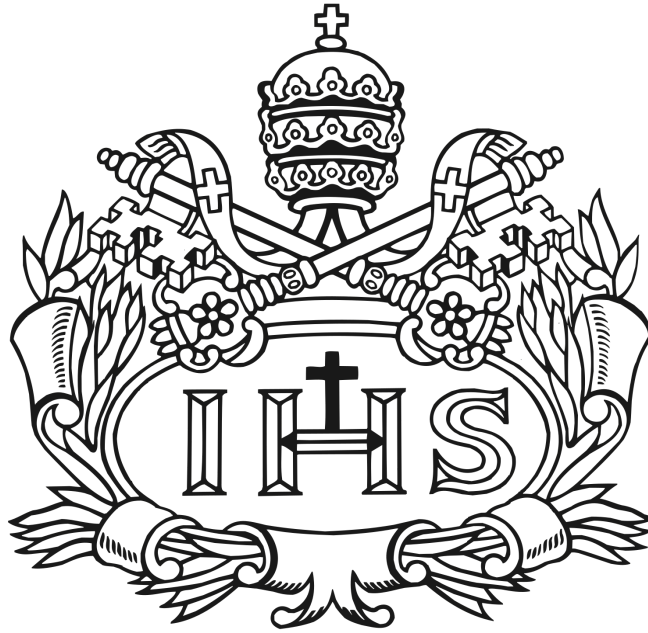


**Pontificia Universidad Javeriana**

**Proyecto de Procesamiento de Datos a Gran Escala**

**Entrega 2**



**Diego Alejandro Martínez Oviedo**

**Julian Andrey Mendez Rodriguez**

**Andres Felipe Galan Cardenas**

**Samuel Andres Lamilla Verjan**

**Steven Viscillinovick Robles Patiño**

## Entendimiento del negocio

El proyecto a realizar consiste en un análisis de datos a gran escala de múltiples conjuntos de datos, con la finalidad de identificar una relación entre los resultados de la prueba ICFES 11 y distintos factores como cobertura del servicio de internet, índices de pobreza en la región, niveles de educación, entre otros, para los municipios de Cundinamarca para el año 2023. El análisis se realiza con la finalidad de generar un plan de acción estratégico que optimice dichos indicadores y mejore la calidad educativa en Cundinamarca. Para esta proyecto se toman en cuenta los treinta municipios con más población del departamento de Cundinamarca para el año 2023, los cuales corresponden a Soacha, Facatativá, Fusagasugá, Chía, Zipaquirá, Mosquera, Madrid, Girardot, Funza, Cajicá, Villa de San Diego de Ubaté, Tocancipá, Sibaté, Cota, La Mesa, Guaduas, La Calera, Villeta, Sopó, Pacho, El Colegio, Cogua, Tenjo, Tabio, Silvania, El Rosal, Chocontá, Suesca, Gachancipá y Villapinzón [1]; La muestra se restringe a los treinta municipios con más población con la finalidad de proporcionar un análisis más centrado y significativo en un periodo temporal cercano a la actualidad

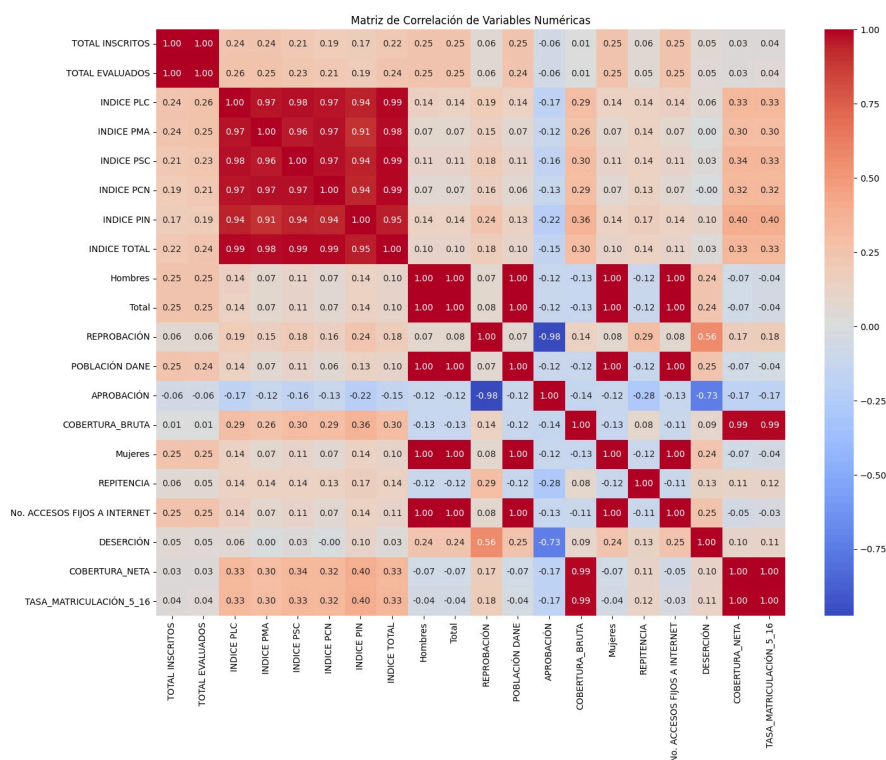
### 1. Filtros y transformaciones

Durante el proceso de procesamiento de datos, se integran múltiples fuentes de información (ICFES, DANE, MEN, MinTIC) para construir una base unificada de los municipios del departamento de Cundinamarca. Esta base consolidada se registró en el archivo [base.csv](#), el cual contiene indicadores clave para el análisis educativo y socio-digital.

Transformaciones realizadas:

- Estandarización de nombres de municipios: Se transformaron todos los nombres a minúsculas, sin tildes ni caracteres especiales, y se compararon con una lista validada de municipios de Cundinamarca.
- Filtrado geográfico: Se excluyeron municipios fuera del departamento objetivo, garantizando el enfoque exclusivo en Cundinamarca.
- Conversión de tipos de datos: Se normalizaron formatos de columnas (por ejemplo, años como categorías, y porcentajes como decimales).
- Unificación de esquemas: Se renombraron columnas con nombres estándar como [Municipio](#), [POBLACIÓN DANE](#), [COBERTURA NETA](#), [APROBACIÓN](#), etc., para facilitar cruces y comparaciones.
- Imputación de datos faltantes:

- En primera instancia, se llenaron los valores nulos con promedios municipales calculados a partir de cada fuente.
- Posteriormente, se emplearon medidas en caso de que algún valor continuará ausente.



Con la base final (**df icfes 4**), se generó una matriz de correlación para identificar posibles redundancias, relaciones significativas y variables irrelevantes. Esta matriz evidenció:

- Alta colinealidad entre los distintos índices del ICFES (**PMA**, **PCS**, **PCN**, **PIN**), lo que sugiere que se pueden representar de forma compacta con el **ÍNDICE TOTAL**.
- Correlaciones fuertes esperadas, como entre **COBERTURA BRUTA** y **TASA MATRICULACIÓN 5\_16**, o entre **APROBACIÓN** y **REPROBACIÓN** (negativa).
- Correlaciones débiles entre indicadores de conectividad (**No. ACCESOS FIJOS A INTERNET**) y rendimiento académico, lo que plantea interrogantes sobre su impacto real.

La tabla muestra el resultado del archivo final (**base.csv**), que contiene **una fila por municipio** con variables demográficas, educativas, de conectividad y académicas listas para análisis.

Esta base representa el **producto depurado del proceso ETL (extracción, transformación y carga)**, y constituye el punto de partida tanto para el análisis exploratorio como para los modelos predictivos y de segmentación desarrollados en las siguientes etapas del proyecto.

Columna	Descripción
<b>TOTAL INSCRITOS</b>	Número total de estudiantes inscritos para la prueba ICFES 11 en el municipio.
<b>TOTAL EVALUADOS</b>	Número total de estudiantes que finalmente presentaron la prueba ICFES 11.
<b>INDICE TOTAL</b>	Índice global de desempeño en la prueba ICFES 11 (promedio ponderado de las áreas evaluadas).
<b>No. ACCESOS FIJOS A INTERNET</b>	Número de conexiones fijas a internet reportadas en el municipio. Indicador de conectividad digital.
<b>POBLACIÓN DANE</b>	Población total proyectada del municipio según el DANE.
<b>COBERTURA NETA</b>	Porcentaje de estudiantes matriculados en el nivel educativo adecuado para su edad.
<b>APROBACIÓN</b>	Porcentaje de estudiantes que aprobaron el año escolar en el municipio.
<b>REPITENCIA</b>	Porcentaje de estudiantes que repitieron el año escolar.
<b>DESERCIÓN</b>	Porcentaje de estudiantes que abandonaron el sistema educativo en el año.
<b>DEPARTAMENTO</b>	Nombre del departamento al que pertenece el municipio (en este caso, debe ser siempre “Cundinamarca”).

Municipio	Nombre del municipio al que pertenecen los datos.
-----------	---

## 2. Respuesta a preguntas de negocios planteadas

Una vez estructurada y depurada la base de datos, se formularon cinco preguntas clave con el fin de comprender el comportamiento educativo en los municipios de Cundinamarca, tomando como base las pruebas Saber 11 y variables contextuales como conectividad, cobertura y población. Las preguntas planteadas buscan responder a desafíos reales de política educativa, entre ellos:

- Determinar si el acceso a Internet influye significativamente en el rendimiento académico.
- Explorar el papel que juegan la cobertura educativa y las tasas de matrícula en los resultados de las pruebas.
- Establecer la relación entre indicadores de abandono escolar y el desempeño académico.
- Identificar territorios con desempeño sobresaliente o preocupante.
- Analizar tendencias recientes y su posible evolución en el corto plazo.

Cada una de estas preguntas fue abordada utilizando análisis estadísticos, visualización de datos y comparación de indicadores clave, para brindar insumos estratégicos a tomadores de decisiones en el ámbito educativo y de conectividad digital.

### 1. ¿Cómo afecta el acceso a Internet el rendimiento en las pruebas ICFES?

La relación entre el número de accesos fijos a Internet y el índice total ICFES es muy débil. Aunque existe una leve tendencia positiva, esta no es significativa. La mayoría de los municipios, sin importar su nivel de conectividad, tienen un rendimiento similar en las

pruebas ICFES. Esto sugiere que el acceso a Internet por sí solo no garantiza un mejor desempeño académico.

2. ¿Cuál es el impacto de la tasa de matrícula y la cobertura educativa en los resultados de la prueba ICFES 11?

La cobertura educativa (como la cobertura neta y la tasa de aprobación) tiene una relación positiva moderada con el desempeño en las pruebas ICFES. Los municipios con mayor cobertura y mayor aprobación escolar tienden a tener mejores resultados. Sin embargo, no es una relación fuerte ni determinante, lo que indica que hay otros factores influyentes además de la cobertura.

3. ¿Existe correlación entre la tasa de deserción escolar en los municipios y los puntajes promedio de la prueba ICFES?

Existe una correlación negativa débil entre la tasa de deserción escolar y el índice total ICFES. Aunque los municipios con mayor deserción tienden a obtener puntajes ligeramente más bajos, la relación es muy dispersa y poco significativa. Por lo tanto, la deserción escolar no es un buen predictor del rendimiento ICFES por sí sola.

4. ¿Cuáles municipios tienen el mejor y peor desempeño en la prueba ICFES?

Los municipios con mejor desempeño son Sopó y La Calera, ambos con un índice total ICFES cercano a 0.9, lo que indica un rendimiento excelente. Por el contrario, Ricaurte, Topaipí, Girardot y Nimaíma se encuentran entre los municipios con peores resultados, con índices por debajo de 0.5. Esto evidencia desigualdad en la calidad educativa entre los municipios.

5. ¿Cómo ha variado la relación entre el acceso a internet y el desempeño académico en los últimos años en Cundinamarca?

Entre 2021 y 2023, la relación entre acceso a internet y desempeño en las pruebas ICFES se ha mantenido estable y débil. Las líneas de tendencia por año son similares y casi horizontales, lo que sugiere que, a pesar de posibles mejoras en conectividad, no ha habido un cambio notable en su impacto sobre el rendimiento académico.

## Conclusión

El análisis de los municipios de Cundinamarca evidencia que el desempeño en las pruebas Saber 11 está influido de forma limitada por factores como el acceso a Internet, la cobertura

educativa y la deserción escolar. Si bien una mayor cobertura y aprobación escolar se asocian moderadamente con mejores resultados, la conectividad y la deserción muestran relaciones débiles con el rendimiento académico. Además, persisten desigualdades significativas entre municipios, lo que subraya la necesidad de políticas educativas más integrales y focalizadas que aborden no solo el acceso, sino también la calidad de la educación y las condiciones contextuales que afectan el aprendizaje.

### **3. Selección de técnicas de aprendizaje**

Para realizar con satisfacción este apartado de manera eficaz y eficiente, se hizo valoraron varias técnicas de aprendizaje, tanto supervisado como no supervisado, en cuanto a los supervisados se tuvo en cuenta la regresión logística, random forest, SVM y K-NN, teniendo en cuenta el tamaño de la base de datos, tiempo y los objetivos que se aclararon previamente en el primer documento, se optó por usar random forest.

Bajo las mismas métricas que se tomaron en cuenta para la supervisada, la no supervisada, los posibles algoritmos fueron planteados, K-mean clustering, mapas autogenerados, DBSCAN, finalmente se decidió usar K-mean clustering.

### **4. Preparación de datos para el modelado**

Para preparar la base de datos, primero se reunieron todas las fuentes existentes y se integraron en un único conjunto para facilitar la comparación entre variables. Luego, se realizó una limpieza exhaustiva para eliminar valores duplicados o inconsistentes, asegurando que la información estuviera lista para análisis sin sesgos. También se unificaron formatos, como estructuras de datos y fechas, para mantener la coherencia. Posteriormente, se aplicaron pruebas estadísticas y exploraciones para detectar variables con alta correlación; las menos relevantes fueron descartadas para optimizar el rendimiento del análisis. Se hicieron ajustes en los valores para mejorar la interpretabilidad, como escalamiento y codificación de variables categóricas, además de generar nuevas métricas derivadas. Finalmente, se validó la base de datos mediante pruebas con subsets, asegurando que estuviera lista para su uso en modelos o análisis posteriores. Si necesitas más detalles sobre algún aspecto en particular, dime qué parte quieres explorar a fondo.

