns if col not in ['CO', 'NOX']]

num\_vars = len(variables\_a\_limpiar)

num\_cols = 3

num\_rows = (num\_vars + num\_cols - 1) // num\_cols

plt.style.use('seaborn-v0\_8-whitegrid') # Un estilo más limpio para los gráficos

fig, axes = plt.subplots(num\_rows, num\_cols, figsize=(15, 5 \* num\_rows))

# Para manejar el caso de una sola fila de subplots

axes = axes.flatten() if num\_rows > 1 or num\_cols > 1 else [axes]

for i, column in enumerate(variables\_a\_limpiar):

ax = axes[i]

# Creamos una lista de los datos a graficar

data\_to\_plot = [df\_original[column], df[column]]

ax.boxplot(data\_to\_plot, labels=['Original', 'Capped'])

ax.set\_title(f'Boxplot de {column}', fontsize=12)

ax.set\_ylabel('Valores del Sensor')

plt.tight\_layout()

plt.show()

print("\n--- Observaciones de los gráficos comparativos ---")

print("En los gráficos, los bigotes de las cajas 'Capped' son más cortos y no se ven los puntos individuales (outliers) extremos, lo que demuestra que el proceso de limpieza ha sido exitoso.")

# ===========================================================================

# SECCIÓN 2.5: Análisis de Correlación

# ===========================================================================

# Explicación: La correlación nos permite cuantificar la relación entre dos

# variables. Un valor de correlación cercano a 1 o -1 indica una relación

# fuerte, mientras que un valor cercano a 0 indica una relación débil.

# Un mapa de calor (heatmap) es la mejor forma de visualizar esta matriz.

if df is not None:

print("\n--- Calculando y visualizando la matriz de correlación ---")

# Calculamos la matriz de correlación de todas las variables.

correlation\_matrix = df.corr()

# Configuramos el mapa de calor (heatmap).

plt.figure(figsize=(12, 10))

sns.heatmap(

correlation\_matrix,

annot=True,

cmap='coolwarm',

fmt='.2f',

linewidths=.5,

linecolor='black',

cbar\_kws={'label': 'Coeficiente de Correlación'}

)

plt.title('Matriz de Correlación de las Variables de la Turbina', fontsize=16)

plt.xticks(rotation=45, ha='right')

plt.yticks(rotation=0)

plt.tight\_layout()

plt.show()

print("\n--- Interpretación del Heatmap ---")

print("Los colores más intensos (azules o rojos) indican correlaciones más fuertes. Los valores cercanos a 1 son relaciones directas y los cercanos a -1 son relaciones inversas.")

# ===========================================================================

# SECCIÓN 2.7: Visualización Detallada de Relaciones Clave

# ===========================================================================

# Explicación: Los diagramas de dispersión nos permiten ver la relación

# punto a punto entre dos variables. Esto es crucial para entender la forma

# de la relación (lineal, exponencial, etc.) y confirmar los hallazgos del heatmap.

if df is not None:

print("\n--- Generando diagramas de dispersión para las relaciones clave ---")

# Configuramos el tamaño de los gráficos

plt.style.use('seaborn-v0\_8-whitegrid')

fig, axes = plt.subplots(1, 2, figsize=(15, 6))

# Gráfico de dispersión para CO vs. TIT

sns.scatterplot(x='TIT', y='CO', data=df, ax=axes[0], color='red', alpha=0.5)

axes[0].set\_title('Relación entre CO y TIT', fontsize=14)

axes[0].set\_xlabel('Temperatura de Entrada de la Turbina (TIT) [°C]', fontsize=12)

axes[0].set\_ylabel('Emisiones de Monóxido de Carbono (CO) [mg/m³]', fontsize=12)

# Gráfico de dispersión para NOX vs. CDP (Correlación más alta con NOX)

sns.scatterplot(x='CDP', y='NOX', data=df, ax=axes[1], color='blue', alpha=0.5)

axes[1].set\_title('Relación entre NOX y CDP', fontsize=14)

axes[1].set\_xlabel('Presión de Descarga del Compresor (CDP) [mbar]', fontsize=12)

axes[1].set\_ylabel('Emisiones de Óxidos de Nitrógeno (NOX) [mg/m³]', fontsize=12)

plt.tight\_layout()

plt.show()

