“Ejercicio integrador para clase: Análisis y clasificación con dataset Iris”

Introducciòn.

Este informe detalla el análisis de clasificación supervisada y clustering no supervisado realizado con el dataset Iris. Se incluyen los modelos Árbol de Decisión, Random Forest y SVM (kernel RBF) para clasificación, y los algoritmos de clustering K-Means y DBSCAN. Se utiliza solo las dos primeras características del dataset para facilitar la visualización y el análisis.

Preguntas:

Parte 1: Clasificación supervisada
1. Carga el dataset Iris.
2. Divide los datos en conjunto de entrenamiento (70%) y prueba (30%).
3. Entrena y evalúa estos modelos para predecir la especie de iris:
a. Árbol de decisión
b. Random Forest (100 árboles)
c. SVM con kernel RBF
4. Calcula y muestra la precisión de cada modelo en el conjunto de prueba.
5. Visualiza el árbol de decisión entrenado.

Parte 2: Clustering no supervisado
1. Usa solo las dos primeras características para facilitar visualización.
2. Aplica K-Means para encontrar 3 clusters. Visualiza los clusters y sus centroides.
3. Aplica DBSCAN con eps=0.5 y min_samples=5.
Visualiza los clusters y los puntos etiquetados como ruido.
4. Compara los clusters encontrados con las etiquetas reales.

Preguntas para discusión

• ¿Cuál modelo supervisado tuvo mejor precisión? ¿Por qué?.

Conexión Mujeres TIC

Realizado por:
Gina Esther Ortega Palacios

Iris Flower Classification

Project Based Learning

1. Carga el dataset Iris.

```
# Importar las librerías necesarias
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.cluster import KMeans, DBSCAN
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.datasets import make_blobs
from sklearn.metrics import adjusted_rand_score, homogeneity_score
# Cargar el dataset Iris
iris = load_iris()
X = iris.data
y = iris.target
feature_names = iris.feature_names
target_names = iris.target_names
# Convertir a DataFrame de pandas y mostrar las primeras filas
df = pd.DataFrame(X, columns=iris.feature_names)
df['species'] = pd.Categorical.from_codes(y, iris.target_names)
print(df.head())
```

Preguntas para discusiòn

- ¿Cómo se relacionan los clusters de K-Means con las clases reales?
- ¿Qué utilidad tiene DBSCAN en comparación con K Means?
- ¿Qué ventajas y desventajas tiene un árbol de decisión frente a un Random Forest?
- ¿Cómo ayuda la visualización del árbol para interpretar el modelo?

Bonus (opcional)

- Experimenta con diferentes valores de eps y min_samples en DBSCAN.
- Cambia el kernel de SVM a linear y compara resultados.
- Intenta usar todas las características para clustering y observa qué cambia.

Desarrollo

Parte 1: Clasificación Supervisada

1. Carga del dataset Iris: Se utiliza el dataset clásico de Iris que contiene 150 muestras de tres especies (Setosa, Versicolor y Virginica) con 4 características y se importa las librerías necesarias para ejecuciones:

1.A. Código y resultado:

En la siguiente columna veremos los codigos desarrollados y los resultados de las primeras 5 filas a la hora de cargar el data set reconocido Iris y luego convertirlo en Dataframe:

1.B.Resultados obtenidos:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

	species
0	setosa
1	setosa
2	setosa
3	setosa
4	setosa

2. División de datos: Los datos se dividen en un conjunto de entrenamiento y prueba. Se utilizó el conjunto de datos completo de entrenamiento y prueba, con una división 70% - 30% y estratificación para mantener la proporción de clases.