PERIODO	DEPARTAMENTO	Municipio	CODIGO PLAN	GRADO	TOTAL INSCRI	TOTAL EVALU	CLASIFICACION
2021	antioquia	granada	1,0531E+11	11.0	271.0	267.0	B
2021	antioquia	granada	2,0531E+11	11.0	13.0	12.0	D
2021	antioquia	granada	2,0531E+11	11.0	30.0	30.0	C
2021	antioquia	nariño	1,0548E+11	11.0	84.0	81.0	C
2021	antioquia	nariño	2,0548E+11	11.0	3.0	3.0	SC
2021	antioquia	nariño	2,0548E+11	11.0	55.0	55.0	D
2021	antioquia	nariño	2,0548E+11	11.0	14.0	14.0	D
2021	antioquia	nariño	2,0548E+11	11.0	4.0	4.0	SC
2021	antioquia	nariño	2,0548E+11	11.0	44.0	41.0	D
2021	cundinamarca	beltrán	1,2509E+11	11.0	30.0	29.0	C
2021	cundinamarca	facatativá	1,2527E+11	11.0	388.0	386.0	B
2021	cundinamarca	funza	1,2529E+11	11.0	304.0	300.0	B
2021	cundinamarca	funza	1,2529E+11	11.0	223.0	218.0	A
2021	cundinamarca	funza	1,2529E+11	11.0	172.0	169.0	B
2021	cundinamarca	funza	1,2529E+11	11.0	276.0	271.0	B
2021	cundinamarca	girardot	1,2531E+11	11.0	443.0	437.0	C
2021	cundinamarca	gutierrez	1,2534E+11	11.0	84.0	80.0	D
2021	cundinamarca	gutierrez	2,2534E+11	11.0	13.0	13.0	D
2021	cundinamarca	lenguazaque	1,2541E+11	11.0	147.0	146.0	B
2021	cundinamarca	lenguazaque	2,2541E+11	11.0	34.0	34.0	C
2021	cundinamarca	madrid	1,2543E+11	11.0	480.0	473.0	B
2021	cundinamarca	nocaima	2,2549E+11	11.0	7.0	7.0	SC
2021	cundinamarca	nocaima	1,2549E+11	11.0	60.0	59.0	B
2021	cundinamarca	pacho	1,2551E+11	11.0	146.0	142.0	B
2021	cundinamarca	san bernardo	1,2565E+11	11.0	169.0	165.0	C
2021	cundinamarca	san bernardo	2,2565E+11	11.0	32.0	31.0	C
2021	cundinamarca	san bernardo	2,2565E+11	11.0	12.0	12.0	C

El siguiente paso consistió en normalizar los valores de la base de datos, asegurando que todas las variables tuvieran una escala común para mejorar la precisión de las técnicas que se aplicarían posteriormente. Este proceso permitió reducir posibles sesgos y garantizar que los resultados fueran más fiables y comparables entre sí. Al estandarizar los datos, se optimizó la interpretación de los modelos y se facilitó la extracción de patrones significativos. A continuación, se presenta un ejemplo que ilustra este procedimiento.

I	J	K	L	M	N
NDICE PLC	INDICE PMA	INDICE PSC	INDICE PCN	INDICE PIN	INDICE TOTAL
.708904	0.711411	0.641873	0.652348	0.618718	0.674025
.587412	0.63394	0.559144	0.56819	0.498274	0.580333
.701277	0.705358	0.617326	0.632426	0.535714	0.654221
.678442	0.633899	0.647682	0.642313	0.567074	0.64416
.6284051724137931	0.5920815517241379	0.5835002066	0.5891703446	0.5489836896	0.594490448;
.61848	0.560764	0.567687	0.563373	0.531691	0.574046
.524566	0.4782	0.461367	0.500892	0.465392	0.489267
.6284051724137931	0.5920815517241379	0.5835002066	0.5891703446	0.5489836896	0.594490448;
.582384	0.535553	0.519271	0.535393	0.504986	0.540214
.667527	0.63617	0.62699	0.612637	0.612429	0.634031
.724716	0.714592	0.670899	0.67989	0.676534	0.69591
.737246	0.714464	0.655971	0.677309	0.653712	0.692975
.756319	0.75356	0.702084	0.704439	0.671665	0.724682
.714784	0.708928	0.649431	0.66496	0.649522	0.681833
.719203	0.709182	0.669115	0.67585	0.670223	0.69156
.685543	0.676226	0.611603	0.619598	0.637766	0.647437
.644779	0.622898	0.578374	0.605961	0.531377	0.606724
.600209	0.53826	0.477346	0.568015	0.512459	0.543381
.712376	0.733222	0.643839	0.67871	0.631631	0.68739
.662864	0.693512	0.568034	0.61685	0.621504	0.634253
.726325	0.729454	0.661697	0.67344	0.669564	0.695562
.7005547857142858	0.6746117142857143	0.631775	0.6473280714	0.6130078571	0.659689714;
.724246	0.706007	0.638621	0.663767	0.611078	0.677615
.722855	0.717423	0.660197	0.67342	0.639562	0.689327
.700006	0.679449	0.634944	0.641835	0.630553	0.661481
.668518	0.636769	0.591297	0.656811	0.601282	0.635497
.668518	0.636769	0.591297	0.656811	0.601282	0.635497



Finalmente, se realizó una técnica de análisis de regresión para elegir automáticamente las mejores variables, y las más útiles, que son las más relevantes en nuestro modelo, esto mejora la eficiencia del análisis, a continuación las variables seleccionadas por stepwise.

- **TOTAL EVALUADOS:** Número total de personas evaluadas en el ICFES
- **HOMBRES:** Indica la población total de hombres que presentaron el ICFES.
- **MUJERES:** Indica la población total de mujeres que presentaron el ICFES.
- **POBLACIÓN DANE:** Se refiere a la población total registrada por el DANE en el municipio, donde se tomaron los registros del ICFES.
- **No. ACCESOS FIJO A INTERNET:** Número total de estudiantes que cuentan con internet y que presentaron el examen ICFES.
- **TOTAL INSCRITOS:** Número total de estudiantes registrados para presentar el ICFES.
- **TOTAL:** Suma total de mujeres y hombres que presentaron el ICFES:
- **COBERTURA NETA:** Es el indicador que mide el porcentaje de una población que tiene acceso a internet, en relación con el grupo objetivo total.

## **5. Aplicación técnicas sobre los datos con Mlib**

**Técnica No supervisada:**

**Preprocesamiento de datos para el modelo K mean:**

### **1. Identificación de variables:**

```
variables_cluster = [
    'TOTAL_INSCRITOS', 'TOTAL_EVALUADOS', 'ACCESOS_INTERNET',
    'POBLACION_DANE', 'COBERTURA_NETA', 'APROBACION',
    'REPITENCIA', 'DESERCIÓN'
]
```

En el primer paso también se carga el archivo CSV de municipios de Cundinamarca en un DataFrame de Spark. Posteriormente, se renombran todas las columnas eliminando espacios y caracteres especiales (como tildes y puntos) para garantizar compatibilidad con SQL y evitar errores en operaciones posteriores.

## 2. Método para encontrar el mejor K:

```
# Método del codo: evaluar inercia (WSSSE) para diferentes valores de k
inercia = []
K_range = range(2, 10)

for k in K_range:
    kmeans = KMeans(featuresCol="features", k=k, seed=42)
    modelo = kmeans.fit(df_scaled)
    wssse = modelo.summary.trainingCost # equivalente a inertia_
    inercia.append(wssse)
```

El algoritmo evalúa sistemáticamente desde K=2 hasta K=9 clusters, calculando para cada configuración la inercia (WSSSE - Within Set Sum of Squared Errors) que mide la capacidad interna de los clusters. En cada iteración se entrena un modelo K-means con semilla fija (seed=42) para garantizar reproducibilidad, y se extrae el **trainingCost** que es equivalente a la inercia de scikit-learn. Este proceso permite identificar el punto de inflexión donde agregar un cluster adicional no mejora significativamente la cohesión interna, evitando tanto el sub-agrupamiento (muy pocos clusters que no capturen la diversidad municipal) como el sobre-agrupamiento (demasiados clusters que fragmenten patrones educativos genuinos).

La gráfica resultante muestra la curva de inercia versus número de clusters, donde el "codo" indica el punto óptimo donde la mejora marginal se estabiliza.

## 3. Vectorización y escalamiento:

La vectorización combina las 8 variables individuales en un único vector utilizando VectorAssembler.

Se aplica Standard Scaler que transforma cada variable para tener media cero y desviación estándar uno.

El escalamiento iguala la importancia de todas las variables, permitiendo una comparación justa entre municipios grandes y pequeños, y optimizando la preparación técnica para el algoritmo basado en distancias euclidianas.

## 4. Inicialización Múltiple K-Means:

```
# === K-MEANS CON INICIALIZACION MULTIPLE ===
k = 3
n_inits = 50
best_score = -1
results = []

for seed in range(n_inits):
    kmeans = KMeans(featuresCol="features", k=k, seed=seed)
    model = kmeans.fit(df_scaled)
    predictions = model.transform(df_scaled)
    evaluator = ClusteringEvaluator(featuresCol="features", metricName="silhouette", distanceMeasure="squaredEuclidean")
    score = evaluator.evaluate(predictions)
    results.append((seed, score))

    if score > best_score:
        best_score = score
        best_model = model
        best_predictions = predictions
        best_seed = seed
```

Se ejecutan 50 modelos K-means diferentes (cada uno con semilla aleatoria distinta) para encontrar la configuración que produzca la mejor separación entre grupos de municipios.

Se usa Silhouette Score para medir qué tan bien separados están los clusters. Valores más altos (cercanos a 1) indican mejor separación entre grupos.

## 5. Resultado de la clusterización.

Cluster	Accesos_Internet	Poblacion	Cobertura	Aprobacion	Repitencia	Desercion	Indice_Total
0	128998.0	688156.0	88.73	93.25	3.01	3.55	0.68
1	8078.85	47737.04	93.01	92.97	3.45	3.21	0.66
2	31826.72	170887.63	95.44	91.67	3.63	3.5	0.7

Los resultados revelan tres perfiles municipales (Clusters) claramente diferenciados en Cundinamarca que reflejan la heterogeneidad educativa del departamento.

El Cluster 0 representa municipios grandes y urbanos con alta conectividad (128,998 accesos a internet) y gran población estudiantil (688,156 habitantes), pero con menor cobertura educativa (88.73%) y rendimiento ICFES moderado (0.68), sugiriendo desafíos de saturación y acceso en centros urbanos consolidados.

El Cluster 1 agrupa municipios pequeños y rurales con baja conectividad (8,078 accesos) y población reducida (47,737 habitantes), pero con buena cobertura neta (93.01%) y rendimiento ICFES competitivo (0.66), indicando eficiencia educativa en contextos de recursos limitados.

El Cluster 2 identifica municipios intermedios que logran el equilibrio óptimo con la mejor cobertura educativa (95.44%), menor deserción (3.5%) y el más alto rendimiento ICFES (0.70), representando el modelo aspiracional para el departamento.

## 6. Evaluación del modelo:

Se escogieron 3 métricas enfocadas en los modelos de clusterización no supervisados relacionados a continuación:

- **Silhouette Score:** El Silhouette Score evalúa la calidad del clustering calculando para cada municipio dos distancias clave: la distancia promedio a todos los municipios de su propio cluster (cohesión interna) y la distancia promedio al cluster más cercano (separación externa). La métrica final es el cociente entre estas medidas, normalizando el resultado entre -1 y +1. Un valor positivo indica que el municipio está mejor ubicado en su cluster actual que en cualquier otro, mientras que valores negativos sugieren mala asignación.
- **Calinski-Harabasz:** Esta métrica calcula el ratio entre la varianza entre clusters (qué tan separados están los grupos) y la varianza dentro de cada cluster (qué tan compactos son internamente). Se basa en el análisis de varianza (ANOVA) multivariado, donde valores más altos indican mayor separación entre grupos manteniendo alta cohesión interna. No tiene límite superior teórico, pero valores superiores a 1000 se consideran excelentes en datasets educativos.
- **Davies-Bouldin:** El índice Davies-Bouldin mide la similitud promedio entre cada cluster y su cluster más similar, considerando tanto la distancia entre centroides como la dispersión interna de cada grupo. Se calcula identificando para cada cluster cuál es el más parecido, midiendo el ratio entre la suma de dispersiones internas y la distancia entre centroides. Valores menores indican mejor clustering, siendo 0 el ideal teórico (clusters perfectamente separados sin dispersión interna).

### Técnica supervisada:

Una vez consolidada la base de datos con las variables de interés, se procede a implementar el modelo supervisado de regresión **Random Forest**. Para ello, se utilizan dos versiones del conjunto de datos: la primera corresponde a la base original preprocesada; la segunda incluye una columna adicional denominada “**Cluster**”, la cual fue generada previamente mediante un modelo no supervisado de tipo **K-means**. Esta columna clasifica los municipios en tres grupos según características como tamaño poblacional, cobertura educativa, conectividad y

desempeño en las pruebas ICFES. El objetivo de utilizar ambas versiones es evaluar el impacto que tiene esta variable de segmentación en el rendimiento del modelo predictivo.

```
from pyspark.sql import SparkSession

spark = SparkSession.builder \
    .appName("Proyecto ICFES Cundinamarca") \
    .getOrCreate()

ruta_archivo = "/content/base_reducida.csv"
kmeans_archivo = "/content/base_con_clusters.csv"
df = spark.read.csv(ruta_archivo, header=True, inferSchema=True)
df_kmeans = spark.read.csv(kmeans_archivo, header=True, inferSchema=True)

df = df.withColumnRenamed("No. ACCESOS FIJOS A INTERNET", "ACCESOS_FIJOS_INTERNET")
df_kmeans = df_kmeans.withColumnRenamed("No. ACCESOS FIJOS A INTERNET", "ACCESOS_FIJOS_INTERNET")
df.show(5)
df.printSchema()
df_kmeans.printSchema()
df_kmeans.show(5)
```

## Preprocesamiento de datos para el modelo Random Forest

Antes de entrenar el modelo de Random Forest, es necesario transformar los datos para que sean compatibles con los algoritmos de aprendizaje automático de PySpark. Este proceso incluye la codificación de columnas categóricas, la consolidación de la variable predictora y la creación del conjunto de características (features).

### 1. Identificación de variables:

Se identifican dos variables:

- Variables categóricas: “Departamento”, “Municipio” y “Clasificación”. Estas variables son de tipo texto y deben ser transformadas a formato numérico para poder ser utilizadas por el modelo.
- Variables numéricas: Incluyen datos como “Periodo”, “Grado”, “Total Inscritos”, “Total Evaluados”, “Accesos\_fijos\_internet”, entre otros factores relacionados con conectividad, población y desempeño educativo.

