SAMSUNG INNOVATION CAMPUS

PROYECTO DE INTELIGENCIA ARTIFICIAL APLICADA A LA CONTAMINACIÓN DEL AIRE

Integrantes:

- Dante Sánchez Rosales
- Miguel Venegas Rocha
- Fernando Felipe Escutia

04/Abril/2025

INTRODUCCIÓN

En el presente proyecto, a partir de los conocimientos adquiridos de inteligencia artificial se busca ayudar al sector medioambiental y de salud de las personas a partir de conocer la calidad del aire la cual presenta varios factores a considerar por lo que a través de modelos de regresión, clasificación y agrupamiento se busca conocer a detalle información valiosa de este problema, así como el generar predicciones que nos permitan saber que tan buena es la calidad del aire , dependiendo de distintas concentraciones que contiene el aire tales como los PM10 y PM2.5. Para esto se utilizaron dos conjuntos de datos, uno para el entrenamiento de nuestros modelos y el otro como entrada para la predicción de datos nuevos.

Como resultado de lo anterior, lo que se obtuvo fue la determinación de los compuestos que más influencia tienen en la proliferación de partículas sólidas o líquidas que que tienen un diámetro de 2.5 micrómetros o menos (PM2.5), las cuales son las que mayor amenaza presentan en la salud de las personas. De igual manera se logró generar una predicción de la calidad del aire a partir de los datos de composición del aire, así como por último un modelo para agrupar las distintas características en el aire que influyen en la contaminación del aire.

El problema de la contaminación en el aire

La contaminación del aire es una gran amenaza para la salud humana y para el ambiente la cual ha visto un crecimiento espontaneo por la llegada del nuevo milenio. Según información de la ONU del 2021 hubo un aproximado de 8.1 millones de muertes las cuales se vinculan a la contaminación del aire, incluyendo infantes menores a 5 años los cuales se aproximan a la cantidad de 700,000 fallecimientos.

De la cifra reconocida por la ONU se estiman que 7.8 millones de los fallecimientos se atribuyen a la partícula PM2,5, micropartículas de 2,5 micrómetros de diámetro las cuales pueden permanecer en los pulmones por su tamaño e ingresar al torrente sanguíneo, afectando a los sistemas orgánicos y aumentando la probabilidad de contraer enfermedades no transmisibles, siendo provenientes de la quema de combustibles fósiles, biomasa o contaminantes del hogar.

La población más vulnerable suele ser la ubicada en países en desarrollo ya que cuentan con pocas o nulas tecnologías limpias.

Un reto al querer comprender y predecir el nivel de la contaminación del aire es su naturaleza dinámica, ya que depende de un grupo de factores y variables como la cantidad de viento, la humedad, la temperatura o las emisiones que industriales, del tráfico vehicular o las condiciones geográficas.

Por estos retos es que establecer una estación de monitoreo es extremadamente cara y en las escasas que hay no pueden transmitir la información en tiempo real en áreas remotas o fuera de áreas muy específicas

Tratamiento de los Datos

1. Se realizaron dos procesos principales de preprocesamiento:
   - Selección y limpieza de variables:
   - Se definió PM2.5 como la variable dependiente.
   - Se eliminaron columnas irrelevantes para los modelos predictivos.
   - Se removieron datos negativos o valores extremos.}
   - Se acomodaron las dos bases de datos para que tuvieran los mismos nombres y orden
2. Normalización de los datos:
   - Aplicada principalmente para la Red Neuronal, mejorando el rendimiento del modelo.

Predicción

- Ridge Regression fue empleada debido a la alta colinealidad entre variables del dataset:
- Obtuvo un R^2 superior a 0.95, demostrando un excelente desempeño en la predicción de PM2.5.
- XGBoost superó a Random Forest en tareas de clasificación:
- Accuracy: 92.5% en la predicción de niveles de contaminación.

Clasificación (Redes Neuronales)

- Se diseñó una red neuronal con la siguiente arquitectura:
- 3 capas densas (128 y 64 neuronas), acompañadas de Dropout y Batch Normalization.
- Se aplicó regularización L2 (λ=0.001) para evitar overfitting.
- Se utilizó Early Stopping para validar el modelo.
- Resultados:
- Accuracy: 99% en datos de validación.
- Pérdida de entrenamiento y validación estables, sin indicios de sobreajuste.

Agrupamiento (Clustering)

- Se compararon dos métodos: K-Means y DBSCAN, siendo K-Means el más efectivo para identificar patrones de contaminación.

Métrica	K-Means	DBSCAN
Silhouette	0.36	0.28
N° de Clusters	5	3

- Argumentos.
  - Incluir algoritmos de machine learning junto a los sensores IoT(Internet de las cosas)
  - Búsqueda de una recopilación de datos en tiempo real de partículas contaminantes como PM2,5, CO2, NO2, ozono y las variables inusuales
  - Predicción de episodios críticos con el uso de redes neuronales recurrentes (RNN) para analizar patrones históricos y meteorológicos para predecir picos de contaminación en horas o días con antelación
  - Detectar fuentes de emisiones ayudado con técnicas de clustering para ubicar fabricas o rutas de tráfico donde se detecte una gran emisión
  - Habilitar un sistema de alerta cuando se detecte una gran concentración de partículas dañinas

❖ Información de la base de datos.

Air Quality and Pollution Assessment:

	Temperature	Humidity	PM2.5	PM10	NO2	SO2	CO	\
0	29.8	59.1	5.2	17.9	18.9	9.2	1.72	
1	28.3	75.6	2.3	12.2	30.8	9.7	1.64	
2	23.1	74.7	26.7	33.8	24.4	12.6	1.63	
3	27.1	39.1	6.1	6.3	13.5	5.3	1.15	
4	26.5	70.7	6.9	16.0	21.9	5.6	1.01	

	Proximity_to_Industrial_Areas	Population_Density	Air Quality
0	6.3	319	Moderate
1	6.0	611	Moderate
2	5.2	619	Moderate
3	11.1	551	Good
4	12.7	303	Good

Variables que contiene el conjunto de datos

Tipos de datos

#	Column	Non-Null Count	Dtype
0	Temperature	5000 non-null	float64
1	Humidity	5000 non-null	float64
2	PM2.5	5000 non-null	float64
3	PM10	5000 non-null	float64
4	NO2	5000 non-null	float64
5	SO2	5000 non-null	float64
6	CO	5000 non-null	float64
7	Proximity_to_Industrial_Areas	5000 non-null	float64
8	Population_Density	5000 non-null	int64
9	Air Quality	5000 non-null	object

	Temperature	Humidity	PM2.5	PM10	NO2	\
count	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	
mean	30.029020	70.056120	20.142140	30.218360	26.412100	
std	6.720661	15.863577	24.554546	27.349199	8.895356	
min	13.400000	36.000000	0.000000	-0.200000	7.400000	
25%	25.100000	58.300000	4.600000	12.300000	20.100000	
50%	29.000000	69.800000	12.000000	21.700000	25.300000	
...						
25%	381.000000					
50%	494.000000					
75%	600.000000					
max	957.000000					