2.A. Codigo y resultados:

2. Divide los datos en conjunto de entrenamiento (70%) y prueba (30%).

```
# 2.  Dividir en entrenamiento y prueba (70% - 30%)
# Dividir datos en entrenamiento y prueba
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, stratify=y)
```

✓ 0.0s python

2.B.Nota sobre stratify=y: A la hora de dividir los datos tuve la duda de colocarlo o no colocarlo en el anterior tramo del código porque sin él, a la hora de evaluar la Precisión del modelo con acurracy e imprimirla este me arrojaba 1.00 pero cuando lo aplico la Precisión del modelo baja al 0.93, por tanto investigue y decidí aplicarlo, En el análisis supervisado, se utilizó train_test_split con stratify=y para asegurar que la proporción de clases se mantuviera en ambos conjuntos (train y test). Esto garantiza una evaluación realista y reproducible. Al omitir stratify, la precisión del modelo puede llegar a 1.00 debido a una distribución de clases desequilibrada en el conjunto de prueba. Por lo tanto, se optó por incluir stratify=y para reflejar resultados más representativos.

- ***"Por tanto, subiré los dos archivos. Ipynb adjuntos para que vea ambos códigos y sus resultados".***

Luego pase a los puntos 3, 4, 5 en un solo bloque para cada uno de los siguientes modelos:

a. Árbol de decisión (Decision Tree Classifier)
b. Random Forest (100 árboles).
c. SVM con kernel RBF. (Support Vector Classifier)

Descrito a continuación:

a. Modelo Árbol de decisión (Decision Tree Classifier)

a.1. Códigos y resultados para modelo Árbol de decisión (Decision Tree Classifier):

a.1.1.Creación, entrenamiento del modelo:

```
# a. Arbol de decisión.
# punto 3 del ejercicio:
# Creamos el modelo de árbol de decisión.
clf = DecisionTreeClassifier(
    random_state=42
)  # Crea un clasificador de árbol de decisión vacío
clf.fit(
    x_train, y_train
)
```

✓ 0.0s

DecisionTreeClassifier

DecisionTreeClassifier(random_state=42)

a.1.2. Predicción y evaluación del modelo, entre otros reporte de clasificación:

```python
# punto 4 del ejercicio:
# predecimos las etiquetas del conjunto de prueba del modelo de Arbol de Decisión.
Y_pred = clf.predict(
    x_test
)
# Evaluar la precisión del modelo
accuracy = (Y_pred == y_test).mean()
# imprimir la precisión del árbol de decisión
print(f"Precisión del Árbol de Decisión: {accuracy:.2f}")
# Lo siguiente lo adicione para analisis mas especificos del modelo para posibles usus futuros
# Reporte de clasificación y matriz de confusión
print("Reporte de clasificación Árbol de Decisión:")
print(classification_report(y_test, Y_pred))
print("Matriz de confusión Árbol de Decisión:")
print(confusion_matrix(y_test, Y_pred))
```

✓ 0.0s

a.1.3. Resultado de la predicción y evaluación del modelo del arbol de desición entre otros reportes de clasificación adicionales:

```
Precisión del Árbol de Decisión: 0.93
Reporte de clasificación Árbol de Decisión:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        15
           1       1.00      0.80      0.89        15
           2       0.83      1.00      0.91        15

    accuracy                           0.93        45
   macro avg       0.94      0.93      0.93        45
weighted avg       0.94      0.93      0.93        45

Matriz de confusión Árbol de Decisión:
[[15  0  0]
 [ 0 12  3]
 [ 0  0 15]]
```

a.2. Conclusión sobre los resultados del Árbol de Decisión:

- **La precisión general del modelo (accuracy)** es del 93%. Esto significa que el modelo acertó en 93 de cada 100 predicciones en el conjunto de prueba.
- **En el reporte de clasificación**:

Para la clase 0: Precisión y recall perfectos (1.00), es decir, el modelo identifica correctamente todas las instancias de esta clase sin confundirlas.

Para la clase 1: Precisión perfecta (1.00) pero recall del 0.80. Esto indica que aunque el modelo no cometió errores al identificar esta clase, dejó pasar (no detectó) un 20% de instancias verdaderas de esta clase.