### 2. Codificación de variables categóricas:

Las variables categóricas se procesan en dos etapas:

- StringIndexer: convierte las categorías de texto en índices numéricos.
- OneHotEncoder: aplica codificación one-hot a los índices para evitar que el modelo interprete una relación ordinal inexistente entre las categorías.

### 3. Ensamblaje de características:

Las variables numéricas, junto con las variables categóricas codificadas, se combinan en una sola columna llamada “features” mediante el uso de “VectorAssembler”. Esta columna será la entrada del modelo de aprendizaje automático.

### 4. Creación del Pipeline:

Se construye un “Pipeline” con los siguientes pasos:

- Indexación de las variables categóricas

- Codificación one-hot
- Ensamblaje de características

## 5. Visualización del resultado:

Finalmente, se muestra una vista preliminar del DataFrame preparado, específicamente las columnas features (entradas del modelo) e “INDICE TOTAL” (variable objetivo a predecir).

```
from pyspark.ml.feature import StringIndexer, OneHotEncoder, VectorAssembler
from pyspark.ml import Pipeline

# Variables categóricas
categorical_cols = ["DEPARTAMENTO", "Municipio", "CLASIFICACION"]

# Variables numéricas
numeric_cols = [
    "PERIODO", "GRADO", "TOTAL INSCRITOS", "TOTAL EVALUADOS",
    "ACCESOS_FIJOS_INTERNET", "APROBACIÓN",
    "POBLACIÓN DANE", "DESERCIÓN", "REPROBACIÓN",
    "REPITENCIA", "COBERTURA_NETA"
]

# Paso 1: Indexación
indexers = [StringIndexer(inputCol=col, outputCol=col + "_index") for col in categorical_cols]
encoders = [OneHotEncoder(inputCol=col + "_index", outputCol=col + "_vec") for col in categorical_cols]

# Paso 2: Ensamblar las columnas numéricas y categóricas
feature_cols = [col + "_vec" for col in categorical_cols] + numeric_cols
assembler = VectorAssembler(inputCols=feature_cols, outputCol="features")

# Paso 3: Pipeline
pipeline = Pipeline(stages=indexers + encoders + [assembler])
pipeline_model = pipeline.fit(df)
df_preparado = pipeline_model.transform(df)

# Resultado
df_preparado.select("features", "INDICE TOTAL").show(5, truncate=False)
```

Resultado:

features	INDICE TOTAL
[136, [1, 18, 121, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135], [1.0, 1.0, 1.0, 2021.0, 11.0, 271.0, 267.0, 1884.0, 91.22, 29451.0, 3.89, 4.89, 3.86, 90.62]]	0.67
[136, [1, 18, 120, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135], [1.0, 1.0, 1.0, 2021.0, 11.0, 13.0, 12.0, 1884.0, 91.22, 29451.0, 3.89, 4.89, 3.86, 90.62]]	0.58
[136, [1, 18, 122, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135], [1.0, 1.0, 1.0, 2021.0, 11.0, 30.0, 30.0, 1884.0, 91.22, 29451.0, 3.89, 4.89, 3.86, 90.62]]	0.65
[136, [1, 26, 122, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135], [1.0, 1.0, 1.0, 2021.0, 11.0, 84.0, 81.0, 199.0, 92.8, 5749.0, 3.05, 3.92, 3.03, 71.69]]	0.64
[136, [1, 26, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135], [1.0, 1.0, 1.0, 2021.0, 11.0, 3.0, 3.0, 199.0, 92.8, 5749.0, 3.05, 3.92, 3.03, 71.69]]	0.59

## Entrenamiento y evaluación del modelo Random Forest Regressor

Una vez pre procesados los datos, se procede al entrenamiento y evaluación del modelo supervisado de regresión utilizando el algoritmo de **Random Forest**, el cual es robusto ante ruido y efectivo para manejar relaciones no lineales entre variables.

### 1. División del conjunto de datos

Se divide el conjunto de datos preparado (df\_preparado) en dos subconjuntos:

- **Entrenamiento (70%):** utilizado para construir el modelo.
- **Prueba (30%):** utilizado para evaluar el rendimiento del modelo con datos no vistos.

Esto se hizo con la función randomSplit, fijando una semilla.

### 2. Configuración y entrenamiento del modelo

Se crea una instancia del modelo RandomForestRegressor con los siguientes parámetros:

- featuresCol: columna que contiene las variables predictoras (features).
- labelCol: variable objetivo (INDICE TOTAL).
- predictionCol: nombre de la columna de salida con las predicciones.

- numTrees: número de árboles del bosque (100).
- maxDepth: profundidad máxima de cada árbol (5).
- seed: para reproducibilidad.

El modelo se entrena con el conjunto de datos de entrenamiento utilizando el método `.fit()`.

### 3. Generación de predicciones

El modelo entrenado se aplica sobre el conjunto de prueba mediante el método `.transform()`, generando predicciones que se almacenan en una nueva columna llamada predicción.

### 4. Evaluación del modelo

Se utilizan distintos evaluadores de regresión provistos por `RegressionEvaluator` para medir el rendimiento del modelo:

- **RMSE (Root Mean Squared Error):** mide el error cuadrático medio, penalizando más los errores grandes.
- **MAE (Mean Absolute Error):** mide el error promedio en valor absoluto.
- **MSE (Mean Squared Error):** error cuadrático medio sin aplicar raíz.
- **R<sup>2</sup> (Coeficiente de Determinación):** indica qué porcentaje de la varianza de la variable objetivo es explicada por el modelo (entre 0 y 1, donde 1 es predicción perfecta).

Estas métricas permiten cuantificar la precisión del modelo y verificar su capacidad predictiva sobre datos no vistos.

```

# Paso 1: Dividir en entrenamiento (70%) y prueba (30%)
datos_entrenamiento, datos_prueba = df_preparado.randomSplit([0.7, 0.3], seed=42)

# Paso 2: Crear y entrenar el modelo de Random Forest
rf = RandomForestRegressor(
    featuresCol="features",
    labelCol="INDICE TOTAL",
    predictionCol="prediccion",
    numTrees=100,
    maxDepth=5,
    seed=42
)

modelo_rf = rf.fit(datos_entrenamiento)

# Paso 3: Hacer predicciones sobre los datos de prueba
predicciones = modelo_rf.transform(datos_prueba)

# Paso 4: Evaluar el modelo
evaluator_rmse = RegressionEvaluator(
    labelCol="INDICE TOTAL", predictionCol="prediccion", metricName="rmse"
)
evaluator_mae = RegressionEvaluator(
    labelCol="INDICE TOTAL", predictionCol="prediccion", metricName="mae"
)
evaluator_mse = RegressionEvaluator(
    labelCol="INDICE TOTAL", predictionCol="prediccion", metricName="mse"
)
evaluator_r2 = RegressionEvaluator(
    labelCol="INDICE TOTAL", predictionCol="prediccion", metricName="r2"
)

# Resultados
rmse = evaluator_rmse.evaluate(predicciones)
mae = evaluator_mae.evaluate(predicciones)
mse = evaluator_mse.evaluate(predicciones)
r2 = evaluator_r2.evaluate(predicciones)

# Imprimir resultados
print(f"RMSE: {rmse:.4f}")
print(f"MAE: {mae:.4f}")
print(f"MSE: {mse:.4f}")
print(f"R²: {r2:.4f}")

```

## Visualización de resultados: Predicciones vs Valores reales

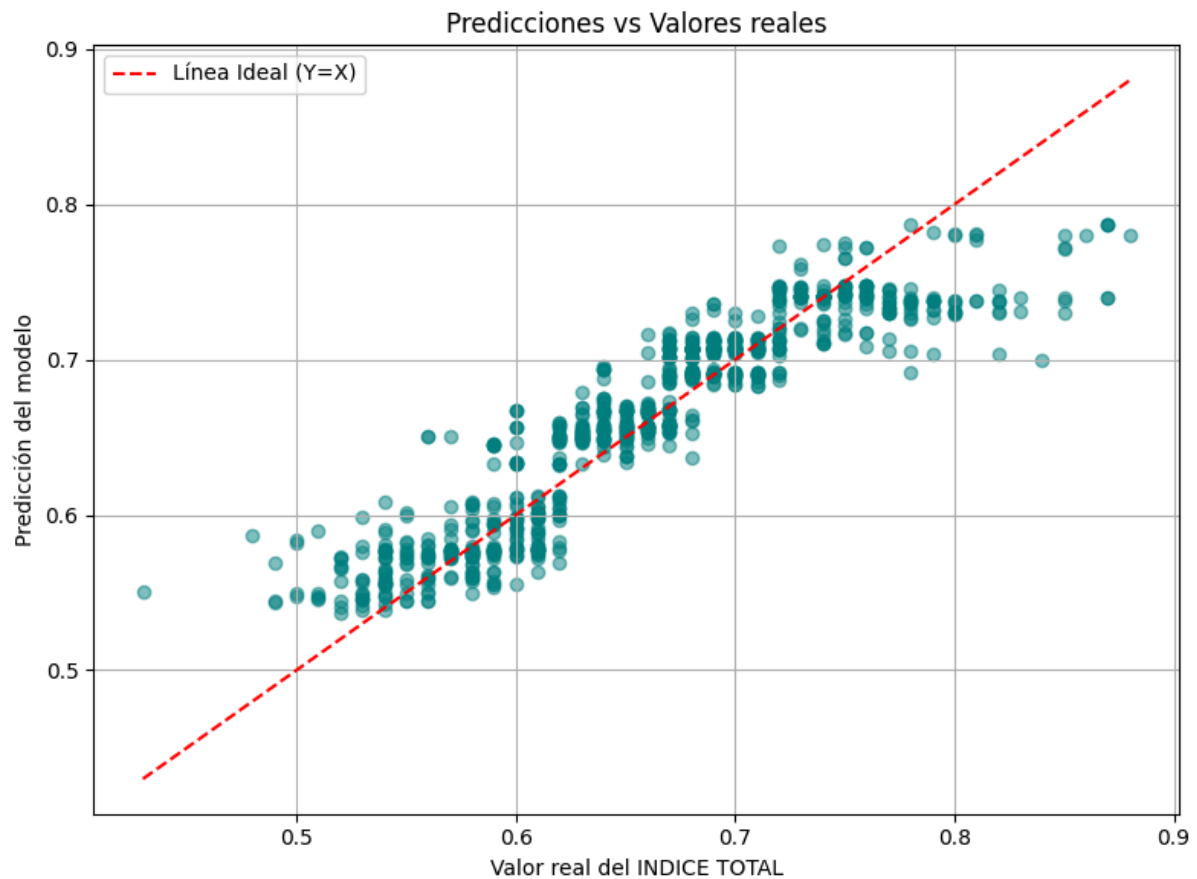
Para complementar la evaluación cuantitativa del modelo, se generó una visualización que compara gráficamente los valores reales del INDICE TOTAL con las predicciones generadas por el modelo de Random Forest.

### Descripción del gráfico:

Se utilizó un diagrama de dispersión donde:



- El eje **X** representa los valores reales del INDICE TOTAL.
- El eje **Y** muestra las predicciones del modelo.
- Cada punto del gráfico corresponde a un ejemplo del conjunto de prueba.
- Se trazó una línea roja discontinua correspondiente a la función identidad ( $Y = X$ ), la cual representa una predicción perfecta.



La mayoría de los puntos se encuentran cercanos a la línea ideal, lo que indica que las predicciones se aproximan bien a los valores reales. Este comportamiento visual apoya los buenos resultados obtenidos en las métricas de evaluación ( $RMSE = 0.0303$  y  $R^2 = 0.8372$ ).

Este mismo procedimiento se repitió con la base de datos con la columna de cluster:

```

datos_entrenamiento_k, datos_prueba_k = df_preparado_k.randomSplit([0.7, 0.3], seed=55)

rf_k = RandomForestRegressor(
    featuresCol="features",
    labelCol="INDICE TOTAL",
    predictionCol="prediccion",
    numTrees=100,
    maxDepth=5,
    seed=42
)

modelo_rf_k = rf_k.fit(datos_entrenamiento_k)

predicciones_k = modelo_rf_k.transform(datos_prueba_k)

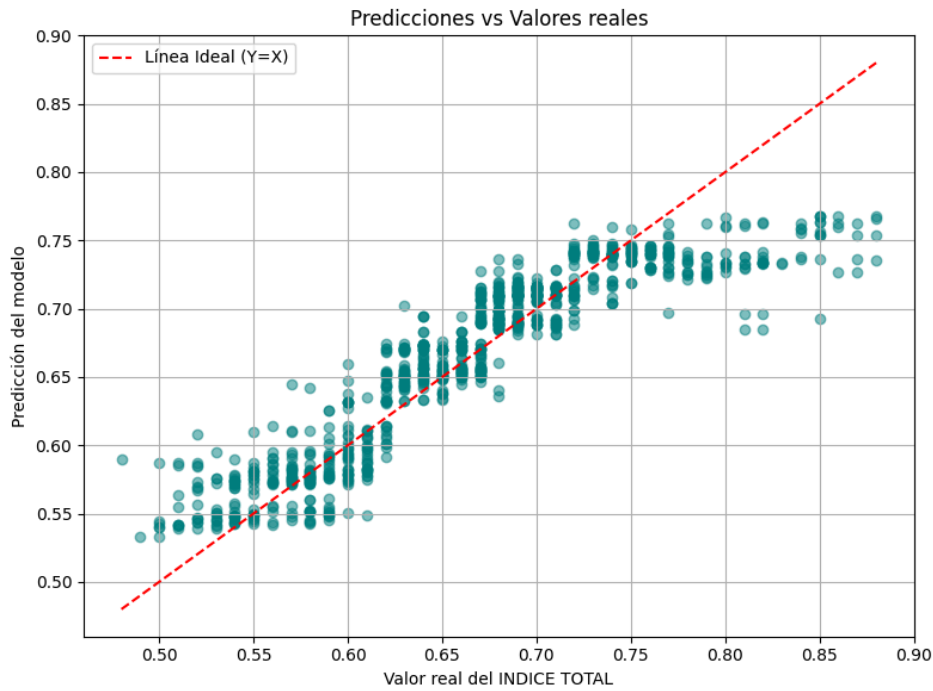
evaluator_rmse_k = RegressionEvaluator(
    labelCol="INDICE TOTAL", predictionCol="prediccion", metricName="rmse"
)
evaluator_mae_k = RegressionEvaluator(
    labelCol="INDICE TOTAL", predictionCol="prediccion", metricName="mae"
)
evaluator_mse_k = RegressionEvaluator(
    labelCol="INDICE TOTAL", predictionCol="prediccion", metricName="mse"
)
evaluator_r2_k = RegressionEvaluator(
    labelCol="INDICE TOTAL", predictionCol="prediccion", metricName="r2"
)

rmse_k = evaluator_rmse_k.evaluate(predicciones_k)
mae_k = evaluator_mae_k.evaluate(predicciones_k)
mse_k = evaluator_mse_k.evaluate(predicciones_k)
r2_k = evaluator_r2_k.evaluate(predicciones_k)

print(f"RMSE: {rmse_k:.4f}")
print(f"MAE: {mae_k:.4f}")
print(f"MSE: {mse_k:.4f}")
print(f"R²: {r2_k:.4f}")

```

Resultados:



Tras comparar el desempeño del modelo de Random Forest utilizando dos versiones del conjunto de datos, una con y otra sin la variable Cluster generada mediante K-mean, se concluye que no se evidencian diferencias significativas en los resultados de las métricas de evaluación. El modelo mantuvo un rendimiento estable y consistente en ambos casos, logrando predecir el INDICE TOTAL con buena precisión. Por lo tanto, la inclusión de Cluster no deteriora el rendimiento, pero tampoco genera una mejora en la capacidad predictiva del modelo.