Descripciones

- No contiene ningun dato nulo.
- Remover datos no relevantes, y datos negativos, así como datos extremos o sin sentido dentro del conjunto de datos.
- En tanto PM10 como en S02 habia datos negativos, lo que se hizo fue eliminar estos valores. Al no tener coherencia datos de meterologia en negativo.

```
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   Temperature                   4181 non-null   float64
 1   Humidity                      4181 non-null   float64
 2   PM2.5                         4181 non-null   float64
 3   PM10                          4181 non-null   float64
 4   NO2                           4181 non-null   float64
 5   SO2                           4181 non-null   float64
 6   CO                            4181 non-null   float64
 7   Proximity_to_Industrial_Areas 4181 non-null   float64
 8   Population_Density            4181 non-null   int64
 9   Air Quality                   4181 non-null   object
```

<u>Asignación de variables.</u>

```
Variables independientes:
   Temperature  Humidity  PM10   NO2   SO2    CO
0         29.8      59.1  17.9  18.9   9.2  1.72
1         28.3      75.6  12.2  30.8   9.7  1.64
2         23.1      74.7  33.8  24.4  12.6  1.63
3         27.1      39.1   6.3  13.5   5.3  1.15
4         26.5      70.7  16.0  21.9   5.6  1.01
```

```
Variable dependiente:
0     5.2
1     2.3
2    26.7
3     6.1
4     6.9
Name: PM2.5, dtype: float64
```

METODOLOGÍA

Regresión: Modelado y predicción de valores numéricos

La regresión es un modelo que consiste en un algoritmo que modela la relación entre una variable dependiente (o variable objetivo) y una o más variables independientes (o predictores). Su objetivo principal es predecir valores numéricos y comprender cómo las variables independientes influyen en la dependiente. Por lo tanto, se puede decir que es una herramienta poderosa para predecir y analizar datos cuya capacidad para simplificar datos complejos y revelar relaciones importantes la convierte en una pieza clave en la toma de decisiones informadas.

-Implementación:

- El primer paso para comenzar a hacer nuestro modelo de regresión fue definir nuestras variables. Teniendo en cuenta que para la regresión se requieren de datos numéricos, entonces establecimos una de nuestras variables como dependientes y las otras como independientes. Lo anterior tomando en cuenta que ya se hizo el tratamiento de la base de datos para poder trabajar con los datos que necesitamos
    - La elección de la variable dependiente fue PM2.5 ya que ésta representa a las partículas que causan un mayor riesgo a la salud, por tal motivo queremos buscar la relación de los demás factores para saber lo que influye en tener una alta o baja concentración de estas partículas.

- Habiendo elegio las variables, lo que se hizo fue escalar los datos para que las variables independientes tengan la misma escala.
- Luego procedimos a dividir los datos en conjuntos de entrenamiento y de prueba para garantizar que el modelo puede generalizar bien a datos no vistos, evaluando su desempeño en un conjunto de datos separados.
- Con los datos listos, generamos graficos de dispersión para cada variable independiente y conocer si tenían alta o baja colinealidad con respecto a la variable dependiente
- Después de ver los resultados de la gráfica se pudo ver en su mayoría poca linealidad, lo cual nos hacía descartar algunos modelos a usar, sin embargo, procedimos a hacer un análisis más.
- El análisis que hicimos fue calcular el VIF (Variance Inflation Factor) ya que ésta se utiliza para detectar multicolinealidad entre las variables independientes siendo éstas las interpretaciones a los resultados a tomar en cuenta:
    - VIF = 1 : No hay correlación entre la variable y las demás
    - 1 < VF < 5: Hay correlación es moderada y generalmente aceptable
    - VIF > 5: Existe alta correlación (multicolinealidad severa)
    - VIF > 10: Multicolinealidad extrema

- Los resultados obtenidos mostraron una alta colinelaidad en la mayoría de las variables. Por lo tanto, habiendo recopilado toda la información mencionada a través del análisis, procedimos a la investigación de los modelos que teníamos como alternativas, llegando a la conclusión de probar el modelo de **Ridge** ya que este tiene como propósito principal manejar esta multicolinealidad al regularizar los coeficientes, evitando que se inflen debido a relaciones redundantes entre las variables. Y de igual manera a diferencia de otros

métodos, ésta no elimina variables del modelo, lo que nos es beneficioso ya que todas las variables tienen importancia en el análisis.

- Una vez establecido el modelo comenzamos con su implementación, la cual debido a que tiene un hiperparámetro llamado alpha que controla la intensidad de la penalización aplicada a los coeficientes del modelo, decidimos hacer una prueba rápida para determinar el alpha que mejor nos dé un resultado
- Posteriormente implementamos el modelo de Ridge directamente desde el módulo de linear_model de la biblioteca scikit-learn
- Después haber aplicado el modelo y hecho una predicción, generamos un análisis del error promedio al cuadrado (MSE) entre los valores reales y los valores predichos para evalluar la precisión del modelo
    - Entre más cercano al 0, significa que son más precisas las predicciones
- De igual manera obtuvimos el coeficiente de determinación (R2) ya que evalua el porcentaje de la variabilidad de los datos reales
    - Entre más cercano al 1, significa que el modelo explica casi toda la variabilidad de los datos reales, lo que indican un buen ajuste
- Complementando lo anterior generamos una gráfica de dispersión para ver las tendencias entre valores reales y predichos
- Por último, hicimos una predicción introduciendo datos nuevos y guardar dicha predicción en un archivo csv.

Clasificación: Categorización de datos en clases predefinidas

Un modelo de clasificación es una herramienta utilizada para categorizar datos en clases o categorías predefinidas basándose en patrones aprendidos durante su entrenamiento. Su objetivo principal es predecir etiquetas, ya sea en problemas binarios o en problemas multicategóricos. Estos modelos permiten automatizar decisiones, mejorar la precisión en la asignación de categorías y procesar grandes volúmenes de información de manera eficiente, facilitando la resolución de problemas complejos en diversos campos.

-Implementación:

- El primer paso fue elegir las variables independientes y la dependiente. Ahora bien, como ahora estamos trabajando con un modelo de clasificación, aquí se podría trabajar con variables numéricas y categóricas. Por lo tanto, para nuestro proyecto, lo que hicimos fue definir como variable dependiente la calidad del aire, la cual es categórica porque tiene como valores: “Moderate”, “Good”, “Hazardous”, “Poor”.

```
print(df.columns)
print("Valores únicos en 'Air Quality':", df['Air Quality'].unique())
```

```
Index(['Temperature', 'Humidity', 'PM2.5', 'PM10', 'NO2', 'SO2', 'CO',
       'Air Quality'],
      dtype='object')
Valores únicos en 'Air Quality': ['Moderate' 'Good' 'Hazardous' 'Poor']
```

  - La elección de ésta como dependiente se debe a que queremos a partir de los compuestos que se encuentran en el aire, determinar cuando la calidad del aire es buena, mala, pobre o riesgosa.
- Ahora bien, para facilidad del modelo que se está generando, se etiquetaron estos valores para volver numéricos los valores categóricos

```
label_encoder = LabelEncoder()
y_clas = label_encoder.fit_transform(df['Air Quality'])

# Verificar la codificación
print("Clases codificadas:", label_encoder.classes_)  # Mapeo original
print("y_clas después de codificar:", y_clas[:10])    # Primeros valores codificados
```

```
Clases codificadas: [0 1 2 3]
y_clas después de codificar: [2 2 2 0 0 1 3 2 3 3]
```

- Teniendo las variables definidas y listas, se dividieron los datos en conjuntos de entrenamiento y de prueba
- Posteriormente, ya que hay que elegir un modelo de las varias opciones que se tienen, procedimos a investigar las alternativas, y debido a que cada una es recomendada para casos particulares, para este caso donde estamos trabajando con variables independientes numéricas y una dependiente categórica, la opción que más se adecúa es Gradient boosting, por lo tanto, elegimos esa para nuestro análisis.
- Una vez definido lo anterior, creamos y entrenamos el modelo con ayuda de la biblioteca y función ya existentes de XGBClassifier dentro de xgboost. Despúes obtuvimos la precisión del modelo.
- Al haber obtenido una precisión aceptable, verificamos la importancia de las características del modelo para saber si alguna no era necesaria para hacer correcciones y mejoras en nuestro modelo.