Para la clase 2: Precisión más baja (0.83) pero recall perfecto (1.00). El modelo detecta todas las instancias de esta clase, pero también clasifica erróneamente otras instancias como clase 2.

- **La matriz de confusión muestra**:
  - Clase 0: 15 aciertos, 0 errores.
  - Clase 1: 12 aciertos, 3 errores (estas 3 instancias fueron clasificadas incorrectamente como clase 2).
  - Clase 2: 15 aciertos, 0 errores.
- **Conclusión final:**
  El modelo de Árbol de Decisión funciona bien, especialmente para las clases 0 y 2. Sin embargo, tiene dificultades para diferenciar la clase 1 de la clase 2, como lo muestra el recall reducido (0.80) y los errores en la matriz de confusión. Es posible que las características usadas para el entrenamiento no sean suficientes para separarlas completamente. Esto es común en conjuntos de datos donde las clases tienen similitudes.

a.3. Desarrollo del punto 5 del ejercicio Visualización del árbol de decisión:
Se genera un diagrama que muestra las reglas de decisión aprendidas por el modelo, incluí este punto aquí para que quedaran junto con los códigos del árbol de decisión:

a.3.Visualización del árbol de decisión:

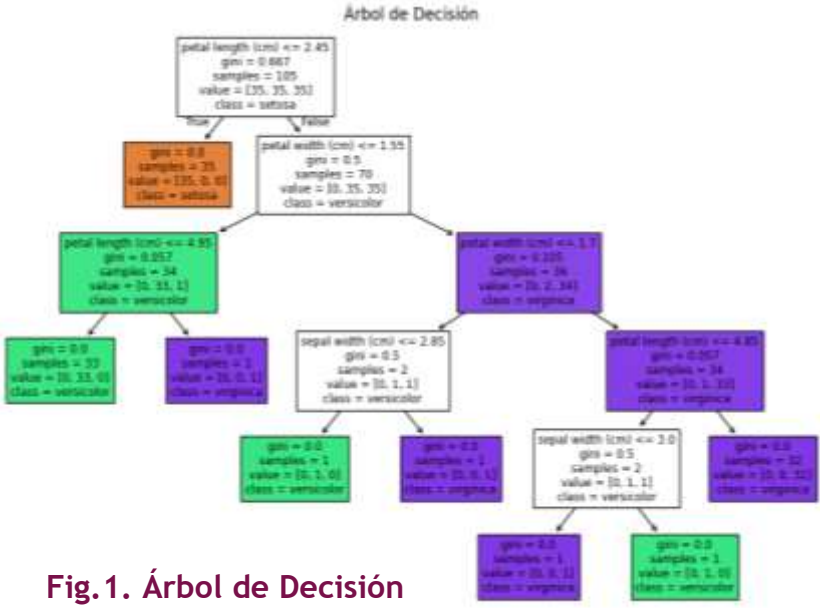


Fig.1. Árbol de Decisión

a.3.1 Conclusiones de la visualización del arbol de desición:

- El modelo ha clasificado correctamente las tres especies del conjunto Iris utilizando principalmente dos características: longitud y anchura del pétalo.
- La primera división, petal length ≤ 2.45, separa claramente la especie setosa del resto, demostrando su fácil distinción.
- Los nodos terminales con gini = 0 muestran que la clasificación en esos puntos es perfecta (100% de una sola clase).
- **Precisión obtenida:**

-Accuracy = 0.93 (cuando se usa stratify=y para balancear las clases).
-Accuracy = 1.00 (sin stratify, pero con posible sobreajuste).

b.1.2. Predicción y evaluación del modelo Random forest:

```
# Punto 4 del ejercicio:
# Predecimos las etiquetas del conjunto de prueba del modelo de Random Forest.
y_pred_rf = rf.predict(x_test)  Undefined name `rf`  Undefined name `x_test`

# Evaluar la precisión
accuracy_rf = (y_pred_rf == y_test).mean()  Undefined name `y_test`
print(f"Precisión del Random Forest: {accuracy_rf:.2f}")
```