## 6. Aplicación técnicas sobre los datos con Mlib sobre el cluster:

Se inicia jupyter lab con el comando:

```
(base) [estudiante@NPDG24 almacén]$ jupyter lab --no-browser --ip=10.43.101.135 --port=8080
[I 2025-05-25 22:26:59.936 ServerApp] jupyter_lsp | extension was successfully linked.
[I 2025-05-25 22:26:59.936 ServerApp] jupyter_server_terminals | extension was successfully linked.
[I 2025-05-25 22:26:59.941 ServerApp] jupyterlab | extension was successfully linked.
[I 2025-05-25 22:26:59.947 ServerApp] notebook | extension was successfully linked.
[I 2025-05-25 22:27:00.207 ServerApp] notebook_shim | extension was successfully linked.
[I 2025-05-25 22:27:00.228 ServerApp] notebook_shim | extension was successfully loaded.
[I 2025-05-25 22:27:00.230 ServerApp] jupyter_lsp | extension was successfully loaded.
[I 2025-05-25 22:27:00.231 ServerApp] jupyter_server_terminals | extension was successfully loaded.
[I 2025-05-25 22:27:00.232 LabApp] JupyterLab extension loaded from /home/estudiante/miniconda3/lib/python3.12/site-packages/jupyterlab
[I 2025-05-25 22:27:00.232 LabApp] JupyterLab application directory is /home/estudiante/miniconda3/share/jupyter/lab
[I 2025-05-25 22:27:00.233 LabApp] Extension Manager is 'pypi'.
[I 2025-05-25 22:27:00.302 ServerApp] jupyterlab | extension was successfully loaded.
[I 2025-05-25 22:27:00.306 ServerApp] notebook | extension was successfully loaded.
[I 2025-05-25 22:27:00.306 ServerApp] The port 8080 is already in use, trying another port.
[I 2025-05-25 22:27:00.307 ServerApp] The port 8081 is already in use, trying another port.
[I 2025-05-25 22:27:00.307 ServerApp] Serving notebooks from local directory: /almacen
[I 2025-05-25 22:27:00.307 ServerApp] Jupyter Server 2.15.0 is running at:
[I 2025-05-25 22:27:00.307 ServerApp] http://10.43.101.135:8082/lab?token=853ad543a0f8130771e16ebe7d5c99d68a6df66a7d611752
[I 2025-05-25 22:27:00.307 ServerApp] http://127.0.0.1:8082/lab?token=853ad543a0f8130771e16ebe7d5c99d68a6df66a7d611752
[I 2025-05-25 22:27:00.307 ServerApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 2025-05-25 22:27:00.311 ServerApp]

To access the server, open this file in a browser:
file:///home/estudiante/.local/share/jupyter/runtime/jpserver-1810547-open.html
Or copy and paste one of these URLs:
http://10.43.101.135:8082/lab?token=853ad543a0f8130771e16ebe7d5c99d68a6df66a7d611752
http://127.0.0.1:8082/lab?token=853ad543a0f8130771e16ebe7d5c99d68a6df66a7d611752
[I 2025-05-25 22:27:00.331 ServerApp] Skipped non-installed server(s): bash-language-server, dockerfile-language-server-nodejs, javascript-typescript-languageserver, jedi-language-server, julia-language-server, pyright, python-language-server, python-lsp-server, r-language-server, sql-language-server, texlab, typescript-language-server, unified-language-server, vscode-css-languageserver-bin, vscode-html-languageserver-bin, vscode-javascript-language-server-bin, yaml-language-server
[W 2025-05-25 22:27:11.936 LabApp] Could not determine jupyterlab build status without nodejs
```

Se usó jupyter lab para ejecutar los archivos en el cluster, luego se cargaron los archivos de datos y los de códigos a una carpeta llamada proyecto2:

/ proyecto2 /	
Name	Modified
 base_con_clusters.csv	7 min. ago
 base_reducida.csv	18 min. ago
 ModeloSuper_RF.ipynb	3 min. ago

Se instaló el notebook en el cluster:

```
(base) [estudiante@NPDG24 proyecto2]$ pip install notebook
Requirement already satisfied: notebook in /home/estudiante/miniconda3/lib/python3.12/site-packages (7.3.3)
Requirement already satisfied: jupyter-server<3,>=2.4.0 in /home/estudiante/miniconda3/lib/python3.12/site-packages (from notebook) (2.15.0)
Requirement already satisfied: jupyterlab-server<3,>=2.27.1 in /home/estudiante/miniconda3/lib/python3.12/site-packages (from notebook) (2.27.3)
Requirement already satisfied: jupyterlab<4.4,>=4.3.6 in /home/estudiante/miniconda3/lib/python3.12/site-packages (from notebook) (4.3.6)
Requirement already satisfied: notebook-shim<0.3,>=0.2 in /home/estudiante/miniconda3/lib/python3.12/site-packages (from notebook) (0.2.4)
Requirement already satisfied: tornado<6.2.0 in /home/estudiante/miniconda3/lib/python3.12/site-packages (from notebook) (6.4.2)
Requirement already satisfied: anyio<3.1.0 in /home/estudiante/miniconda3/lib/python3.12/site-packages (from jupyter-server<3,>=2.4.0->notebook) (4.8.0)
Requirement already satisfied: argon2-cffi<=21.1 in /home/estudiante/miniconda3/lib/python3.12/site-packages (from jupyter-server<3,>=2.4.0->notebook) (23.1.0)
Requirement already satisfied: Jinja2<=3.0.3 in /home/estudiante/miniconda3/lib/python3.12/site-packages (from jupyter-server<3,>=2.4.0->notebook) (3.1.6)
Requirement already satisfied: jupyter-client<=7.4.4 in /home/estudiante/miniconda3/lib/python3.12/site-packages (from jupyter-server<3,>=2.4.0->notebook) (8.6.3)
Requirement already satisfied: jupyter-core<=5.0.*,>=4.12 in /home/estudiante/miniconda3/lib/python3.12/site-packages (from jupyter-server<3,>=2.4.0->notebook) (5.7.2)
Requirement already satisfied: jupyter-events<=0.11.0 in /home/estudiante/miniconda3/lib/python3.12/site-packages (from jupyter-server<3,>=2.4.0->notebook) (0.12.0)
Requirement already satisfied: jupyter-server-terminals<=0.4.4 in /home/estudiante/miniconda3/lib/python3.12/site-packages (from jupyter-server<3,>=2.4.0->notebook) (0.5.3)
Requirement already satisfied: nbconvert<=6.4.4 in /home/estudiante/miniconda3/lib/python3.12/site-packages (from jupyter-server<3,>=2.4.0->notebook) (7.16.6)
Requirement already satisfied: nbformat<=5.3.0 in /home/estudiante/miniconda3/lib/python3.12/site-packages (from jupyter-server<3,>=2.4.0->notebook) (5.10.4)
Requirement already satisfied: overrides<=5.0 in /home/estudiante/miniconda3/lib/python3.12/site-packages (from jupyter-server<3,>=2.4.0->notebook) (7.7.0)
Requirement already satisfied: packaging<=22.0 in /home/estudiante/miniconda3/lib/python3.12/site-packages (from jupyter-server<3,>=2.4.0->notebook) (24.2)
Requirement already satisfied: prometheus-client<=0.9 in /home/estudiante/miniconda3/lib/python3.12/site-packages (from jupyter-server<3,>=2.4.0->notebook) (0.21.1)
Requirement already satisfied: pyzmq<=24 in /home/estudiante/miniconda3/lib/python3.12/site-packages (from jupyter-server<3,>=2.4.0->notebook) (26.2.0)
Requirement already satisfied: send2trash<=1.8.2 in /home/estudiante/miniconda3/lib/python3.12/site-packages (from jupyter-server<3,>=2.4.0->notebook) (1.8.3)
Requirement already satisfied: terminado<=0.8.3 in /home/estudiante/miniconda3/lib/python3.12/site-packages (from jupyter-server<3,>=2.4.0->notebook) (0.18.1)
Requirement already satisfied: traitlets<=5.6.0 in /home/estudiante/miniconda3/lib/python3.12/site-packages (from jupyter-server<3,>=2.4.0->notebook) (5.14.3)
Requirement already satisfied: websocket-client<=1.7 in /home/estudiante/miniconda3/lib/python3.12/site-packages (from jupyter-server<3,>=2.4.0->notebook) (1.8.0)
Requirement already satisfied: async-lru<=1.0.0 in /home/estudiante/miniconda3/lib/python3.12/site-packages (from jupyterlab<4.4,>=4.3.6->notebook) (2.0.5)
Requirement already satisfied: httpx<=0.25.0 in /home/estudiante/miniconda3/lib/python3.12/site-packages (from jupyterlab<4.4,>=4.3.6->notebook) (0.28.1)
Requirement already satisfied: ipykernel<=6.5.0 in /home/estudiante/miniconda3/lib/python3.12/site-packages (from jupyterlab<4.4,>=4.3.6->notebook) (6.29.5)
Requirement already satisfied: jupyter-lsp<=2.0.0 in /home/estudiante/miniconda3/lib/python3.12/site-packages (from jupyterlab<4.4,>=4.3.6->notebook) (2.2.5)
Requirement already satisfied: setuptools<=40.8.0 in /home/estudiante/miniconda3/lib/python3.12/site-packages (from jupyterlab<4.4,>=4.3.6->notebook) (75.8.0)
Requirement already satisfied: babel<=2.10 in /home/estudiante/miniconda3/lib/python3.12/site-packages (from jupyterlab-server<3,>=2.27.1->notebook) (2.17.0)
Requirement already satisfied: jsonschema<=4.18.0 in /home/estudiante/miniconda3/lib/python3.12/site-packages (from jupyterlab-server<3,>=2.27.1->notebook) (4.23.0)
Requirement already satisfied: requests<=2.31 in /home/estudiante/miniconda3/lib/python3.12/site-packages (from jupyterlab-server<3,>=2.27.1->notebook) (2.32.3)
Requirement already satisfied: idna<=2.8 in /home/estudiante/miniconda3/lib/python3.12/site-packages (from anyio<3.1.0->jupyter-server<3,>=2.4.0->notebook) (3.7)
Requirement already satisfied: sniffio<=1.1 in /home/estudiante/miniconda3/lib/python3.12/site-packages (from anyio<3.1.0->jupyter-server<3,>=2.4.0->notebook) (1.3.1)
Requirement already satisfied: typing-extensions<=4.5 in /home/estudiante/miniconda3/lib/python3.12/site-packages (from anyio<3.1.0->jupyter-server<3,>=2.4.0->notebook) (4.12.2)
Requirement already satisfied: argon2-cffi-bindings in /home/estudiante/miniconda3/lib/python3.12/site-packages (from argon2-cffi<=21.1->jupyter-server<3,>=2.4.0->notebook) (21.2.0)
Requirement already satisfied: certifi in /home/estudiante/miniconda3/lib/python3.12/site-packages (from httpx<=0.25.0->jupyterlab<4.4,>=4.3.6->notebook) (2025.1.31)
Requirement already satisfied: httptools<=1.* in /home/estudiante/miniconda3/lib/python3.12/site-packages (from httpx<=0.25.0->jupyterlab<4.4,>=4.3.6->notebook) (1.0.7)
Requirement already satisfied: h11<=0.15,>=0.13 in /home/estudiante/miniconda3/lib/python3.12/site-packages (from httptools<=1.*->httpx<=0.25.0->jupyterlab<4.4,>=4.3.6->notebook) (0.14.0)
Requirement already satisfied: comm<=0.1.1 in /home/estudiante/miniconda3/lib/python3.12/site-packages (from ipykernel<=6.5.0->jupyterlab<4.4,>=4.3.6->notebook) (0.2.2)
```

```
(base) [estudiante@MPDG24 proyecto2]$ jupyter nbconvert --to notebook --execute ModeloSuper_RF.ipynb --output output.ipynb
[NbConvertApp] Converting notebook ModeloSuper_RF.ipynb to notebook
Setting default log level to "WARN".
To adjust logging level use c.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
25/05/25 22:49:20 WARN NativeCodeLoader: Unable to load native-heapoop library for your platform... using builtin-java classes where applicable
25/05/25 22:49:33 WARN SparkStringUtils: Truncated the string representation of a plan since it was too large. This behavior can be adjusted by setting 'spark.sql.debug.maxToStringFields'.
[NbConvertApp] Writing 319886 bytes to output.ipynb
```



```
spark = SparkSession.builder \
    .appName("Proyecto ICIES Cundinamarca") \
    .getOrCreate()

ruta_archivo = "base_reducida.csv"
kmeans_archivo = "base_con_clusters.csv"
df = spark.read.csv(ruta_archivo, header=True, inferSchema=True)
df_kmeans = spark.read.csv(kmeans_archivo, header=True, inferSchema=True)

df = df.withColumnRenamed("No. ACCESOS FÍJOS A INTERNET", "ACCESOS_FIJOS_INTERNET")
df_kmeans = df_kmeans.withColumnRenamed("No. ACCESOS FÍJOS A INTERNET", "ACCESOS_FIJOS_INTERNET")
df.show(60)
df.printSchema()
df_kmeans.printSchema()
df_kmeans.show(5)
```

```
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
25/05/25 22:49:20 WARN NativeCodeLoader: Unable to load native-heapdump library for your platform... using builtin-java classes where applicable
```

```
+-----+-----+
|PERIODO|DEPARTAMENTO|Municipio|CODIGO PLANTEL EDUCATIVO|GRADO|TOTAL INSCRITOS|TOTAL EVALUADOS|CLASIFICACION|INDICE TOTAL|APROBACIÓN|POBLACIÓN DANE|ACCESOS_FIJOS_INTERNET|REPITENCIA|DESE
RCIÓN|(COBERTURA_NETA)|REPROBACIÓN|
+-----+-----+
| 2021 | antioquia | granada | 105313000016 | 11.0 | 271.0 | 267.0 | B | 0.67 | 91.22 | 29451 | 1884 | 3.86 |
3.89 | 90.62 | 4.89 |
| 2021 | antioquia | granada | 205313000215 | 11.0 | 13.0 | 12.0 | D | 0.58 | 91.22 | 29451 | 1884 | 3.86 |
3.89 | 90.62 | 4.89 |
| 2021 | antioquia | granada | 205313000517 | 11.0 | 30.0 | 30.0 | C | 0.65 | 91.22 | 29451 | 1884 | 3.86 |
3.89 | 90.62 | 4.89 |
| 2021 | antioquia | nariño | 105483000347 | 11.0 | 84.0 | 81.0 | C | 0.64 | 92.8 | 5749 | 199 | 3.03 |
3.05 | 71.69 | 3.92 |
| 2021 | antioquia | nariño | 205483000392 | 11.0 | 3.0 | 3.0 | SC | 0.59 | 92.8 | 5749 | 199 | 3.03 |
3.05 | 71.69 | 3.92 |
+-----+-----+
only showing top 5 rows

root
|-- PERIODO: integer (nullable = true)
|-- DEPARTAMENTO: string (nullable = true)
|-- Municipio: string (nullable = true)
|-- CODIGO PLANTEL EDUCATIVO: long (nullable = true)
|-- GRADO: double (nullable = true)
|-- ACCESOS_FIJOS_A_INTERNET: double (nullable = true)
|-- REPITENCIA: double (nullable = true)
|-- DESERCIÓN: double (nullable = true)
|-- COBERTURA_NETA: double (nullable = true)
|-- REPROBACIÓN: double (nullable = true)
```

```
from pyspark.ml.feature import StringIndexer, OneHotEncoder, VectorAssembler
from pyspark.ml import Pipeline

# Variables categóricas
categorical_cols = ["DEPARTAMENTO", "Municipio", "CLASIFICACION"]

# Variables numéricas
numeric_cols = [
    "PERIODO", "GRADO", "TOTAL INSCRITOS", "TOTAL EVALUADOS",
    "ACCESOS_FIJOS_INTERNET", "APROBACIÓN",
    "POBLACIÓN DANE", "DESERCIÓN", "REPROBACIÓN",
    "REPITENCIA", "COBERTURA_NETA"
]

# Paso 1: Indexación
indexers = [StringIndexer(inputCol=col, outputCol=col + "_index") for col in categorical_cols]
encoders = [OneHotEncoder(inputCol=col + "_index", outputCol=col + "_vec") for col in categorical_cols]

# Paso 2: Ensamblar las columnas numéricas y categóricas
feature_cols = [col + "_vec" for col in categorical_cols] + numeric_cols
assembler = VectorAssembler(inputCols=feature_cols, outputCol="features")

# Paso 3: Pipeline
pipeline = Pipeline(stages=indexers + encoders + [assembler])
pipeline_model = pipeline.fit(df)
df_preparado = pipeline_model.transform(df)

# Resultado
df_preparado.select("features", "INDICE TOTAL").show(5, truncate=False)
```

```
25/05/25 22:49:33 WARN SparkStringUtils: Truncated the string representation of a plan since it was too large. This behavior can be adjusted by setting 'spark.sql.debug.maxToStringField
s'.
```

```
+-----+-----+
|features|INDICE TOTAL|
+-----+-----+
|[136],[1,18,121,125,126,127,128,129,130,131,132,133,134,135],[1,0,1,0,1,0,2021,0,11,0,271,0,267,0,1884,0,91,22,29451,0,3,89,4,89,3,86,90,62]]|0.67|
|[136],[1,18,120,125,126,127,128,129,130,131,132,133,134,135],[1,0,1,0,1,0,2021,0,11,0,13,0,12,0,1884,0,91,22,29451,0,3,89,4,89,3,86,90,62]]|0.58|
|[136],[1,18,122,125,126,127,128,129,130,131,132,133,134,135],[1,0,1,0,1,0,2021,0,11,0,30,0,30,0,1884,0,91,22,29451,0,3,89,4,89,3,86,90,62]]|0.65|
|[136],[1,26,122,125,126,127,128,129,130,131,132,133,134,135],[1,0,1,0,1,0,2021,0,11,0,84,0,81,0,199,0,92,8,5749,0,3,05,3,92,3,03,71,69]]|0.64|
|[136],[1,26,124,125,126,127,128,129,130,131,132,133,134,135],[1,0,1,0,1,0,2021,0,11,0,3,0,3,0,199,0,92,8,5749,0,3,05,3,92,3,03,71,69]]|0.59|
+-----+-----+
only showing top 5 rows
```



```

from pyspark.ml.regression import RandomForestRegressor
from pyspark.ml.evaluation import RegressionEvaluator

# Paso 1: Dividir en entrenamiento (70%) y prueba (30%)
datos_entrenamiento, datos_prueba = df_preparado.randomSplit([0.7, 0.3], seed=42)

# Paso 2: Crear y entrenar el modelo de Random Forest
rf = RandomForestRegressor(
    featuresCol="features",
    labelCol="INDICE TOTAL",
    predictionCol="prediccion",
    numTrees=100,
    maxDepth=5,
    seed=42
)

modelo_rf = rf.fit(datos_entrenamiento)

# Paso 3: Hacer predicciones sobre los datos de prueba
predicciones = modelo_rf.transform(datos_prueba)

# Paso 4: Evaluar el modelo
evaluator_rmse = RegressionEvaluator(
    labelCol="INDICE TOTAL", predictionCol="prediccion", metricName="rmse"
)
evaluator_mae = RegressionEvaluator(
    labelCol="INDICE TOTAL", predictionCol="prediccion", metricName="mae"
)
evaluator_mse = RegressionEvaluator(
    labelCol="INDICE TOTAL", predictionCol="prediccion", metricName="mse"
)
evaluator_r2 = RegressionEvaluator(
    labelCol="INDICE TOTAL", predictionCol="prediccion", metricName="r2"
)

# Resultados
rmse = evaluator_rmse.evaluate(predicciones)
mae = evaluator_mae.evaluate(predicciones)
mse = evaluator_mse.evaluate(predicciones)
r2 = evaluator_r2.evaluate(predicciones)

# Imprimir resultados
print(f"RMSE: {rmse:.4f}")
print(f"MAE: {mae:.4f}")
print(f"MSE: {mse:.4f}")
print(f"R²: {r2:.4f}")