- Después le ingresamos datos nuevos a nuestro modelo entrenado con el fin de que, a partir de los valores de los componentes del aire, nuestro modelo nos indique cuál es la calidad del aire y estos se guarden en un archivo csv.
    - Cabe recordar que al inicio habíamos etiquetado los valores categóricos en numéricos por lo que al final antes de guardarlos en el documento, convertimos las etiquetas a sus valores categóricos para que la salida sea más entendible.
- Por último, a modo de hacer correcciones, probamos el modelo de Rnadom Forest y así hacer una comparación con el método utilizado por lo que lo implementamos, lo entrenamos e hicimos una tabla comparativa de las características de ambos modelos y sus valores de precisión

Agrupamiento: Identificación de patrones y agrupación de datos no etiquetados

Un modelo de agrupación, también conocido como clustering, es una técnica en machine learning no supervisada que consiste en agrupar datos en subconjuntos llamados clusters o grupos, basándose en la similitud entre ellos. Los datos dentro de un mismo grupo son más similares entre sí que con los de otros grupos. Esta técnica es especialmente útil cuando no se tienen etiquetas predefinidas, y permite descubrir patrones ocultos en los datos. Su importancia radica en que proporciona una visión más profunda y estructurada de los datos, permitiendo una mejor toma de decisiones al revelar estructuras naturales o relaciones desconocidas entre los elementos, optimizando procesos y estrategias en diversos campos.

-Implementación:

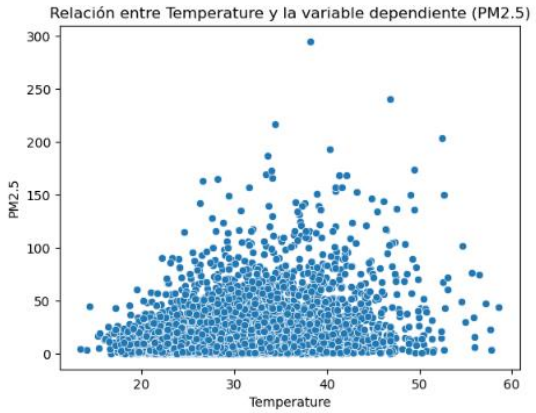
- Lo primero es escalar los datos con el fin de transformar las variables independientes para que tengan la misma escala
- Para poder entender mejor los datos con los que se entrenará el modelo, utilizamos la técnica de PCA para transformar el conjunto de datos con muchas variables en un conjunto más pequeño que aún conserva la mayor parte de la información presente en los datos originales. Así podemos visualizar la forma de nuestros datos
- Luego utilizando el método del codo, obtuvimos visualmente el mejor valor de k, que representa el número de clústeres a utilizar
- Habiendo definido lo anterior creamos y entrenamos dos modelos de agrupación, uno con K-Means y otro con DBSCAN para visualizar y comparar los resultados de ambos.
- Finalmente, para la comparación utilizamos Silhouette Score que es una métrica utilizada para evaluar la calidad de las agrupaciones (clusters) en algoritmos de clustering cuyo objetivo principal es medir qué tan bien cada objeto (dato) está asignado a su cluster y qué tan separados están los clusters entre sí.
    - Cercano a 1: El objeto está bien asignado a su cluster.
    - Cercano a 0: El objeto está en la frontera entre clusters.
    - Negativo: El objeto está asignado al cluster equivocado.

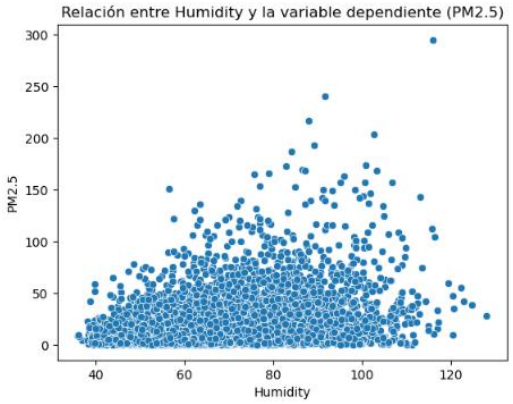
RESULTADOS

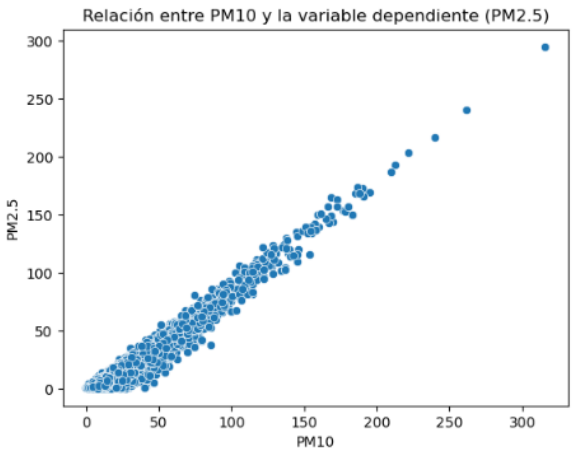
- ## Regresión

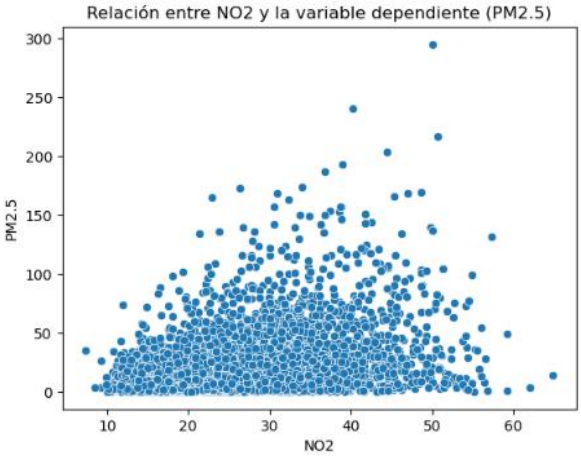
-Resultados que se obtuvieron

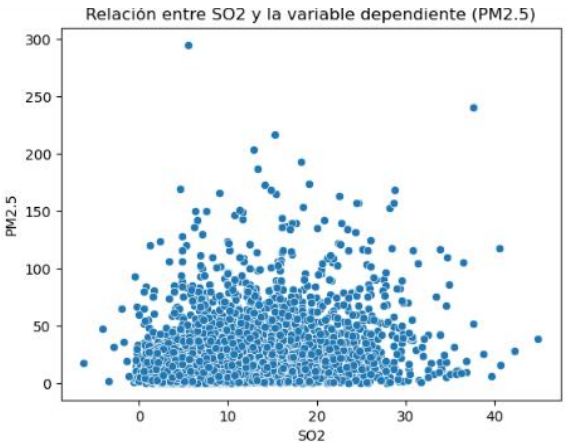
- Visualización de la relación entre las variables independientes y la dependiente