✓ 0.0s

```
Precisión del Random Forest: 0.89
```

- **Conclusión:** El modelo es efectivo, pero es importante evaluar generalización con datos nuevos. La representación gráfica facilita la comprensión del modelo.

b. Modelo Random Forest (100 árboles).

b.1. Códigos y resultados para modelo Random Forest (100 árboles):

b.1.1. Creación, entrenamiento del modelo**:**

```
# b. Random Forest (100 árboles).
# Punto 3 del ejercicio:
# Creamos el modelo de Random Forest con 100 árboles.
rf = RandomForestClassifier(n_estimators=100, random_state=42)
# Entrenamos el modelo.
rf.fit(x_train, y_train)
```

✓ 0.1s

```
RandomForestClassifier
RandomForestClassifier(random_state=42)
```

b.2. Conclusión sobre los resultados del modelo Random Forest:

- La precisión general del modelo (accuracy) es del 89%. Esto significa que el modelo acertó en 89 de cada 100 predicciones en el conjunto de prueba.
- Comparado con el Árbol de Decisión (que tenía un accuracy de 93%), el modelo Random Forest muestra un rendimiento ligeramente inferior. Esto puede deberse a varios factores:
  - -Diferencias en el muestreo aleatorio al crear los múltiples árboles (recordemos que Random Forest combina muchos árboles).
  - -Posiblemente, el parámetro random_state o el tamaño del conjunto de datos y su partición influyeron.
  - -Otra posible razón es que el modelo Random Forest tiende a suavizar predicciones, lo cual puede reducir

ligeramente el rendimiento en conjuntos de datos pequeños como el Iris.

- **Conclusión final**:
  El modelo Random Forest, aunque potente y generalmente más robusto que un solo árbol, no siempre garantiza una precisión más alta, especialmente en conjuntos de datos pequeños y balanceados como Iris. Aquí obtuvo una precisión **ligeramente menor (0.89)** en comparación con el Árbol de Decisión. Esto resalta la importancia de considerar el tamaño y características del conjunto de datos, así como la necesidad de ajustar hiperparámetros (como el número de árboles, profundidad, etc.) para mejorar el rendimiento.

c. Modelo SVM con kernel RBF. (Support Vector Classifier)

c.1. Códigos y resultados para modelo SVM con kernel RBF. (Support Vector Classifier):

c.1.1.Creación, entrenamiento del modelo:

```
# c. SVM con kernel RBF.
# Punto 3 del ejercicio:
# Creamos el modelo SVM con kernel RBF.
svm_model = SVC(kernel='rbf', random_state=42)
# Entrenamos el modelo SVM.
svm_model.fit(x_train, y_train)
```

c.1.2. Predicción y evaluación del modelo

```
# Punto 4 del ejercicio:
# Predecimos las etiquetas del conjunto de prueba del modelo SVM.
y_pred_svm = svm_model.predict(x_test)
# Evaluar la precisión del modelo SVM
accuracy_svm = (y_pred_svm == y_test).mean()
print(f"Precisión del SVM: {accuracy_svm:.2f}")
```

c.1.3. Resultado de la precisión del modelo:

```
Precisión del SVM: 0.96
```

C.2. Conclusiones del modelo:

- El modelo SVM con kernel RBF se aplicó para clasificar las tres especies del conjunto de datos Iris.
- Se utilizó un split 70%-30% para entrenamiento y prueba, logrando una precisión del 96%.
- Este nivel de precisión demuestra que el modelo SVM es capaz de separar correctamente la mayoría de las especies, gracias a su capacidad para encontrar fronteras no lineales complejas en los datos.
- El kernel RBF (Radial Basis Function) es especialmente efectivo para casos en los que los datos no son separables linealmente, como ocurre con algunas especies del Iris.
- A pesar de que no alcanza el 100% de precisión (como puede ocurrir con modelos sobreajustados), este resultado refleja un buen equilibrio entre ajuste y generalización.
- Conclusión clave: El modelo SVM con kernel RBF es altamente efectivo para este tipo de problema y supera a modelos como KMeans y DBSCAN, que no usan etiquetas. Es un modelo supervisado que logra una separación precisa y consistente entre las especies.