```

```

RMSE: 0.0303
MAE: 0.0224
MSE: 0.0009
R²: 0.8372

```

```

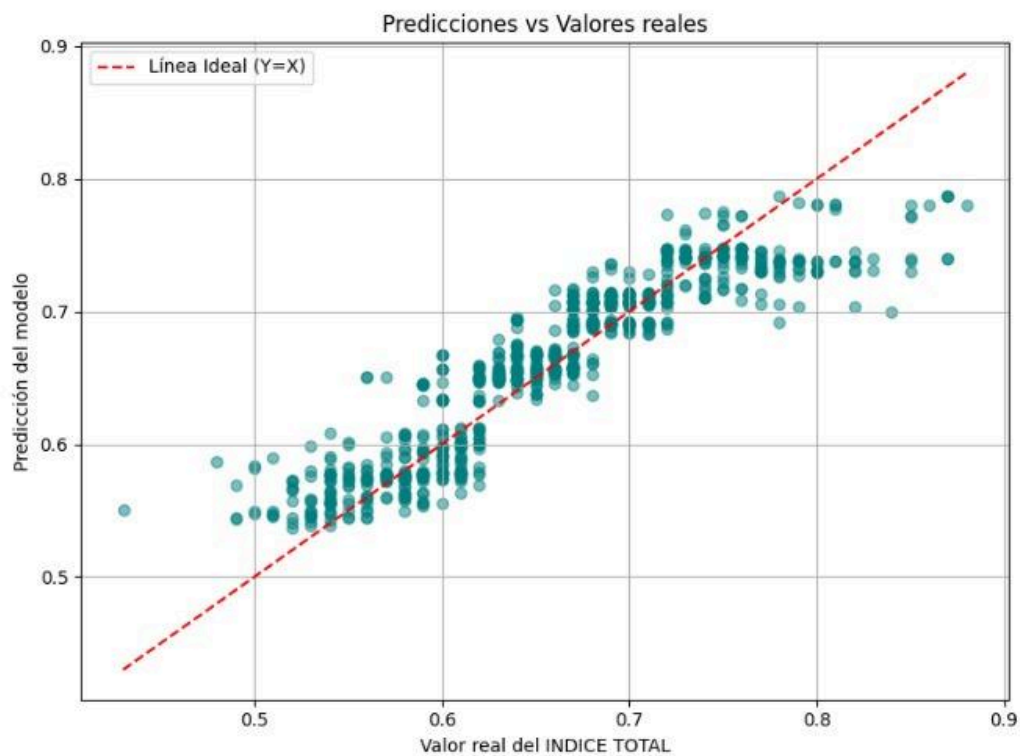
import matplotlib.pyplot as plt

# Convertir a pandas para graficar
preds = predicciones.select("INDICE TOTAL", "prediccion").toPandas()

# Gráfico
plt.figure(figsize=(8, 6))
plt.scatter(preds["INDICE TOTAL"], preds["prediccion"], alpha=0.5, color='teal')
plt.plot([preds["INDICE TOTAL"].min(), preds["INDICE TOTAL"].max()],
         [preds["INDICE TOTAL"].min(), preds["INDICE TOTAL"].max()],
         color='red', linestyle='--', label="Línea Ideal (Y=X)")

plt.title("Predicciones vs Valores reales")
plt.xlabel("valor real del INDICE TOTAL")
plt.ylabel("Predicción del modelo")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()

```



```

from pyspark.sql.functions import col

r2_por_anio = {}

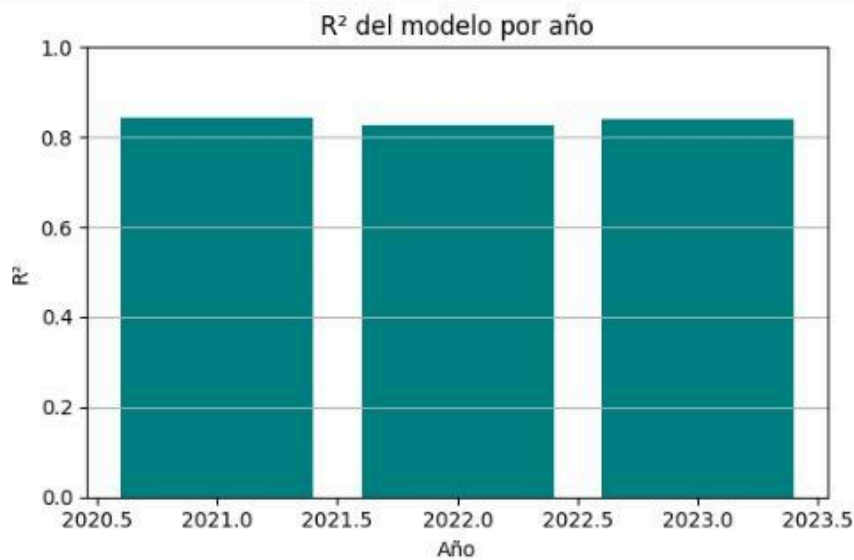
# Evaluador
evaluator_r2 = RegressionEvaluator(labelCol="INDICE TOTAL", predictionCol="prediccion", metricName="r2")

# Iterar por cada año
for anio in [2021, 2022, 2023]:
    df_anio = predicciones.filter(col("PERIODO") == anio)
    r2 = evaluator_r2.evaluate(df_anio)
    r2_por_anio[anio] = r2

# Gráfico
df_r2_anio = pd.DataFrame(list(r2_por_anio.items()), columns=["Año", "R2"])

plt.figure(figsize=(6, 4))
plt.bar(df_r2_anio["Año"], df_r2_anio["R2"], color="teal")
plt.title("R² del modelo por año")
plt.xlabel("Año")
plt.ylabel("R²")
plt.ylim(0, 1)
plt.grid(axis="y")
plt.tight_layout()
plt.show()

```





```

from pyspark.ml.evaluation import RegressionEvaluator

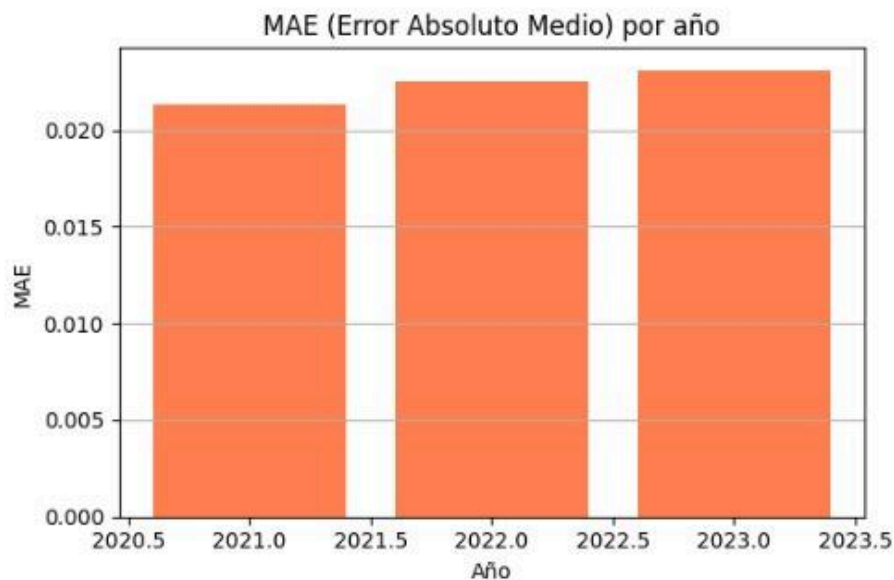
mae_por_anio = {}
evaluator_mae = RegressionEvaluator(labelCol="INDICE TOTAL", predictionCol="prediccion", metricName="mae")

for anio in [2021, 2022, 2023]:
    df_anio = predicciones.filter(col("PERIODO") == anio)
    mae = evaluator_mae.evaluate(df_anio)
    mae_por_anio[anio] = mae

# Gráfico
df_mae_anio = pd.DataFrame(list(mae_por_anio.items()), columns=["Año", "MAE"])

plt.figure(figsize=(6, 4))
plt.bar(df_mae_anio["Año"], df_mae_anio["MAE"], color="coral")
plt.title("MAE (Error Absoluto Medio) por año")
plt.xlabel("Año")
plt.ylabel("MAE")
plt.grid(axis="y")
plt.tight_layout()
plt.show()