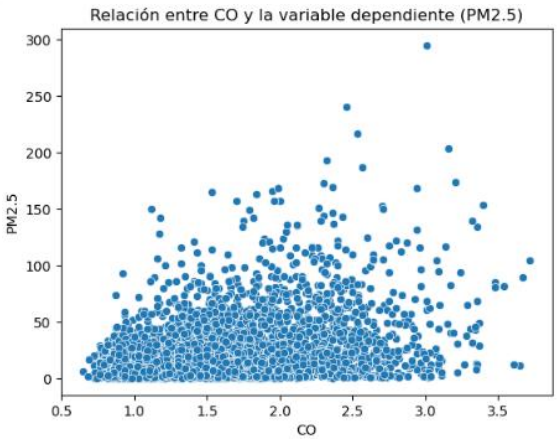












- Cálculo del VIF (Variance Inflation Factor)

```python
# Calcular VIF
vif = pd.DataFrame()
vif["Variable"] = X.columns
vif["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
print(vif)
```

```
      Variable        VIF
0  Temperature  29.341705
1     Humidity  22.436404
2         PM10   3.036695
3          NO2  21.320026
4          SO2   5.894436
5           CO  27.936957
```

- Cálculo del mejor alpha para el modelo de regresión de Ridge

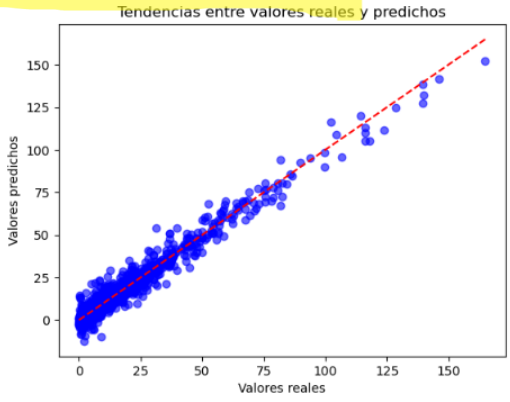
```python
[707]: # Definir el rango de valores de alpha para reegresión Ridge
parametros = {'alpha': [0.01, 0.1, 1, 10, 100]}

# Configurar GridSearchCV para buscar el mejor alpha
modelo_ridge = Ridge()  # Modelo base
grid_search = GridSearchCV(modelo_ridge, parametros, cv=5, scoring='neg_mean_squared_error')  # 5 pliegues
grid_search.fit(X_train, y_train)

# Obtener el mejor valor de alpha y entrenar el modelo
mejor_alpha = grid_search.best_params_['alpha']
print("Mejor valor de alpha:", mejor_alpha)
```

```
Mejor valor de alpha: 0.1
```

- Cálculo de MSE , R2 y gráfica de tendencias entre valores reales y predichos

```python
# Aplicar regresión Ridge
modelo_ridge = Ridge(alpha=0.1)  # Ajusta alpha para controlar la penalización
modelo_ridge.fit(X_train, y_train)

# Predicción
predicciones_ridge = modelo_ridge.predict(X_test)

# Evaluación del modelo
mse_ridge = mean_squared_error(y_test, predicciones_ridge)
r2_ridge = r2_score(y_test, predicciones_ridge)
print(f"Ridge MSE: {mse_ridge}, R²: {r2_ridge}")

# Identificar tendencias visualmente
plt.scatter(y_test, y_pred_optimizado, color='blue', alpha=0.6)
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red', linestyle='--')
plt.xlabel('Valores reales')
plt.ylabel('Valores predichos')
plt.title('Tendencias entre valores reales y predichos')
plt.show()
```

```
Ridge MSE: 20.99045880146898, R²: 0.9585285402332842
```



- Predicción de los datos nuevos ingresados al modelo entrenado

```python
# Generar predicciones con el modelo optimizado
predicciones_nuevas = modelo_ridge_optimizado.predict(datos_nuevos)

# Mostrar las predicciones
print("Predicciones para los datos del archivo CSV:")
print(predicciones_nuevas)

# Guardar las predicciones en un archivo CSV
predicciones_df = pd.DataFrame({'Predicciones': predicciones_nuevas})
predicciones_df.to_csv('predicciones_ridge.csv', index=False)

print("\nLas predicciones han sido guardadas en 'predicciones_ridge.csv'.")

# Predicciones con el modelo optimizado
predicciones = modelo_ridge_optimizado.predict(datos_nuevos)
print("Predicciones para los datos del archivo CSV:", predicciones)

predicciones_df = pd.DataFrame({'Predicciones': predicciones_nuevas})
predicciones_df.to_csv('predicciones_ridge.csv', index=False)

print("\nLas predicciones han sido guardadas en 'predicciones_ridge.csv'.")
```

```
Predicciones para los datos del archivo CSV:
[5308.02205799 3666.84815778  626.98206937 ... 2297.6289593   337.62668624
 1015.02613797]

Las predicciones han sido guardadas en 'predicciones_ridge.csv'.
Predicciones para los datos del archivo CSV: [5308.02205799 3666.84815778  626.98206937 ... 2297.6289593   337.62668624
 1015.02613797]

Las predicciones han sido guardadas en 'predicciones_ridge.csv'.
```

- **Clasificación**

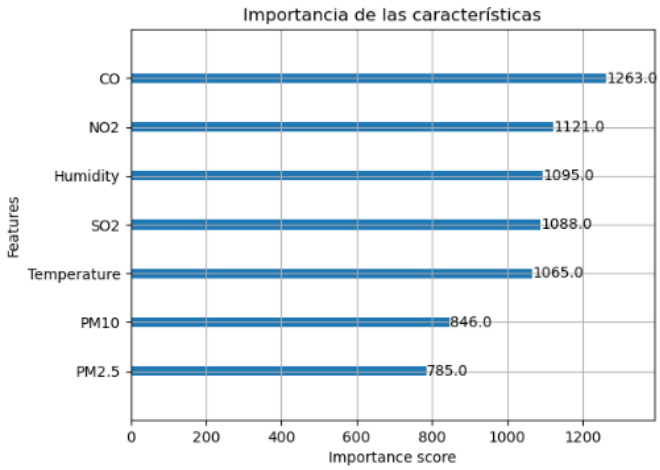
<u>-Resultados que se obtuvieron</u>

- Modelo de clasificación y precisión

```python
# Crear el modelo de Gradient Boosting
modelo_xgb = XGBClassifier(random_state=42, eval_metric='mlogloss')

# Entrenar el modelo
modelo_xgb.fit(X_train, y_train)

# Evaluar el modelo en los datos de prueba
y_pred_xgb = modelo_xgb.predict(X_test)
print(f"Precisión: {accuracy_score(y_test, y_pred_xgb)}")

print("Valores únicos en y:", np.unique(y_clas))
print("Cantidad de clases:", len(np.unique(y_clas)))
```

```
Precisión: 0.918
Valores únicos en y: [0 1 2 3]
Cantidad de clases: 4
```