Parte 2: Clustering no supervisado

1. Usa solo las dos primeras características para facilitar visualización.

1.a. Codigo para visualizar las dos primeras caracteristicas:

```
# 1. Usa solo las dos primeras características para facilitar visualización.
# Seleccionamos las dos primeras características
X_vis = X[:, :2] # Usar 'sepal length' y 'sepal width'
```

Este codigo a diferencia del usado en la clase no afecta el conjunto de entrenamiento/prueba, solo limita las características a dos columnas, mientras que el de la clase es otro código selecciona qué clases (Setosa, versicolor o virgnica) usar encambio este es diferente esta direccionado a las 4 caracteristicas de las flores iris.

2. Aplica K-Means para encontrar 3 clusters. Visualiza los clusters y sus centroides.

2.1. Aplicar K-Means para encontrar 3 clusters:

```
# 2. Aplica K-Means para encontrar 3 clusters. Visualiza
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans_labels = kmeans.fit_predict(X_vis)
centroids = kmeans.cluster_centers_
```

2.2.Evaluar el modelo K-means (punto 4):

Calculo cuantitativo de la aproximidad de los clusters a los datos reales a traves del siguiente codigo:

```
# Evaluar K-Means con ARI y homogeneity_score
# Para evaluar qué tan bien se aproximan los clusters a las clases reales
ari = adjusted_rand_score(y, kmeans_labels)
homogeneity = homogeneity_score(y, kmeans_labels)
print(f"Adjusted Rand Index (K-Means): {ari:.2f}")
print(f"Homogeneity Score (K-Means): {homogeneity:.2f}")
```

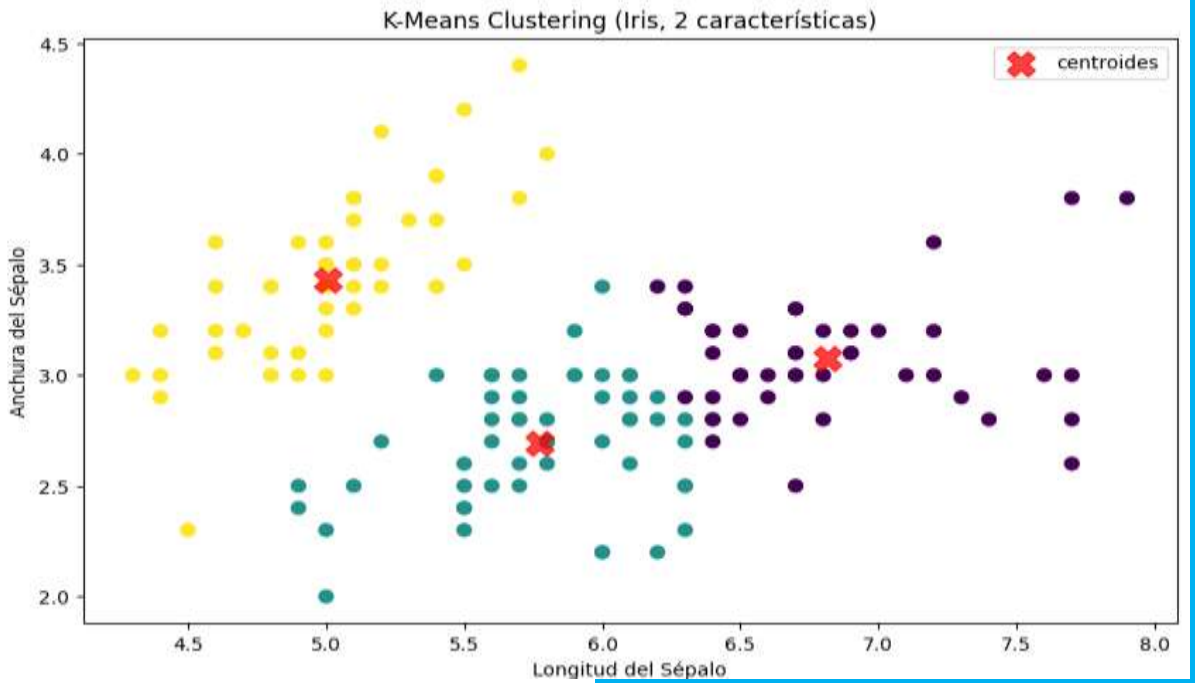
2.2.1.Resultado de la evaluación del modelo K-means:

```
Adjusted Rand Index (K-Means): 0.60
Homogeneity Score (K-Means): 0.65
```