```



```

from pyspark.ml.evaluation import RegressionEvaluator

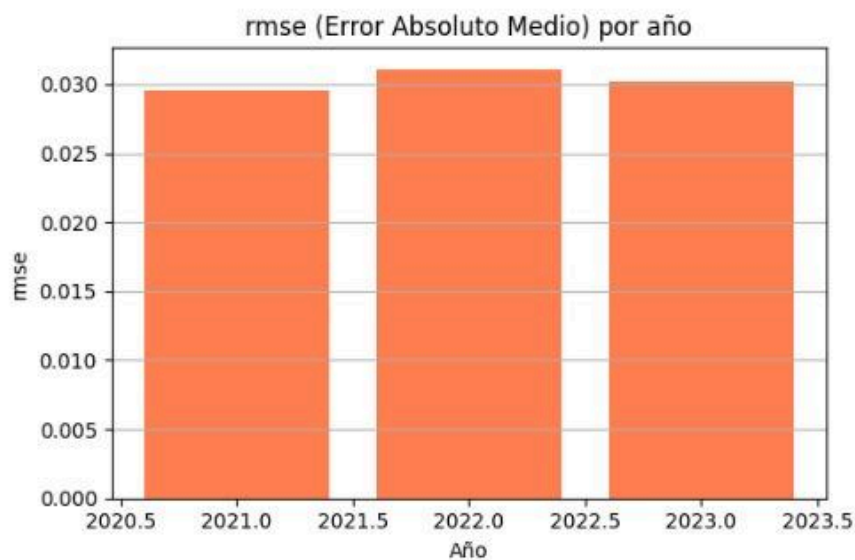
rmse_por_anio = {}
evaluator_rmse = RegressionEvaluator(labelCol="INDICE TOTAL", predictionCol="prediccion", metricName="rmse")

for anio in [2021, 2022, 2023]:
    df_anio = predicciones.filter(col("PERIODO") == anio)
    rmse = evaluator_rmse.evaluate(df_anio)
    rmse_por_anio[anio] = rmse

# Gráfico
df_rmse_anio = pd.DataFrame(list(rmse_por_anio.items()), columns=["Año", "rmse"])

plt.figure(figsize=(6, 4))
plt.bar(df_rmse_anio["Año"], df_rmse_anio["rmse"], color="coral")
plt.title("rmse (Error Absoluto Medio) por año")
plt.xlabel("Año")
plt.ylabel("rmse")
plt.grid(axis="y")
plt.tight_layout()
plt.show()

```



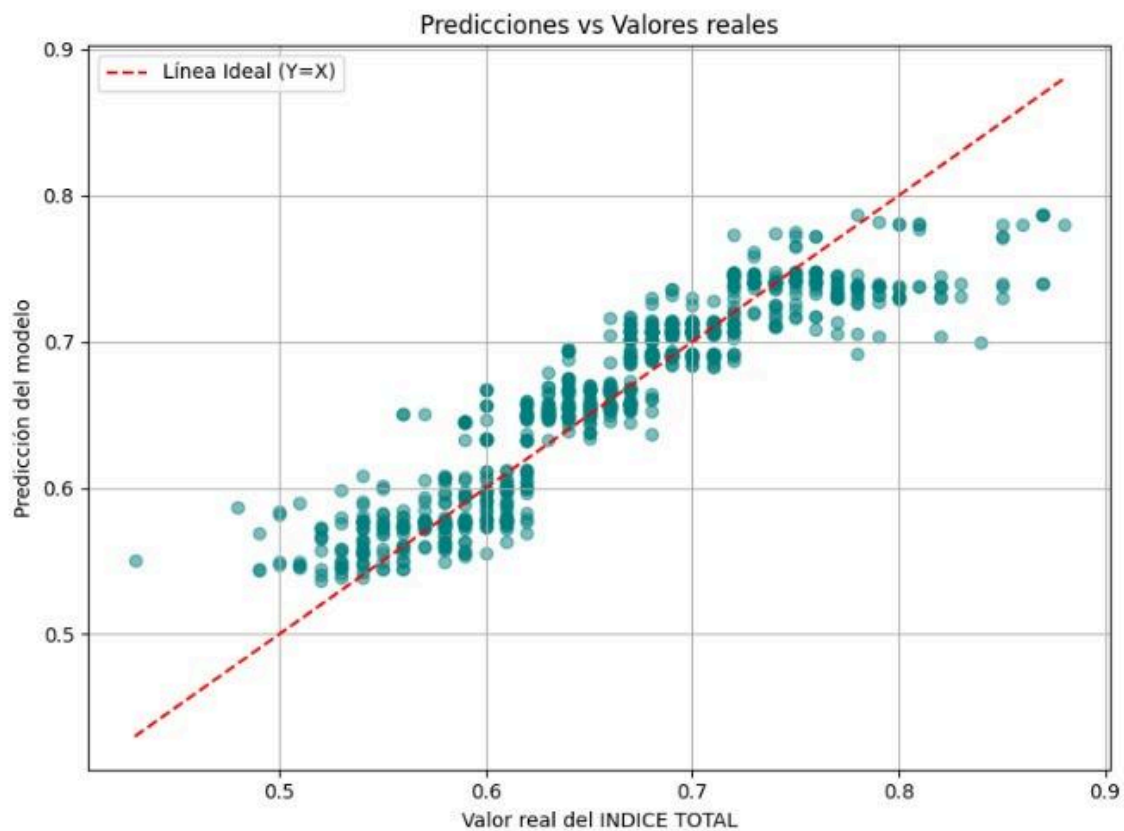
```

# Convertir a pandas para graficar
preds_k = predicciones.select("INDICE TOTAL", "prediccion").toPandas()

# Gráfico
plt.figure(figsize=(8, 6))
plt.scatter(preds_k["INDICE TOTAL"], preds_k["prediccion"], alpha=0.5, color='teal')
plt.plot([preds_k["INDICE TOTAL"].min(), preds_k["INDICE TOTAL"].max()],
         [preds_k["INDICE TOTAL"].min(), preds_k["INDICE TOTAL"].max()],
         color='red', linestyle='--', label="Línea Ideal (Y=X)")

plt.title("Predicciones vs Valores reales")
plt.xlabel("Valor real del INDICE TOTAL")
plt.ylabel("Predicción del modelo")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()

```



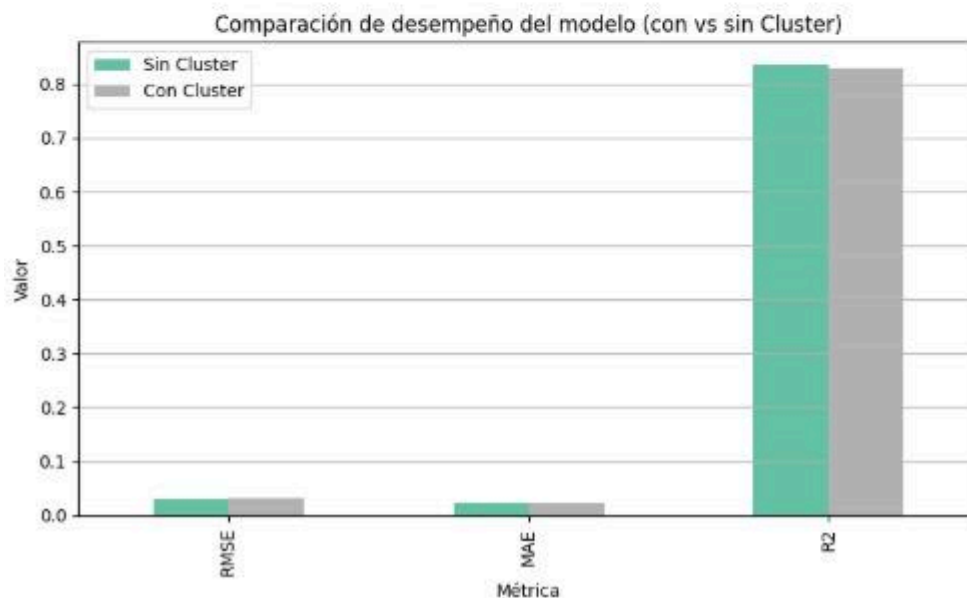
```
# Sin cluster
resultados_sin_cluster = entrenar_y_evaluar(df, incluir_cluster=False)
print("Modelo SIN Cluster:")
for k, v in resultados_sin_cluster.items():
    print(f"{k}: {v:.4f}")

# Con cluster
resultados_con_cluster = entrenar_y_evaluar(df_kmeans, incluir_cluster=True)
print("\nModelo CON Cluster:")
for k, v in resultados_con_cluster.items():
    print(f"{k}: {v:.4f}")
```

```
Modelo SIN Cluster:
RMSE: 0.0303
MAE: 0.0224
R2: 0.8372
```

```
Modelo CON Cluster:
RMSE: 0.0309
MAE: 0.0229
R2: 0.8309
```

```
df_comparacion = pd.DataFrame({
    "Métrica": ["RMSE", "MAE", "R2"],
    "Sin Cluster": [resultados_sin_cluster["RMSE"], resultados_sin_cluster["MAE"], resultados_sin_cluster["R2"]],
    "Con Cluster": [resultados_con_cluster["RMSE"], resultados_con_cluster["MAE"], resultados_con_cluster["R2"]]
})
# Gráfico
df_comparacion.set_index("Métrica").plot(kind="bar", figsize=(8,5), colormap="Set2")
plt.title("Comparación de desempeño del modelo (con vs sin Cluster)")
plt.ylabel("Valor")
plt.grid(axis="y")
plt.tight_layout()
plt.show()
```



Así se puede observar el procedimiento desde la consola local:

```
[I 2025-05-25 22:45:19.531 ServerApp] Uploading file to /proyecto2/base_con_clusters.csv
[W 2025-05-25 22:46:13.657 ServerApp] Notebook proyecto2/ModeloSuper_RF.ipynb is not trusted
[I 2025-05-25 22:46:13.940 ServerApp] Connecting to kernel ec22282c-f425-4050-bdf5-05b17c2c6577.
[I 2025-05-25 22:48:31.763 ServerApp] Saving file at /proyecto2/ModeloSuper_RF.ipynb
[W 2025-05-25 22:48:31.764 ServerApp] Notebook proyecto2/ModeloSuper_RF.ipynb is not trusted
[I 2025-05-25 22:48:32.820 ServerApp] Saving file at /proyecto2/ModeloSuper_RF.ipynb
[W 2025-05-25 22:48:32.821 ServerApp] Notebook proyecto2/ModeloSuper_RF.ipynb is not trusted
[I 2025-05-25 22:48:34.486 ServerApp] Starting buffering for ec22282c-f425-4050-bdf5-05b17c2c6577:70e64a6c-02c9-4a9c-8a90-7c07f7678689
[W 2025-05-25 22:48:35.839 ServerApp] Notebook proyecto2/ModeloSuper_RF.ipynb is not trusted
[I 2025-05-25 22:48:36.165 ServerApp] Connecting to kernel ec22282c-f425-4050-bdf5-05b17c2c6577.
[I 2025-05-25 22:49:12.552 ServerApp] Saving file at /proyecto2/ModeloSuper_RF.ipynb
[W 2025-05-25 22:49:12.553 ServerApp] Notebook proyecto2/ModeloSuper_RF.ipynb is not trusted
[W 2025-05-25 22:50:08.790 ServerApp] Notebook proyecto2/output.ipynb is not trusted
[I 2025-05-25 22:50:09.112 ServerApp] Kernel started: 15110833-f205-47f9-b5d6-811992bdc4c2
[I 2025-05-25 22:50:09.732 ServerApp] Connecting to kernel 15110833-f205-47f9-b5d6-811992bdc4c2.
```