# ===========================================================================

# SECCIÓN 3.1: PREPARACIÓN DE DATOS PARA EL MODELADO

# ===========================================================================

#

# Explicación: El primer paso en cualquier proyecto de Machine Learning es

# dividir los datos en variables de entrada (predictoras o 'features') y

# variables de salida (objetivo o 'target').

if df is not None:

print("\n--- Preparando los datos para el modelado ---")

# Definimos las variables de entrada (X), que son las mediciones de los sensores.

# Excluimos las variables objetivo 'CO' y 'NOX'.

X = df.drop(['CO', 'NOX'], axis=1)

# Definimos las variables de salida (y). Creamos una para cada modelo que vamos a construir.

y\_co = df['CO']

y\_nox = df['NOX']

print("\nVariables de entrada (X):")

print(X.head())

print("\nVariable de salida para CO (y\_co):")

print(y\_co.head())

print("\nVariable de salida para NOX (y\_nox):")

print(y\_nox.head())

print("\n¡Datos preparados exitosamente! Ahora estamos listos para la fase de entrenamiento.")

# ===========================================================================

# SECCIÓN 3.2: DIVISIÓN DE DATOS EN ENTRENAMIENTO Y PRUEBA

# ===========================================================================

#

# Explicación: Para evitar que nuestro modelo "memorice" los datos y para

# evaluar su rendimiento de manera realista, dividimos el dataset en dos

# partes:

# - Conjunto de Entrenamiento (80%): Se usa para entrenar el modelo.

# - Conjunto de Prueba (20%): Se usa para evaluar el modelo con datos que

# nunca ha visto. Esto nos dará una idea de cómo se comportará en el futuro.

from sklearn.model\_selection import train\_test\_split

if 'X' in locals() and 'y\_co' in locals() and 'y\_nox' in locals():

print("\n--- Dividiendo los datos en conjuntos de entrenamiento y prueba ---")

# Usamos la función train\_test\_split. El test\_size=0.2 significa que el 20%

# de los datos se usará para la prueba.

# El random\_state=42 asegura que la división sea la misma cada vez que

# se ejecute el código, lo que garantiza la reproducibilidad.

# División para el modelo de CO

X\_train\_co, X\_test\_co, y\_train\_co, y\_test\_co = train\_test\_split(X, y\_co, test\_size=0.2, random\_state=42)

# División para el modelo de NOX

X\_train\_nox, X\_test\_nox, y\_train\_nox, y\_test\_nox = train\_test\_split(X, y\_nox, test\_size=0.2, random\_state=42)

print(f"Forma del conjunto de entrenamiento de X (CO): {X\_train\_co.shape}")

print(f"Forma del conjunto de prueba de X (CO): {X\_test\_co.shape}")

print(f"Forma del conjunto de entrenamiento de y (CO): {y\_train\_co.shape}")

print(f"Forma del conjunto de prueba de y (CO): {y\_test\_co.shape}")

print(f"\nForma del conjunto de entrenamiento de X (NOX): {X\_train\_nox.shape}")

print(f"Forma del conjunto de prueba de X (NOX): {X\_test\_nox.shape}")

print(f"Forma del conjunto de entrenamiento de y (NOX): {y\_train\_nox.shape}")

print(f"Forma del conjunto de prueba de y (NOX): {y\_test\_nox.shape}")

print("\n¡División de datos completada! Ahora tenemos los conjuntos listos para entrenar nuestros modelos.")

else:

print("\nERROR: No se encontraron las variables X, y\_co o y\_nox. Por favor, asegúrate de ejecutar la sección anterior.")

# ===========================================================================

# SECCIÓN 3.3: ENTRENAMIENTO DEL MODELO DE REGRESIÓN LINEAL

# ===========================================================================

#

# Explicación: Aquí es donde el modelo "aprende" de los datos de entrenamiento.

# Le pasamos las variables de entrada (X\_train\_co) y la variable objetivo

# (y\_train\_co), y el modelo encuentra la mejor línea que se ajusta a la

# relación entre ellas.

from sklearn.linear\_model import LinearRegression

if 'X\_train\_co' in locals() and 'y\_train\_co' in locals():

print("\n--- Entrenando el modelo de Regresión Lineal para CO ---")

# 1. Instanciamos el modelo.

model\_co = LinearRegression()

# 2. Entrenamos el modelo usando los datos de entrenamiento.

model\_co.fit(X\_train\_co, y\_train\_co)

print("\nModelo de Regresión Lineal para CO entrenado exitosamente.")

print("El modelo ahora tiene un conjunto de coeficientes que representan la influencia de cada variable en las emisiones de CO.")