2.3. Visualización de los clusters y sus centroides:

```
# Visualizar los clusters de K-Means
plt.figure(figsize=(10, 6))
plt.scatter(X_vis[:, 0], X_vis[:, 1], c=kmeans_labels, s=50, cmap="viridis")
plt.scatter(
    centroids[:, 0],
    centroids[:, 1],
    c="red",
    s=200,
    alpha=0.75,
    marker="X",
    label="centroides",
)
plt.title("K-Means Clustering (Iris, 2 características)")
plt.xlabel("Longitud del Sépalo")
plt.ylabel("Anchura del Sépalo")
plt.legend()
plt.show()
```

2.3.1. Resultado de la visualización del modelo K-mean para 3 clusters y dos caracteristicas:



2.4.Conclusiones del modelo K-means para encontrar 3 clusters:

El modelo K-Means aplicado a solo dos características del dataset Iris logra una agrupación de calidad media (moderada). La coincidencia con las etiquetas reales es del 60%, y la homogeneidad de los clusters es del 65%. Esto resalta que aunque K-Means puede identificar patrones en los datos, utilizar únicamente dos características puede limitar su capacidad de separación. Incorporar más características o ajustar parámetros (como número de clusters o inicialización) podría mejorar los resultados.

2.4.1.Conclusiones de la Visualización del modelo k-means:

- K-Means logró agrupar los datos en tres clusters con una precisión visual aceptable. Sin embargo, la superposición entre algunos puntos sugiere que el modelo no logra separar perfectamente todas las clases reales. Es un resultado consistente con las métricas calculadas y adecuado para un análisis exploratorio.
- Los centroides se encuentran aproximadamente en el centro de cada agrupación de puntos. Esto indica que el modelo fue capaz de determinar regiones con alta densidad de datos y establecerlos como referencia de cada grupo.
- El análisis se basó solo en dos características para facilitar la visualización, lo que limita la capacidad del modelo para distinguir claramente todas las clases.

3. Aplica DBSCAN con eps=0.5 y min_samples=5. Visualiza los clusters y los puntos etiquetados como ruido.

3.1.Codigo para aplicar DBSCAN con eps=0.5 y min_samples=5:

```
# 3. Aplica DBSCAN con eps=0.5 y min_samples=5.
# Visualiza los clusters y los puntos etiquetados como ruido.
# Aplicar DBSCAN (eps=0.5, min_samples=5)
dbscan = DBSCAN(eps=0.5, min_samples=5)
dbscan_labels = dbscan.fit_predict(X_vis)
```

3.2.Evaluar el modelo DBSCAN con eps=0.5 y min_samples=5 (punto 4) con datos reales:

Con el siguiente codigo se evalua **cuantitativamente** la comparación de los clusters obtenidos (en `dbscan_labels`) con las etiquetas reales (`y`).

```
# Evaluar DBSCAN comparando con etiquetas reales este lo adicione para verlo:
ari_dbscan = adjusted_rand_score(y, dbscan_labels)
homogeneity_dbscan = homogeneity_score(y, dbscan_labels)
print(f"Adjusted Rand Index (DBSCAN vs Real): {ari_dbscan:.2f}")
print(f"Homogeneity Score (DBSCAN vs Real): {homogeneity_dbscan:.2f}")
```