Se repite el mismo proceso pero con el notebook de python del modelo no supervisado K-means.

```
import matplotlib.pyplot as plt
import random as rnd
import pandas as pd
import numpy as np
from sklearn.metrics import silhouette_samples

# Convertir a pandas
df_resultado_kd = df_resultados_teoricos()
df_cluster_id = best_predictions.select('cluster', 'INDICE_TOTAL', 'variables_cluster').toPandas()

# Extraer features y etiquetas
X = df_scaled.select('features').rdd.map(lambda row: row['features']).toarray().collect()
labels = best_predictions.select('cluster').rdd.map(lambda row: row['cluster']).collect()

# Calcular silhouette por muestra
sil_samples = silhouette_samples(np.array(X), np.array(labels))
df_sil = pd.DataFrame({'cluster': labels, 'silhouette': sil_samples})
cluster_means = df_sil.groupby('cluster')['silhouette'].mean()

# Visualización
fig, axs = plt.subplots(2, 2, figsize=(18, 18))

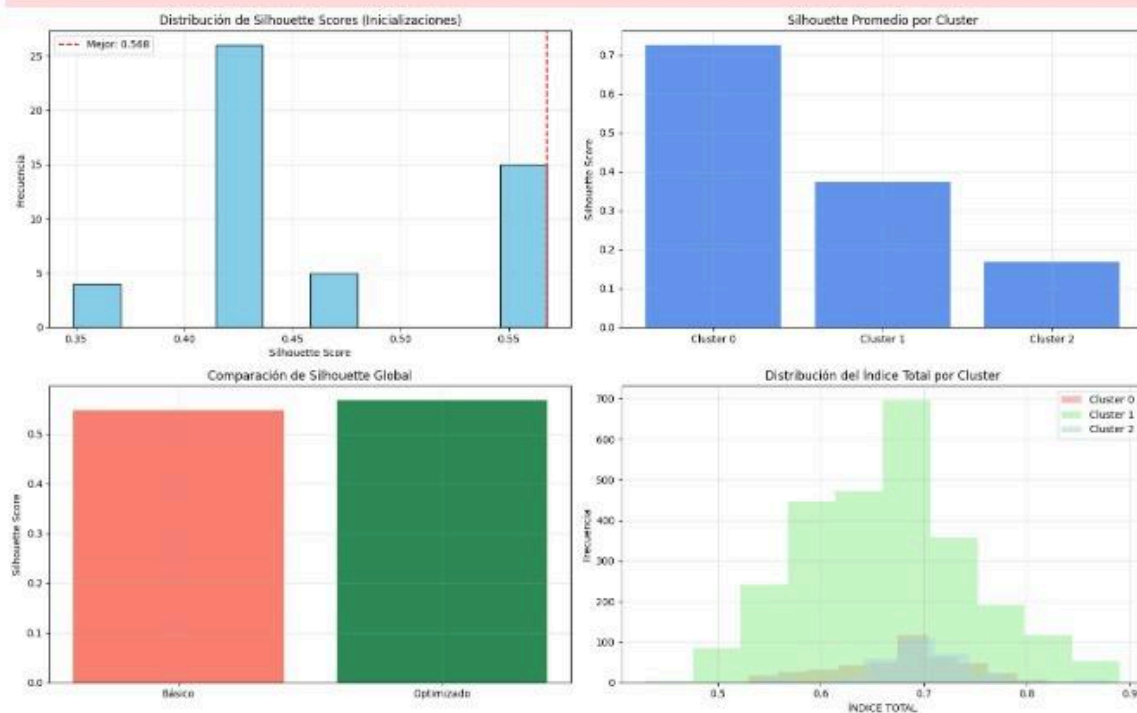
# 1. Histograma de Silhouette scores
axs[0, 0].hist(df_resultado_kd['silhouette'], bins=20, color='teal', edgecolor='black')
axs[0, 0].xaxis.set_label('silhouette', loc='center', color='red', linestyle='--', label=Mejor: (df_resultado_kd['silhouette'].max()-#?))
axs[0, 0].set_title('Distribución de Silhouette Scores (Inicializaciones)')
axs[0, 0].set_xlabel('silhouette score')
axs[0, 0].set_ylabel('frecuencia')
axs[0, 0].legend()
axs[0, 0].grid(True, alpha=0.4)

# 2. Silhouette real por Cluster
axs[0, 1].bar([f'cluster {i}' for i in cluster_means.index], cluster_means.values, color='coralblue')
axs[0, 1].set_title('Silhouette Promedio por Cluster')
axs[0, 1].set_xlabel('silhouette score')
axs[0, 1].grid(True, alpha=0.4)

# 3. Comparación básica vs optimizado
axs[1, 0].bar(['Básico', 'Optimizado'], [silhouette_basico, best_score], color=['salmon', 'seagreen'])
axs[1, 0].set_title('Comparación de Silhouette Global')
axs[1, 0].set_xlabel('silhouette score')
axs[1, 0].grid(True, alpha=0.4)

# 4. Distribución del índice total por cluster
colors = ['lightcoral', 'lightgreen', 'lightblue']
for i in range(df_cluster_id['cluster'].nunique()):
    axs[1, 1].hist(df_cluster_id[df_cluster_id['cluster'] == i]['INDICE_TOTAL'], bins=20,
                  alpha=0.5, label=f'Cluster {i}', color=colors[i % len(colors)])
axs[1, 1].set_title('Distribución del Índice Total por Cluster')
axs[1, 1].set_xlabel('INDICE TOTAL')
axs[1, 1].set_ylabel('frecuencia')
axs[1, 1].legend()
axs[1, 1].grid(True, alpha=0.4)

plt.tight_layout()
plt.show()
```



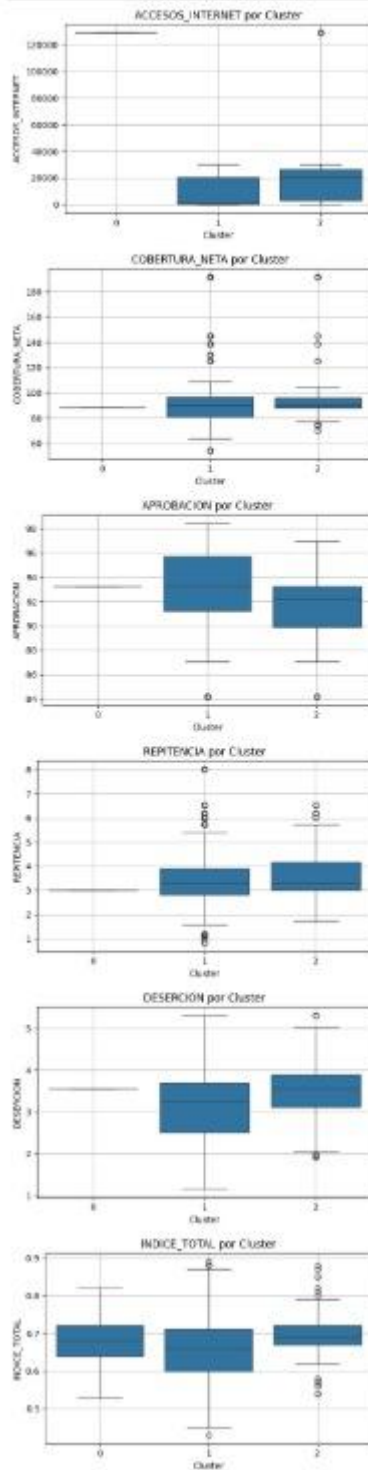
```

variables = [
    "ACCESOS_INTERNET", "COBERTURA_NETA",
    "APROBACION", "REPETENCIA", "DESERCIÓN", "INDICE_TOTAL"
]

# Guardarlo en Pandas
df_clusters_pd = base_predictions_cluster["Cluster", variables].toPandas()

# Ver que se variaban
plt.figure(figsize=(8, 8))
sns.boxplot(data=df_clusters_pd, x="Cluster", y=variables)
plt.xticks(["0", "1", "2"])
plt.grid(True)
plt.show()

```



## 7. Aplicación de métricas:



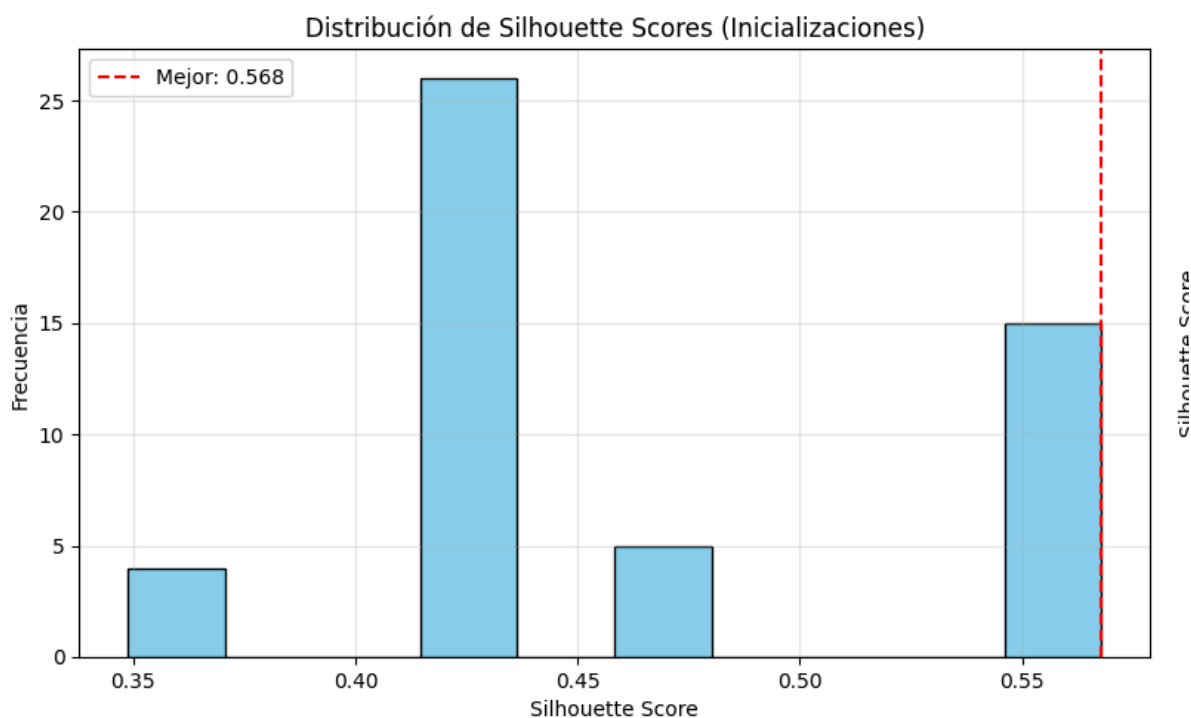
## Métricas modelo no supervisado K-Means:

Resultados obtenidos de la clusterización con K-mean:

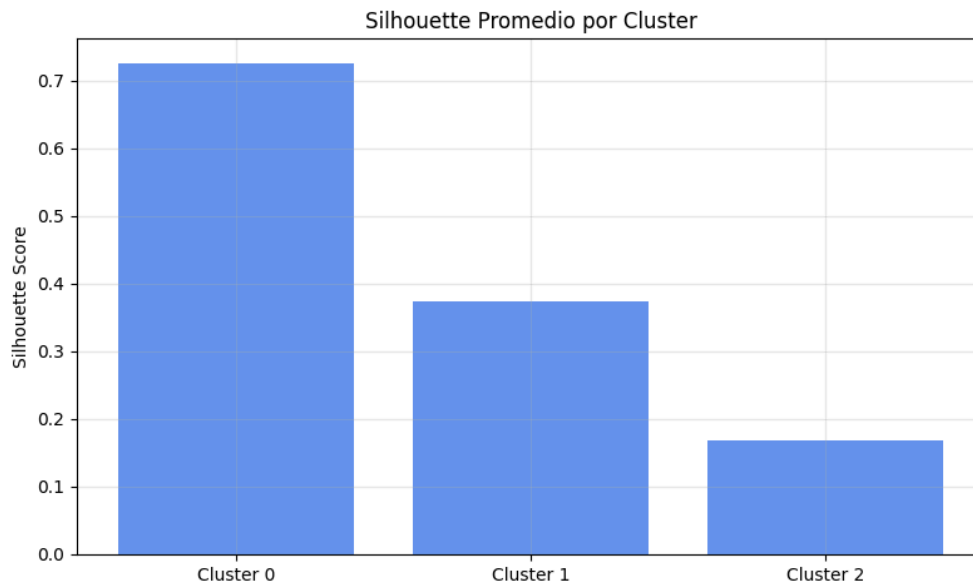
**Silhouette Scor(0.568):** Este puntaje representa **excelente calidad de clustering**. Con 0.568, cada municipio está significativamente más cerca de otros municipios de su mismo grupo que de municipios de grupos diferentes. Esto supera el umbral de 0.5 considerado como "buen clustering" en la literatura académica, indicando que la segmentación captura patrones reales y robustos.

**Calinski-Harabasz (16,872.9):** Este valor **extremadamente alto** indica una separación excepcional entre los 3 clusters de municipios. La cifra significa que la variabilidad entre diferentes grupos de municipios es casi 17,000 veces mayor que la variabilidad promedio dentro de cada grupo. Esto sugiere que los clusters capturan diferencias fundamentales y estructurales entre tipos de municipios.

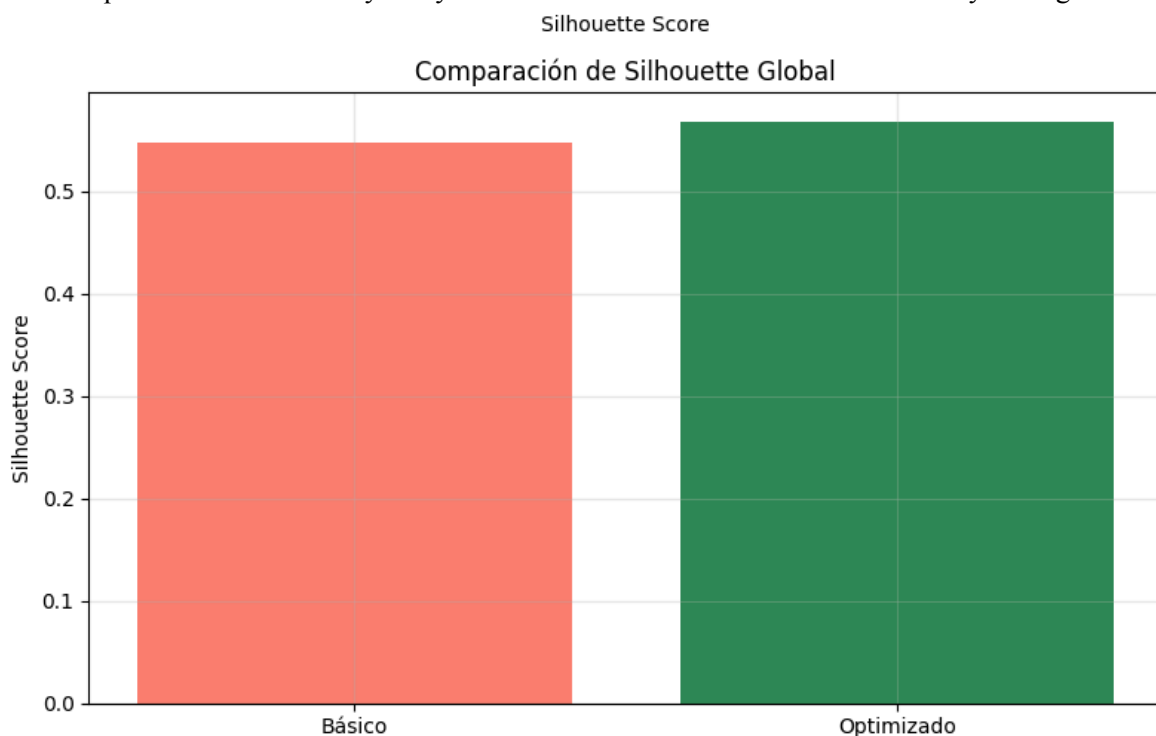
**Davies-Bouldin (1.074):** Con un valor apenas superior a 1.0, este resultado indica **muy buena calidad de clustering**. Significa que cada cluster tiene muy poca superposición con otros clusters, y que la dispersión interna de cada grupo es pequeña comparada con las distancias entre grupos. Un valor cercano a 1 es considerado excelente en aplicaciones prácticas de clustering educativo.