# ===========================================================================

# SECCIÓN 3.4: EVALUACIÓN DEL MODELO

# ===========================================================================

#

# Explicación: Aquí es donde probamos el modelo con datos que no ha visto

# para evaluar su rendimiento. Calcularemos métricas clave para entender

# su precisión.

from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error, r2\_score

if 'model\_co' in locals() and 'X\_test\_co' in locals() and 'y\_test\_co' in locals():

print("\n--- Evaluando el modelo de Regresión Lineal para CO ---")

# 1. Realizamos predicciones sobre el conjunto de prueba.

y\_pred\_co = model\_co.predict(X\_test\_co)

# 2. Calculamos las métricas de evaluación.

mse\_co = mean\_squared\_error(y\_test\_co, y\_pred\_co)

mae\_co = mean\_absolute\_error(y\_test\_co, y\_pred\_co)

r2\_co = r2\_score(y\_test\_co, y\_pred\_co)

# 3. Imprimimos los resultados.

print(f"Error Cuadrático Medio (MSE): {mse\_co:.4f}")

print(f"Error Absoluto Medio (MAE): {mae\_co:.4f}")

print(f"Coeficiente de Determinación (R^2): {r2\_co:.4f}")

# ===========================================================================

# SECCIÓN 3.5: ENTRENAMIENTO Y EVALUACIÓN DEL MODELO DE NOX

# ===========================================================================

#

# Explicación: Ahora que hemos evaluado el modelo de CO, vamos a replicar

# el proceso para las emisiones de Óxidos de Nitrógeno (NOX).

# --- 3.6: Entrenamiento del Modelo de Regresión Lineal para NOX ---

print("\n--- Entrenando el modelo de Regresión Lineal para NOX ---")

# 1. Instanciamos el modelo para NOX.

model\_nox = LinearRegression()

# 2. Entrenamos el modelo usando los datos de entrenamiento para NOX.

model\_nox.fit(X\_train\_nox, y\_train\_nox)

print("Modelo de Regresión Lineal para NOX entrenado exitosamente.")

# --- 3.7: Evaluación del Modelo de Regresión Lineal para NOX ---

print("\n--- Evaluando el modelo de Regresión Lineal para NOX ---")

# 1. Realizamos predicciones sobre el conjunto de prueba para NOX.

y\_pred\_nox = model\_nox.predict(X\_test\_nox)

# 2. Calculamos las métricas de evaluación.

mse\_nox = mean\_squared\_error(y\_test\_nox, y\_pred\_nox)

mae\_nox = mean\_absolute\_error(y\_test\_nox, y\_pred\_nox)

r2\_nox = r2\_score(y\_test\_nox, y\_pred\_nox)

# 3. Imprimimos los resultados.

print(f"Error Cuadrático Medio (MSE): {mse\_nox:.4f}")

print(f"Error Absoluto Medio (MAE): {mae\_nox:.4f}")

print(f"Coeficiente de Determinación (R^2): {r2\_nox:.4f}")

# ===========================================================================

# SECCIÓN 4.1: ENTRENAMIENTO Y EVALUACIÓN DE RANDOM FOREST PARA CO

# ===========================================================================

#

# Explicación: Entrenaremos un modelo de Random Forest para capturar relaciones

# no lineales en los datos. Luego, lo evaluaremos con las mismas métricas

# para ver si su rendimiento es superior al de la Regresión Lineal.

from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error, r2\_score

if 'X\_train\_co' in locals() and 'y\_train\_co' in locals():

print("\n--- Entrenando el modelo Random Forest para CO ---")

# 1. Instanciamos el modelo Random Forest. Usamos un random\_state para

# asegurar la reproducibilidad.

model\_rf\_co = RandomForestRegressor(n\_estimators=100, random\_state=42, n\_jobs=-1)