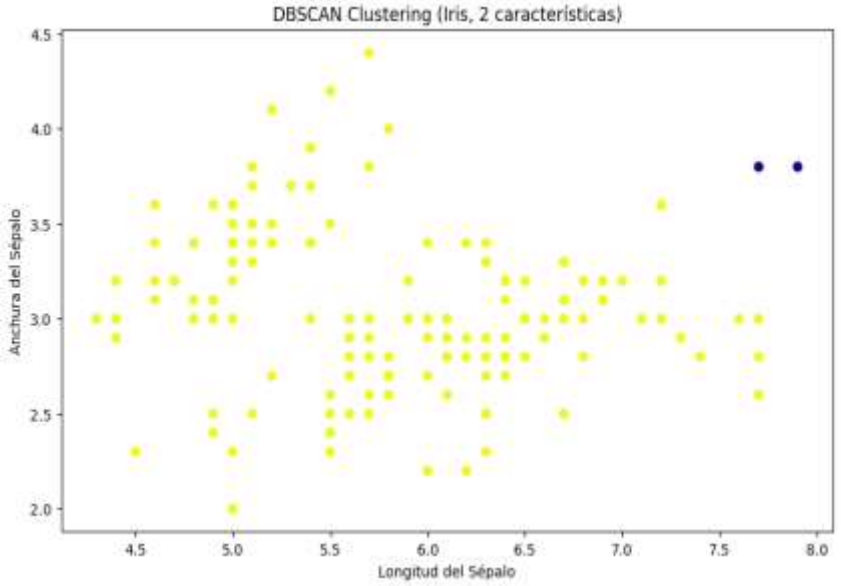
3.2.1.Resultado de la evaluación del modelo

```
Adjusted Rand Index (DBSCAN vs Real): 0.00
Homogeneity Score (DBSCAN vs Real): 0.01
```

3.3.Visualización el modelo DBSCAN con eps=0.5 y min_samples=5 y dos caracteristicas:

```
# Visualizar los clusters de DBSCAN
plt.figure(figsize=(10, 6))
plt.scatter(
    X_vis[:, 0], X_vis[:, 1], c=dbscan_labels, cmap="plasma", label="Clusters DBSCAN"
)
plt.title("DBSCAN Clustering (Iris, 2 caracteristicas)")
plt.xlabel("Longitud del Sépalo")
plt.ylabel("Anchura del Sépalo")
plt.show()
```

3.3.1. Resultado de la visualización del modelo DBSCAN con eps=0.5 y min_samples=5 y dos caracteristicas:



3.4.Conclusiones del modelo DBSCAN con eps=0.5 y min_samples=5 y dos caracteristicas:

- DBSCAN con eps=0.5 y min_samples=5 se aplicó al conjunto de datos Iris usando solo las dos primeras características (sepal length y sepal width).
- Un **ARI de 0.00** indica que **no hay coincidencia** entre los clusters generados por DBSCAN y las etiquetas reales. Es decir, el algoritmo DBSCAN no logró identificar correctamente las clases de flores.
- Un **Homogeneity Score de 0.01** también refleja que **los clusters generados no son homogéneos**, es decir, cada cluster contiene una gran mezcla de etiquetas reales.

3.4.1.Conclusiones de la visualizacion del modelo DBSCAN con eps=0.5 y min_samples=5 y dos caracteristicas:

- La visualización muestra que DBSCAN etiquetó incorrectamente gran parte de los datos, agrupando casi todos los puntos en un solo cluster y marcando algunos como ruido.