- Importancia de las características del modelo de Gradient Boosting



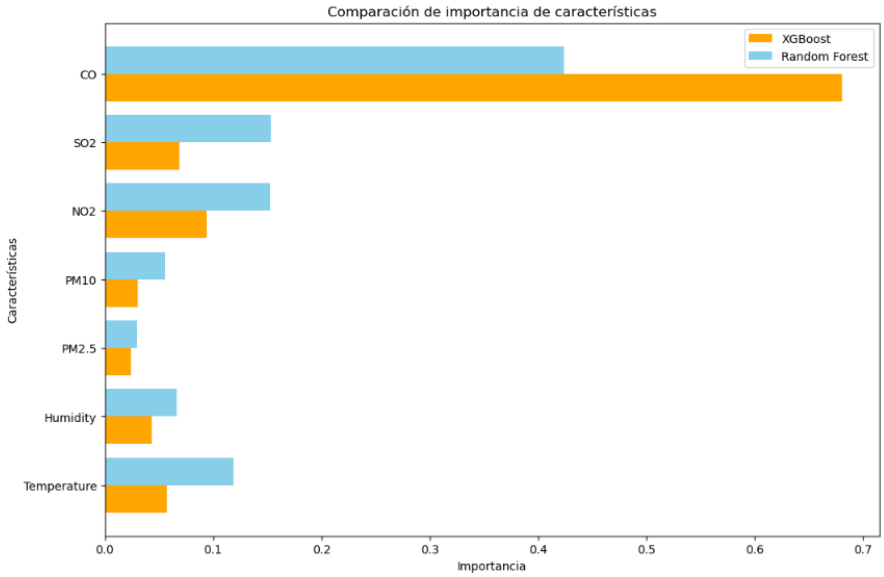
- Implementación del modelo

```
#Predicciones
predicciones = modelo_xgb.predict(datos_nuevos_clas)
print(f"Clase predicha: {predicciones}")
predicciones_df = pd.DataFrame({'Predicciones': predicciones})
predicciones_df.to_csv('predicciones_nuevas.csv', index=False)

print("Predicciones guardadas en 'predicciones_nuevas.csv'")
```

```
Clase predicha: [1 1 3 ... 1 1 2]
Predicciones guardadas en 'predicciones_nuevas.csv'
```

- Comparación de importancia de las características del modelo de Gradient Boosting vs Random Forest

	Características	Importancia_XGBoost	Importancia_RandomForest
0	Temperature	0.057617	0.118696
1	Humidity	0.043131	0.066536
2	PM2.5	0.024494	0.029868
3	PM10	0.030694	0.055531
4	NO2	0.094238	0.152308
5	SO2	0.069056	0.153446
6	CO	0.680769	0.423615

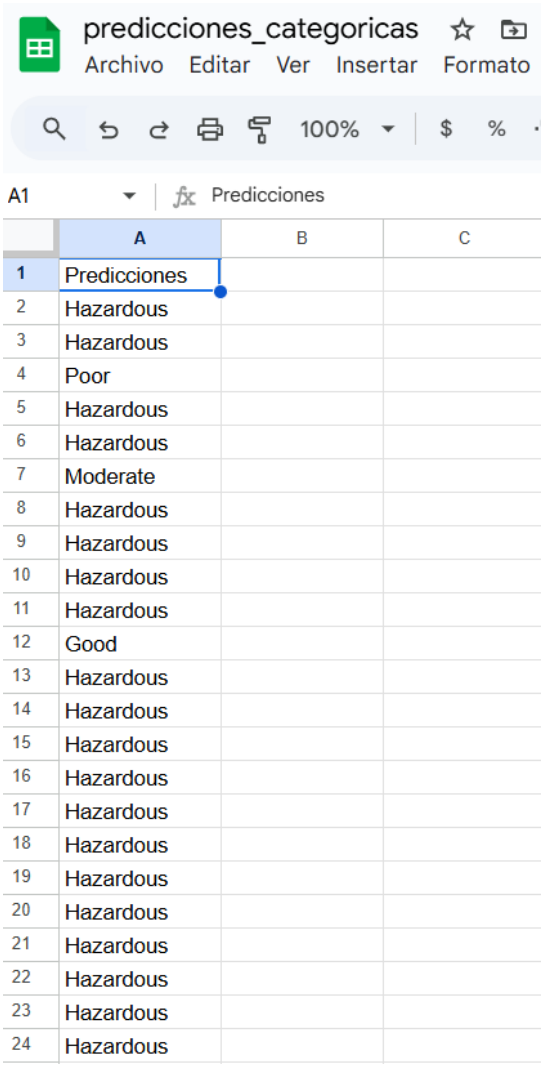


- Comparación de la precisión y matrices de confusión entre el modelo de Gradient Boosting vs Random Forest

[613]:
```python
# Métricas para XGBoost
print("Métricas para XGBoost:")
print("Precisión de XGBoost:", accuracy_score(y_test, y_pred_xgb))
print("Reporte de clasificación de XGBoost:\n", classification_report(y_test, y_pred_xgb))
print("Matriz de confusión de XGBoost:\n", confusion_matrix(y_test, y_pred_xgb))

# Métricas para Random Forest
print("\nMétricas para Random Forest:")
print("Precisión de Random Forest:", accuracy_score(y_test, y_pred_rf))
print("Reporte de clasificación de Random Forest:\n", classification_report(y_test, y_pred_rf))
print("Matriz de confusión de Random Forest:\n", confusion_matrix(y_test, y_pred_rf))

# Comparación final
print("\nComparación de precisión entre modelos:")
print(f"XGBoost: {accuracy_score(y_test, y_pred_xgb) * 100:.2f}%")
print(f"Random Forest: {accuracy_score(y_test, y_pred_rf) * 100:.2f}%")
```

```
Métricas para XGBoost:
Precisión de XGBoost: 0.918
Reporte de clasificación de XGBoost:
              precision    recall  f1-score   support

           0       0.99      0.99      0.99       409
           1       0.83      0.80      0.82       111
           2       0.93      0.93      0.93       294
           3       0.79      0.81      0.80       186

    accuracy                           0.92      1000
   macro avg       0.89      0.88      0.88      1000
weighted avg       0.92      0.92      0.92      1000

Matriz de confusión de XGBoost:
 [[406   0   3   0]
 [  0  89   0  22]
 [  5   0 272  17]
 [  0  18  17 151]]

Métricas para Random Forest:
Precisión de Random Forest: 0.923
Reporte de clasificación de Random Forest:
              precision    recall  f1-score   support

           0       0.99      0.99      0.99       409
           1       0.85      0.79      0.82       111
           2       0.93      0.94      0.93       294
           3       0.81      0.82      0.81       186

    accuracy                           0.92      1000
   macro avg       0.89      0.89      0.89      1000
weighted avg       0.92      0.92      0.92      1000

Matriz de confusión de Random Forest:
 [[406   0   3   0]
 [  0  88   0  23]
 [  4   0 277  13]
 [  0  15  19 152]]

Comparación de precisión entre modelos:
XGBoost: 91.80%
Random Forest: 92.30%
```