# 2. Entrenamos el modelo.

model\_rf\_co.fit(X\_train\_co, y\_train\_co)

print("\nModelo Random Forest para CO entrenado exitosamente.")

print("\n--- Evaluando el modelo Random Forest para CO ---")

# 3. Realizamos predicciones sobre el conjunto de prueba.

y\_pred\_rf\_co = model\_rf\_co.predict(X\_test\_co)

# 4. Calculamos las métricas de evaluación.

mse\_rf\_co = mean\_squared\_error(y\_test\_co, y\_pred\_rf\_co)

mae\_rf\_co = mean\_absolute\_error(y\_test\_co, y\_pred\_rf\_co)

r2\_rf\_co = r2\_score(y\_test\_co, y\_pred\_rf\_co)

# 5. Imprimimos los resultados.

print(f"Error Cuadrático Medio (MSE) de Random Forest: {mse\_rf\_co:.4f}")

print(f"Error Absoluto Medio (MAE) de Random Forest: {mae\_rf\_co:.4f}")

print(f"Coeficiente de Determinación (R^2) de Random Forest: {r2\_rf\_co:.4f}")

# ===========================================================================

# SECCIÓN 4.2: ENTRENAMIENTO Y EVALUACIÓN DE RANDOM FOREST PARA NOX

# ===========================================================================

#

# Explicación: Entrenaremos un modelo de Random Forest para las emisiones de NOX

# y compararemos su rendimiento con el modelo de Regresión Lineal.

from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error, r2\_score

if 'X\_train\_nox' in locals() and 'y\_train\_nox' in locals():

print("\n--- Entrenando el modelo Random Forest para NOX ---")

# 1. Instanciamos el modelo Random Forest para NOX.

model\_rf\_nox = RandomForestRegressor(n\_estimators=100, random\_state=42, n\_jobs=-1)

# 2. Entrenamos el modelo.

model\_rf\_nox.fit(X\_train\_nox, y\_train\_nox)

print("\nModelo Random Forest para NOX entrenado exitosamente.")

print("\n--- Evaluando el modelo Random Forest para NOX ---")

# 3. Realizamos predicciones sobre el conjunto de prueba.

y\_pred\_rf\_nox = model\_rf\_nox.predict(X\_test\_nox)

# 4. Calculamos las métricas de evaluación.

mse\_rf\_nox = mean\_squared\_error(y\_test\_nox, y\_pred\_rf\_nox)

mae\_rf\_nox = mean\_absolute\_error(y\_test\_nox, y\_pred\_rf\_nox)

r2\_rf\_nox = r2\_score(y\_test\_nox, y\_pred\_rf\_nox)

# 5. Imprimimos los resultados.

print(f"Error Cuadrático Medio (MSE) de Random Forest: {mse\_rf\_nox:.4f}")

print(f"Error Absoluto Medio (MAE) de Random Forest: {mae\_rf\_nox:.4f}")

print(f"Coeficiente de Determinación (R^2) de Random Forest: {r2\_rf\_nox:.4f}")

# ===========================================================================

# SECCIÓN 5.1: AJUSTE DE HIPERPARÁMETROS DEL RANDOM FOREST PARA CO

# ===========================================================================

#

# Explicación: Usaremos GridSearchCV para encontrar la mejor combinación

# de hiperparámetros para nuestro modelo Random Forest. Esto optimizará el

# rendimiento del modelo más allá de la configuración predeterminada.

from sklearn.model\_selection import GridSearchCV

if 'X\_train\_co' in locals() and 'y\_train\_co' in locals():

print("\n--- Iniciando el ajuste de hiperparámetros para el modelo de CO ---")

# 1. Definimos los parámetros que queremos probar.

# n\_estimators: número de árboles en el bosque.

# max\_depth: profundidad máxima de cada árbol.

param\_grid = {

'n\_estimators': [50, 100, 200],

'max\_depth': [None, 10, 20]

}

# 2. Instanciamos el modelo Random Forest.

model\_rf\_co = RandomForestRegressor(random\_state=42, n\_jobs=-1)