Esta gráfica demuestra la efectividad de la técnica de inicialización múltiple implementada en el clustering. La distribución muestra que la mayoría de las 50 inicializaciones generaron scores entre 0.40-0.42, formando el pico más alto de la distribución, lo que indica un rendimiento base consistente del algoritmo K-means. Sin embargo, la presencia de inicializaciones con scores más bajos (0.35-0.39) y el grupo de scores superiores (0.52-0.55) evidencia la variabilidad inherente del algoritmo debido a la inicialización aleatoria de centroides. La línea roja punteada marca el mejor resultado obtenido (0.568), que se ubica en el extremo superior derecho, confirmando que la estrategia de múltiples inicializaciones fue exitosa para evitar mínimos locales. Esta distribución válida que sin la optimización de inicialización múltiple, el proyecto habría obtenido un clustering de calidad moderada (alrededor de 0.41), pero la técnica aplicada logró identificar una configuración superior que mejora significativamente la identificación de patrones educativos municipales en Cundinamarca.

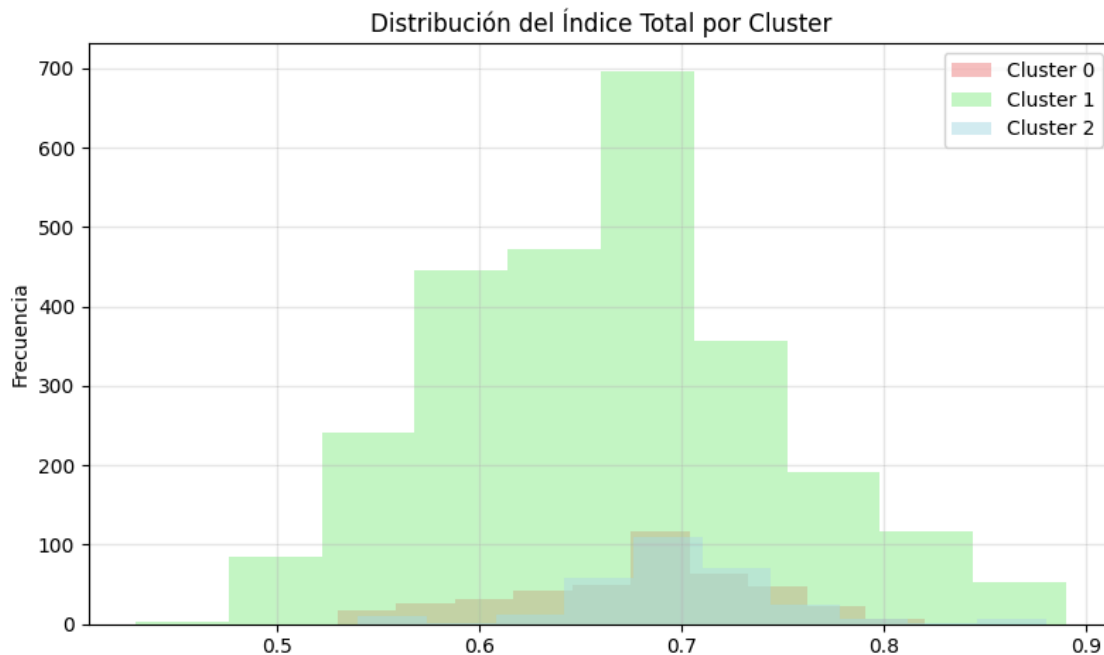


Esta gráfica revela diferencias importantes en la cohesión interna de cada uno de los 3 clusters identificados, proporcionando insights cruciales sobre la calidad de la segmentación municipal. El **Cluster 0** (municipios urbanos grandes) muestra el mejor Silhouette individual ( $\sim 0.72$ ), indicando que estos municipios son muy homogéneos entre sí y claramente diferenciados de otros grupos, lo cual tiene sentido dado que centros urbanos como Soacha, Fusagasugá y similares comparten características demográficas y educativas distintivas. El **Cluster 1** (municipios rurales pequeños) presenta cohesión moderada ( $\sim 0.37$ ), sugiriendo mayor diversidad interna dentro de este grupo, posiblemente debido a que municipios rurales, aunque pequeños, pueden tener variaciones en conectividad, geografía o desarrollo local que los hace menos uniformes. El **Cluster 2** (municipios intermedios) muestra la menor cohesión ( $\sim 0.16$ ), indicando que este grupo es el más heterogéneo internamente, lo cual es esperado ya que municipios "intermedios" representan una categoría de transición que naturalmente incluye mayor variabilidad en características educativas y demográficas.





No se evidencia un gran aumento del rendimiento, sin embargo si se logra optimizar el modelo lo mejor posible nos acercamos a mejores resultados y más precisos.



Esta distribución del rendimiento ICFES por cluster revela patrones educativos fundamentales que confirman la validez y utilidad práctica de la segmentación municipal obtenida. El **Cluster 1** (municipios urbanos grandes) muestra una distribución concentrada principalmente entre 0.6-0.8 con un pico pronunciado alrededor de 0.75, indicando rendimiento educativo relativamente homogéneo y competitivo entre centros urbanos consolidados. El **Cluster 0** (municipios rurales pequeños) presenta una distribución más amplia centrada alrededor de 0.65-0.7, sugiriendo mayor variabilidad en el rendimiento educativo rural, posiblemente debido a diferencias en recursos, acceso a docentes calificados o infraestructura educativa entre municipios pequeños. El **Cluster 2** (municipios intermedios) exhibe la distribución más concentrada y desplazada hacia valores superiores (0.7-0.75), confirmando que estos municipios logran el mejor balance entre recursos disponibles y resultados educativos.

### Métricas modelo supervisado Random Forest

Sin cluster:

```
RMSE: 0.0303
MAE: 0.0224
MSE: 0.0009
R²: 0.8372
```

Con cluster:

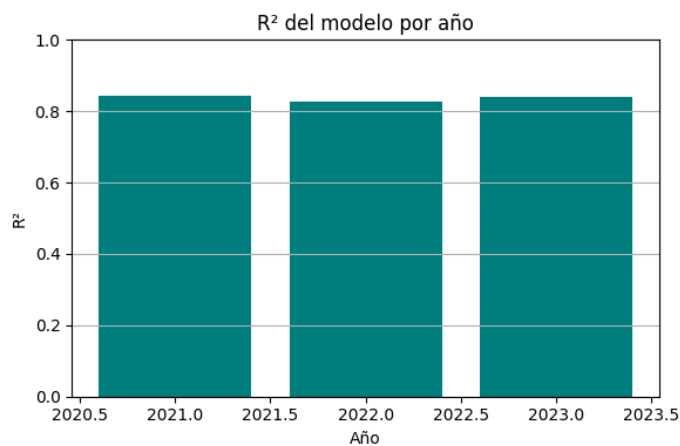
```
RMSE: 0.0330
MAE: 0.0236
MSE: 0.0011
R²: 0.8220
```

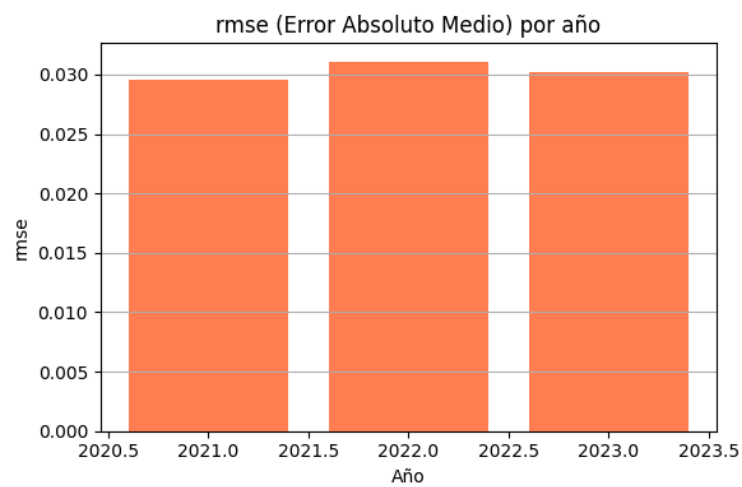
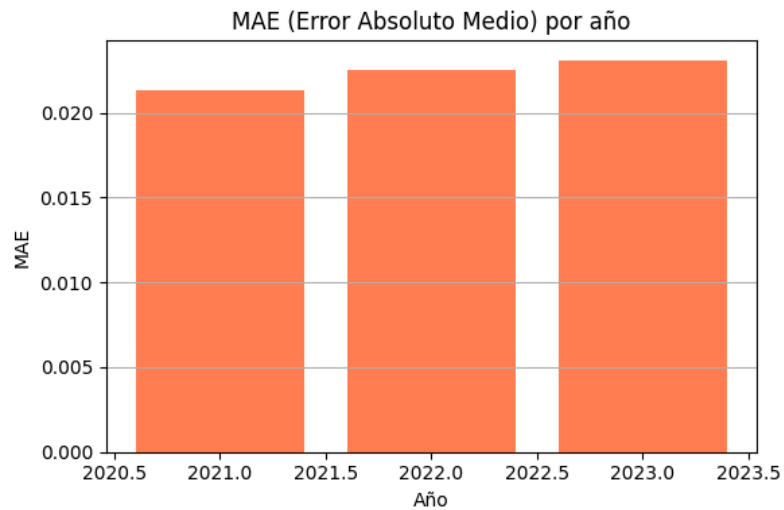
El modelo de Random Forest Regressor fue evaluado utilizando el conjunto de prueba, obteniendo los siguientes resultados:

- **RMSE (Root Mean Squared Error):** 0.0303
- **MAE (Mean Absolute Error):** 0.0224
- **MSE (Mean Squared Error):** 0.0009
- **R<sup>2</sup> (Coeficiente de determinación):** 0.8372

Estos resultados indican un buen desempeño del modelo, con errores promedio bajos y un alto grado de explicación de la variabilidad de la variable objetivo (INDICE TOTAL). En particular, el valor de  $R^2 = 0.8372$  refleja que el modelo explica aproximadamente el 83.72% de la variabilidad en los datos, lo cual es considerado excelente en contextos educativos y sociales donde los datos suelen ser complejos y ruidosos. El bajo valor de RMSE ( $\pm 0.03$ ) también sugiere una buena precisión en las predicciones individuales.

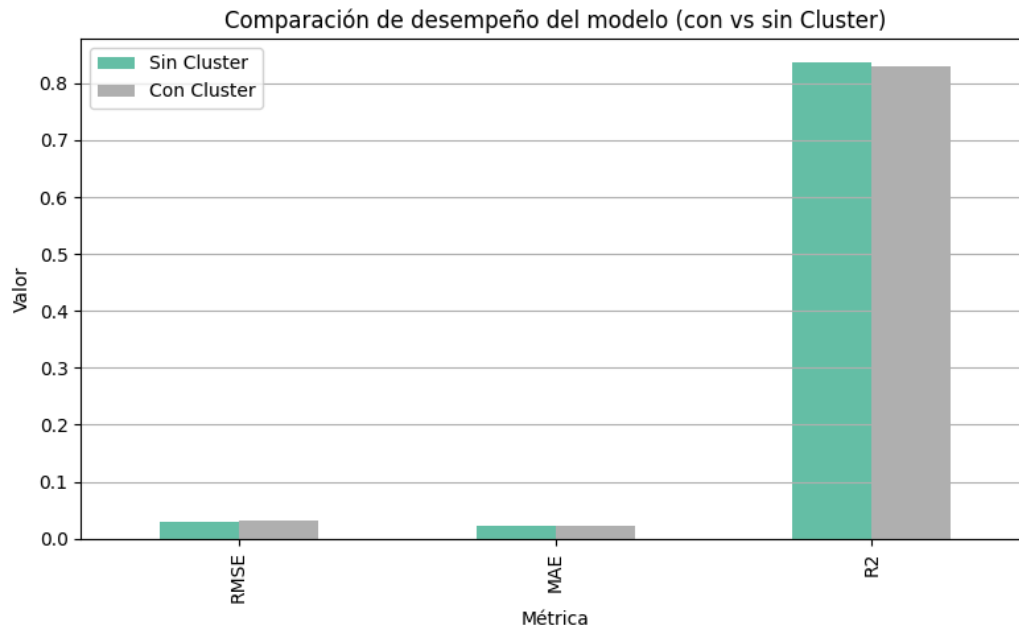
Por otro lado, se graficaron estas métricas pero por periodos, con el fin de identificar si el modelo es estable y si su desempeño varía o se mantiene por año.





En las tres gráficas se observa que el modelo tiene un comportamiento estable en el tiempo, lo cual es clave para confiar en su uso en planes estratégicos a futuro. Indicando que el modelo es de bastante utilidad y bastante fiable.

Por último, se comparó estas métricas en ambas bases de datos, dando como resultado:



RMSE (Root Mean Squared Error) y MAE (Mean Absolute Error) muestran una diferencia mínima entre ambas versiones del modelo. Ambos errores son bajos y prácticamente idénticos, lo que indica que el modelo mantiene su precisión en ambas configuraciones.

$R^2$  (coeficiente de determinación) también se mantiene prácticamente igual, ligeramente por encima de 0.83 en ambos casos, lo que significa que el modelo explica más del 83% de la variabilidad de la variable objetivo sin verse afectado por la inclusión del Cluster.

### **Conclusión:**

La variable Cluster, aunque incorpora un resumen sociodemográfico y educativo de cada municipio, no mejora de manera significativa el rendimiento predictivo del modelo. Esto puede deberse a que la información contenida en Cluster ya está capturada de forma directa en otras variables numéricas utilizadas por el modelo. En consecuencia, el uso de Cluster no es perjudicial, pero tampoco aporta una ventaja predictiva significativa. Sin embargo, sigue siendo una variable valiosa desde un punto de vista interpretativo.

### **Referencias:**

**Breiman, L. (2001).** *Random forests*. *Machine Learning*, 45(1), 5–32.

<https://doi.org/10.1023/A:1010933404324>

**Willmott, C. J., & Matsuura, K. (2005).** *Advantages of the mean absolute error (MAE) over the root mean square error (RMSE) in assessing average model performance*. *Climate Research*, 30(1), 79–82. <https://doi.org/10.3354/cr030079>

**MacQueen, J. (1967). *Some methods for classification and analysis of multivariate observations*. In Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability (Vol. 1, pp. 281–297). University of California Press.**  
**<https://projecteuclid.org/euclid.bsmsp/1200512992>**