- Tabla de predicciones generadas por el modelo de Gradient Boosting


predicciones_categoricas

Archivo Editar Ver Insertar Formato

100% $ %

A1 Predicciones

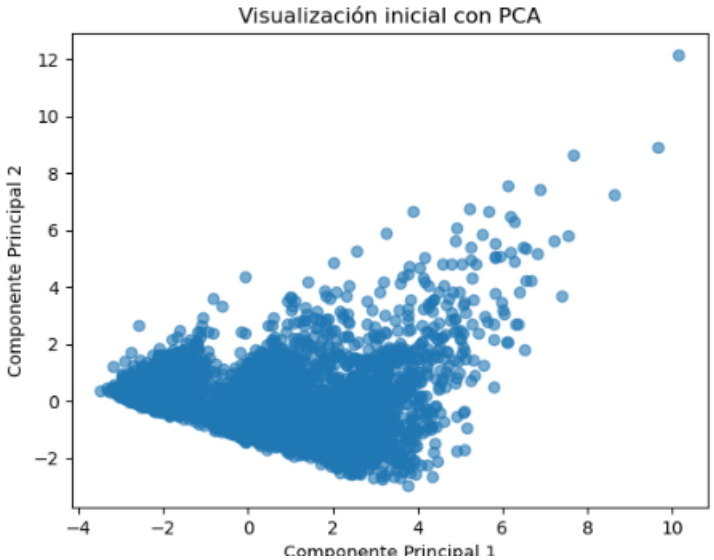
	A	B	C
1	Predicciones		
2	Hazardous		
3	Hazardous		
4	Poor		
5	Hazardous		
6	Hazardous		
7	Moderate		
8	Hazardous		
9	Hazardous		
10	Hazardous		
11	Hazardous		
12	Good		
13	Hazardous		
14	Hazardous		
15	Hazardous		
16	Hazardous		
17	Hazardous		
18	Hazardous		
19	Hazardous		
20	Hazardous		
21	Hazardous		
22	Hazardous		
23	Hazardous		
24	Hazardous		

- **Agrupación**

-Resultados que se obtuvieron

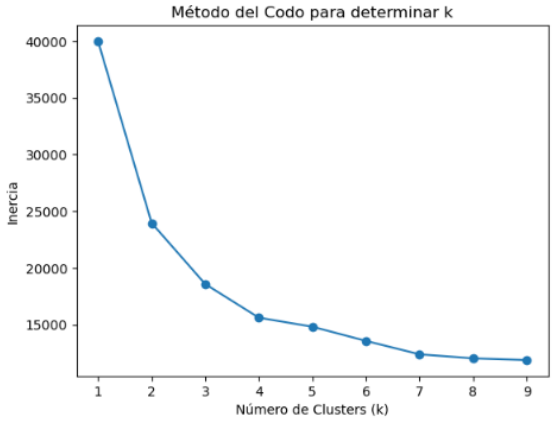
- Técnica de PCA para visualizar los datos

```
pca = PCA(n_components=2)
X_reducido = pca.fit_transform(X_scaled)

plt.scatter(X_reducido[:, 0], X_reducido[:, 1], alpha=0.6)
plt.xlabel('Componente Principal 1')
plt.ylabel('Componente Principal 2')
plt.title('Visualización inicial con PCA')
plt.show()
```



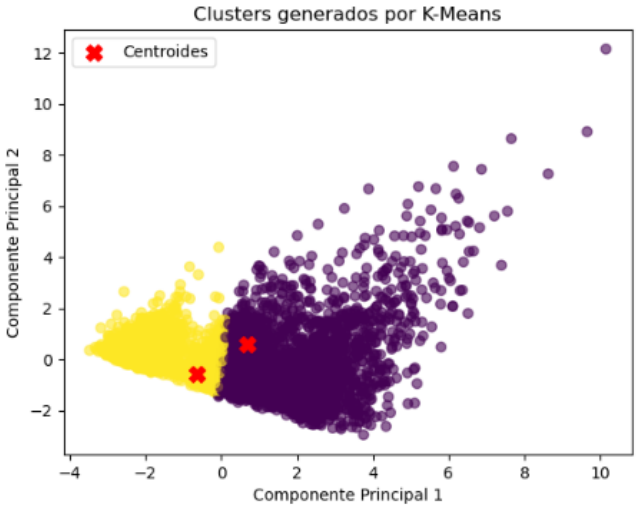
- Método del codo

```
distorsiones = []
K = range(1, 10)  # Probar entre 1 y 10 clusters
for k in K:
    kmeans_temp = KMeans(n_clusters=k, random_state=42).fit(X_scaled)
    distorsiones.append(kmeans_temp.inertia_)

plt.plot(K, distorsiones, marker='o')
plt.xlabel('Número de Clusters (k)')
plt.ylabel('Inercia')
plt.title('Método del Codo para determinar k')
plt.show()
```



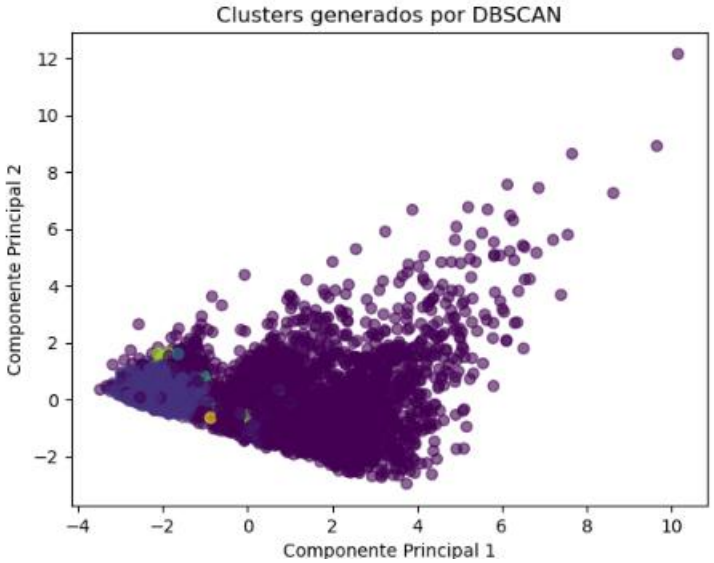
- Clusters generados por K-Means en espacio PCA

```
plt.scatter(X_reducido[:, 0], X_reducido[:, 1], c=modelo_kmeans.labels_, cmap='viridis', alpha=0.6)
plt.scatter(modelo_kmeans.cluster_centers_[:, 0], modelo_kmeans.cluster_centers_[:, 1],
            color='red', marker='X', s=100, label='Centroides')
plt.xlabel('Componente Principal 1')
plt.ylabel('Componente Principal 2')
plt.title('Clusters generados por K-Means')
plt.legend()
plt.show()
```



- Clusters generados por DBSCAN

```
# Aplicar DBSCAN para clusters con densidades variables
modelo_dbscan = DBSCAN(eps=0.5, min_samples=5)  # Ajusta eps y min_samples según tu d
labels_dbscan = modelo_dbscan.fit_predict(X_scaled)

# Visualizar clusters generados por DBSCAN en espacio PCA
plt.scatter(X_reducido[:, 0], X_reducido[:, 1], c=labels_dbscan, cmap='viridis', alpha
plt.xlabel('Componente Principal 1')
plt.ylabel('Componente Principal 2')
plt.title('Clusters generados por DBSCAN')
plt.show()
```