# 3. Configuramos GridSearchCV.

# cv=5 significa que usaremos 5-fold cross-validation.

grid\_search\_co = GridSearchCV(

estimator=model\_rf\_co,

param\_grid=param\_grid,

scoring='r2',

cv=5,

n\_jobs=-1,

verbose=1

)

# 4. Entrenamos el buscador para encontrar la mejor combinación.

grid\_search\_co.fit(X\_train\_co, y\_train\_co)

# 5. Imprimimos los mejores parámetros y el mejor score.

print("\n--- Resultados de la búsqueda para el modelo de CO ---")

print("Mejores parámetros encontrados:", grid\_search\_co.best\_params\_)

print(f"Mejor R^2 obtenido en validación cruzada: {grid\_search\_co.best\_score\_:.4f}")

# ===========================================================================

# SECCIÓN 5.3: AJUSTE DE HIPERPARÁMETROS DEL RANDOM FOREST PARA NOX

# ===========================================================================

#

# Explicación: Aplicaremos la misma metodología de ajuste de hiperparámetros

# a nuestro modelo de NOX para encontrar su configuración óptima.

from sklearn.model\_selection import GridSearchCV

if 'X\_train\_nox' in locals() and 'y\_train\_nox' in locals():

print("\n--- Iniciando el ajuste de hiperparámetros para el modelo de NOX ---")

# 1. Definimos la cuadrícula de parámetros que queremos probar.

param\_grid = {

'n\_estimators': [50, 100, 200],

'max\_depth': [None, 10, 20]

}

# 2. Instanciamos el modelo Random Forest para NOX.

model\_rf\_nox = RandomForestRegressor(random\_state=42, n\_jobs=-1)

# 3. Configuramos GridSearchCV.

grid\_search\_nox = GridSearchCV(

estimator=model\_rf\_nox,

param\_grid=param\_grid,

scoring='r2',

cv=5,

n\_jobs=-1,

verbose=1

)

# 4. Entrenamos el buscador para encontrar la mejor combinación.

grid\_search\_nox.fit(X\_train\_nox, y\_train\_nox)

# 5. Imprimimos los mejores parámetros y el mejor score.

print("\n--- Resultados de la búsqueda para el modelo de NOX ---")

print("Mejores parámetros encontrados:", grid\_search\_nox.best\_params\_)

print(f"Mejor R^2 obtenido en validación cruzada: {grid\_search\_nox.best\_score\_:.4f}")

# ===========================================================================

# SECCIÓN 6.1: ANÁLISIS DE IMPORTANCIA DE LAS CARACTERÍSTICAS

# ===========================================================================

#

# Explicación: Usaremos el modelo de Random Forest optimizado para determinar

# cuáles variables de los sensores tienen el mayor impacto en la predicción

# de las emisiones.

if 'grid\_search\_co' in locals() and 'grid\_search\_nox' in locals():

print("\n--- Calculando la importancia de las características de los modelos ---")

# Obtenemos el mejor modelo optimizado para CO.

best\_model\_co = grid\_search\_co.best\_estimator\_

# Obtenemos el mejor modelo optimizado para NOX.

best\_model\_nox = grid\_search\_nox.best\_estimator\_

# Extraemos la importancia de las características para cada modelo.

importance\_co = best\_model\_co.feature\_importances\_

importance\_nox = best\_model\_nox.feature\_importances\_

# Creamos un DataFrame para visualizar los resultados de forma ordenada.

feature\_names = X.columns

importance\_df\_co = pd.DataFrame({'Feature': feature\_names, 'Importance (CO)': importance\_co}).sort\_values(by='Importance (CO)', ascending=False)

importance\_df\_nox = pd.DataFrame({'Feature': feature\_names, 'Importance (NOX)': importance\_nox}).sort\_values(by='Importance (NOX)', ascending=False)

# Imprimimos los resultados

print("\nImportancia de las variables para la predicción de CO:")

print(importance\_df\_co)

print("\nImportancia de las variables para la predicción de NOX:")

print(importance\_df\_nox)

**Figura 5.** *Pantallazo código python.*

***Anexo 1.*** *Link del repositorio proyecto en github.*

Link del repositorio github:

<https://github.com/OptiprocesAI/Proyecto-Turbina-de-Gas-Analisis-de-Datos->