- Resultados de Silhoutte Score

```python
if len(set(labels_dbscan)) > 1:  # Verifica que haya más de 1 cluster generado
    silhouette_dbscan = silhouette_score(X_scaled, labels_dbscan)
    print(f"Silhouette Score para DBSCAN: {silhouette_dbscan}")
else:
    print("DBSCAN no generó suficientes clusters para calcular el Silhouette Score.")

# Interpretar y decidir
print("Resultados:")
print(f"Silhouette Score de K-Means: {silhouette_kmeans}")
if len(set(labels_dbscan)) > 1:
    print(f"Silhouette Score de DBSCAN: {silhouette_dbscan}")
else:
    print("DBSCAN detectó ruido o clusters insuficientes.")
```

```
Silhouette Score de K-Means: 0.357683674723238
Silhouette Score de DBSCAN: -0.2385121372822191
```

DISCUSIÓN Y CONCLUSIONES

Discusión:
Los resultados obtenidos confirman la eficacia de la aplicación de algoritmos de inteligencia artificial en el análisis de la contaminación del aire. La alta precisión en la predicción y clasificación respalda el uso de estos modelos para apoyar la toma de decisiones en políticas ambientales. No obstante, se identificaron limitaciones, como la presencia de alta colinealidad en algunas variables, lo que sugiere la necesidad de explorar técnicas de reducción de dimensionalidad o nuevos métodos de regularización.

Recomendaciones para su implementación:
Se sugiere ampliar la base de datos con mediciones en tiempo real a través de sensores IoT, así como probar modelos adicionales que puedan optimizar aún más la precisión de las predicciones. Además, se recomienda integrar un sistema de alertas para identificar de manera proactiva episodios críticos de contaminación. Así como si su aplicación se llegara a dar en un futuro, se deberá implementar un modelo de predicción de desastres naturales, como los incendios forestales, con su agregado, la prevención y protección de la población en las áreas circundantes a donde se puedan llegar estos riesgos, podrán estar mejor preparadas y protegidas, ante las caídas de la calidad del aire, y evitar las muertes, enfermedades por la contaminación de agentes nocivos.

En conclusión, el proyecto demuestra que la aplicación de técnicas de machine learning en el análisis de datos ambientales es una herramienta valiosa para comprender y mitigar la problemática de la contaminación del aire, proporcionando una base sólida para futuras investigaciones y mejoras en la gestión de la calidad del aire.

- **Regresión**

-Problemas que se tuvo

- Más que un problema, a la hora de implementar el modelo, se requirió de un análisis para llegar a lo que consideráramos el mejor modelo posible para nuestro caso, lo cual implicó un mayor tiempo antes de llegar a implementar.
- El otro que podría entrar en esta categoría sería el interpretar la predicción que generó y guardó en el archivo csv.

-Interpretación de resultados

- Las gráficas iniciales mostraron en su mayoría una poca linealidad entre las variables independientes con la dependiente
- Los VIF de cada variable demuestran que hay una alta colinealidad entre las variables dependientes
- El mejor alpha para el modelo de regresión de Ridge fue de 0.1 por lo que se espera que poniendo ese valor en el modelo nos dé mejores resultados
- El MSE de 20.99 nos indica que, para nuestro caso de y_test que tienen 1000 valores, se considera muy bajo por lo que suguiere una buena precisión en las predicciones
- El R2 de 0.9585 nos indica que el modelo explica el 95.85% de la variabilidad en los datos reales. Lo que es un excelente ajuste y sugiere que el modelo está capturando muy bien las relaciones entre las variables.

• Clasificación

-Problemas que se tuvo

- El problema que al principio no se tenía claro, era el de trabajar con una variable dependiente categórica y las dependientes numéricas, sin embargo, el etiquetar la categórica, facilitó esa parte.
- El no haber obtenido una precisión demasiada alta nos hizo plantear probar otros métodos a pesar de que la teoría nos decía que el que utilizamos era el más adecuado por las características de nuestro enfoque de los datos.

-Interpretación de los datos una vez ya analizados

- La precisión de 0.918 nos indica que el modelo explica el 91.8% de la variabilidad en los datos reales. Lo que es un excelente ajuste y sugiere que el modelo está capturando muy bien las relaciones entre las variables por lo que genera buenas predicciones, sin embrgo ésta podría ser posiblemente mejor así que se interpretaría que a pesar de que teóricamente éste indique ser el mejor modelo para nuestro caso, podría haber otros que tengan una mejor precisión
- La gráfica de importancia de características muestra que todas tienen relevancia en la influencia de la calidad del aire
- Por último, el archivo csv que muestra las predicciones generaron un buen resultado indicando la calidad de aire para cada fila de datos.

• Agrupamiento

-Problemas que se tuvo

- El único problema que se nos presentó y de gran importancia, es el resultado obtenido que no fue el óptimo ya que ambos modelos usados no presentaron buenos resultados a pesar de ser los que teóricamente más adecuados para nuestro proyecto

-Interpretación de los datos una vez ya analizados

- La técnica de PCA nos mostró que los datos tienen una forma irregular por lo que no hay linealidad entre ellas.
- Las gráficas de clusters generados para los métodos de K-Means y DBSCAN son dispersas y no muestran un agrupamiento definido bueno
- Los valores del puntaje de Silhoutte para ambos métodos nos indican:
    - DBSCAN de –0.23: Un Silhouette Score negativo indica que algunos puntos están más cerca de los clusters vecinos que del propio cluster al que fueron asignados. Esto sugiere que el modelo DBSCAN no está agrupando correctamente los datos o que hay un alto grado de solapamiento entre los clusters.

    - K-Means de 0.35: Un Silhouette Score positivo y cercano a 1 sugiere que los puntos están bien agrupados dentro de sus respectivos clusters y que los clusters están claramente separados entre sí, sin embargo al tener un valor no tan cercano a 1, esto indica que s un valor moderado, lo que implica que el modelo K-Means agrupa los datos de forma aceptable, pero no perfectamente. Es posible que algunos puntos estén en los límites entre clusters o que haya algo de solapamiento entre los grupos.
- Por lo tanto, el resultado aceptable para este modelo es el de K-Means.

REFERENCIAS

- Informe de la ONU sobre la contaminación del aire en 2021: https://www.unicef.org/guatemala/comunicados-prensa/la-contaminaci%C3%B3n-del-aire-represent%C3%B3-81-millones-de-muertes-nivel-mundial-en#:~:text=Comunicado%20de%20prensa-,LA%20CONTAMINACI%C3%93N%20DEL%20AIRE%20REPRESENT%C3%93%208%2C1%20MILLONES%20DE%20MUERTES,NI%C3%91OS%20MENORES%20DE%20CINCO%20A%C3%91OS
- Fundación Aquae. (2021, 26 agosto). Contaminación del aire: causas y tipos - Fundación Aquae. https://www.fundacionaquae.org/wiki/causas-y-tipos-de-la-contaminacion-del-aire/
- Base de datos utilizada para el entenamiento: https://www.kaggle.com/datasets/mujtabamatin/air-quality-and-pollution-assessment
- Bade de datos utilizada para la predicción de nuevos datos: https://www.kaggle.com/datasets/khushikyad001/air-quality-data

ANEXO:

-Modelo de clasificación utilizado como análisis de comparación para verificar resultados comprara con los obtenidos

- Arquitectura de la Red Neuronal

```
model_nn = Sequential([
    Dense(128, activation='relu', kernel_regularizer=l2(0.001), input_shape=(X_train.shape[1],)),
    BatchNormalization(),
    Dropout(0.4),
    Dense(64, activation='relu', kernel_regularizer=l2(0.001)),
    BatchNormalization(),
    Dropout(0.3),
    Dense(len(np.unique(y)), activation='softmax')
])
```

- Se implemento un codigo de red neuronal profunda para clasificar la calidad del aire.

I. Capas y Configuración:

1. Capa de Entrada (Dense 128)

- 128 neuronas con activación ReLU
- Regularización L2 (λ=0.001) para prevenir overfitting
- input_shape=(X_train.shape[1],): Se adapta al número de features

2. Batch Normalization

- Normaliza las salidas de la capa anterior
- Acelera el entrenamiento y mejora la estabilidad

3. Dropout (40%)

- Apaga aleatoriamente el 40% de las neuronas durante el entrenamiento
- Técnica de regularización para evitar sobreajuste

4. Capa Oculta (Dense 64)

- 64 neuronas con ReLU
- Misma regularización L2 que la primera capa

5. Capa de Salida

- Neuronas igual al número de clases (len(np.unique(y)))
- Activación softmax para probabilidades multinomiales

```
optimizer = Adam(learning_rate=0.0005, clipvalue=0.5)
model_nn.compile(optimizer=optimizer,
                 loss='categorical_crossentropy',
                 metrics=['accuracy'])

early_stop = EarlyStopping(monitor='val_loss', patience=15, restore_best_weights=True)
```

Configuración del Entrenamiento

- **Optimizador Adam**:
    - Tasa de aprendizaje: 0.0005 (baja para ajuste fino)
    - clipvalue=0.5: Limita los gradientes para evitar explosiones
- **Función de Pérdida**: categorical_crossentropy (para clasificación multiclase)
- **Early Stopping**:
    - Detiene el entrenamiento si val_loss no mejora en 15 épocas
    - Restaura los mejores pesos encontrados

Proceso de entrenamiento.

```
print("\n🔄 Entrenando Red Neuronal...")
history = model_nn.fit(
    X_train_bal,
    to_categorical(y_train_bal),
    validation_data=(X_test, to_categorical(y_test)),
    epochs=200,
    batch_size=64,
    callbacks=[early_stop],
    verbose=1
)
```

- **Balanceo con SMOTE**: X_train_bal contiene datos aumentados para clases minoritarias
- **One-Hot Encoding**: to_categorical() convierte etiquetas a formato binario
- **Validación en 20% de datos**: validation_data usa el conjunto de prueba original
- **Batch Size 64**: Compromiso entre velocidad y estabilidad

Evaluación de modelo.

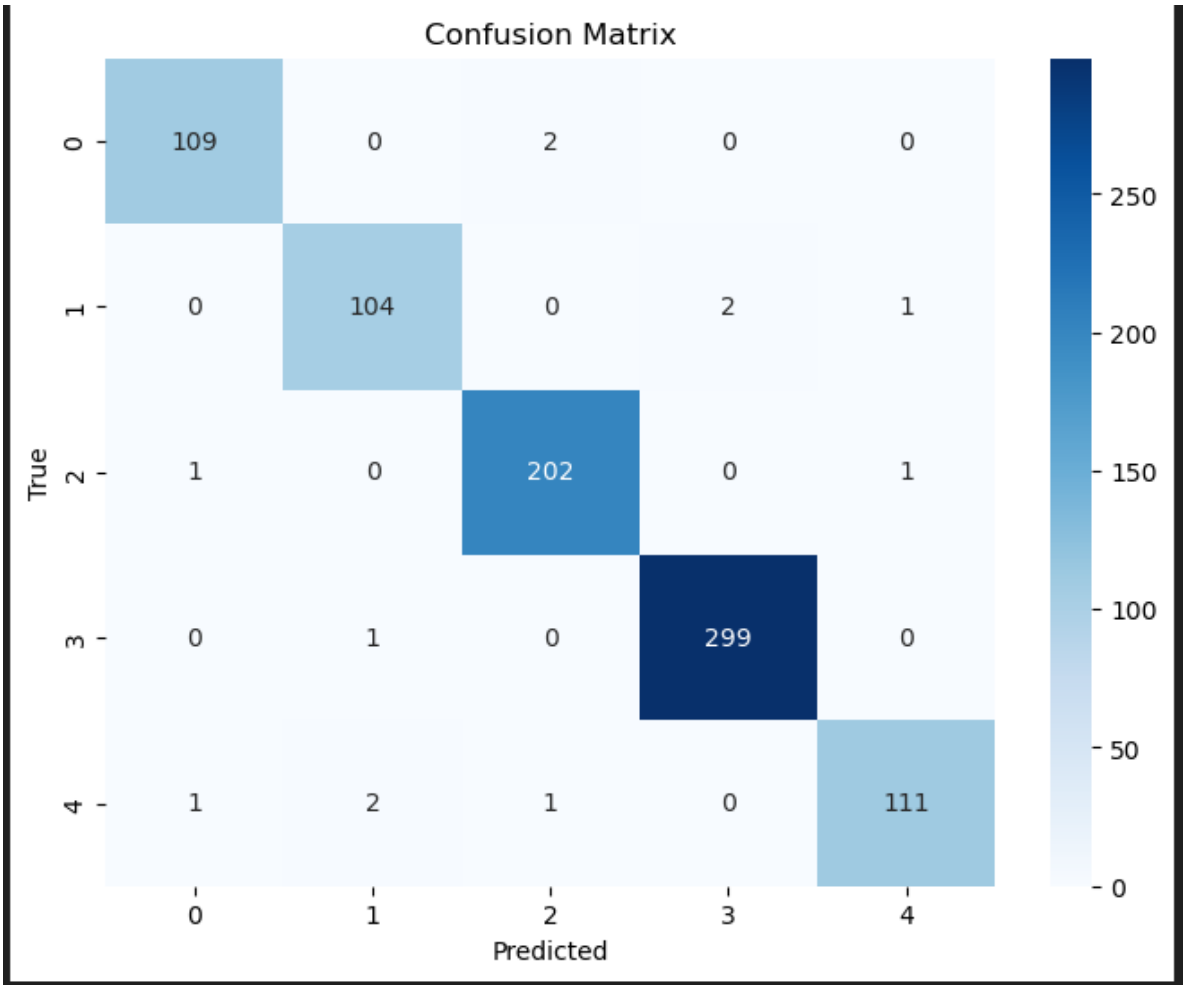
```
y_pred_nn = np.argmax(model_nn.predict(X_test), axis=1)
evaluate_model("Red Neuronal Mejorada", y_test, y_pred_nn)
```

II. Métricas Generadas:

1. **Accuracy**: Porcentaje de predicciones correctas
2. **Classification Report**:
    - Precisión, recall y F1-score por clase
    - Ejemplo para clase "Poor" (minoritaria):

precision recall f1-score support

Poor	0.97	0.96	0.96	142



Matriz de Confusion.