- **Posibles causas de porque el modelo DBSCAN agrupa todo los puntos en un solo clusters y marcando unos como ruido:**

-DBSCAN es sensible a la escala y densidad de los datos. Al usar solo dos características y sin escalar los datos, el algoritmo puede no encontrar densidades adecuadas para separar los clusters.
-Los valores de eps=0.5 y min_samples=5 pueden no ser los más óptimos para el dataset Iris. Se podrían explorar otros valores para obtener mejores resultados.
-DBSCAN es más efectivo para datos con densidades bien diferenciadas, mientras que los datos del Iris tienen clusters más solapados.

"Este análisis resalta la importancia de elegir el algoritmo y los parámetros adecuados **para el problema y los datos específicos"**

4. Compara los clusters encontrados con las etiquetas reales:

Ya este punto se ha venido desarrollando simunltanemente para cada modelos **K-mean** y **DBSCAN** en los item **2.2 y 3.2** respectivamente de forma **cuantitativa** (**evaluacion ARI** y **Homogeneity Score**) dandonos las conclusiones ya detalladas anteriormente para cada modelo.

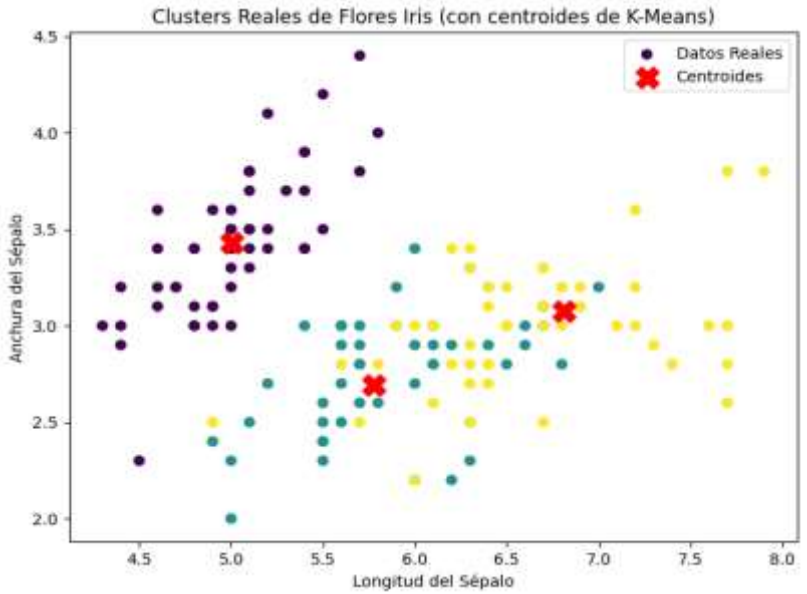
A continuación se evidenciare la comparación de forma visual de la comparación de los clusteres con las etiquetas reales:

4.1.Codigo de Visualización de los clusters reales junto con los centroides de K-Means

```
# 4. Visualizar los clusters con las etiquetas reales
plt.figure(figsize=(8, 6))
plt.scatter(X_vis[:, 0], X_vis[:, 1], c=y, cmap="viridis", label="Datos Reales")
plt.scatter(
    centroids[:, 0], centroids[:, 1], c="red", marker="X", s=200, label="Centroides"
)
plt.title("Clusters Reales de Flores Iris (con centroides de K-Means)")
plt.xlabel("Longitud del Sépalo")
plt.ylabel("Anchura del Sépalo")
plt.legend()
plt.show()
```

- ***Explicación de la visual comparativa***:
- Los clusters reales (etiquetas verdaderas del dataset) son diferentes de los clusters encontrados por K-Means.
- Centroides de K-Means son solo puntos representativos del algoritmo de clustering, que se pueden superponer en el gráfico de las etiquetas reales.
- Entonces, lo que haces es graficar la realidad (datos reales) junto a los centroides calculados por K-Means.

4.1.1.Resultado de la comparacion de los clusteres de K-means VS etiquetas reales:



4.1.1.1.Conclusiones de clusters reales junto con los centroides de K-Means:

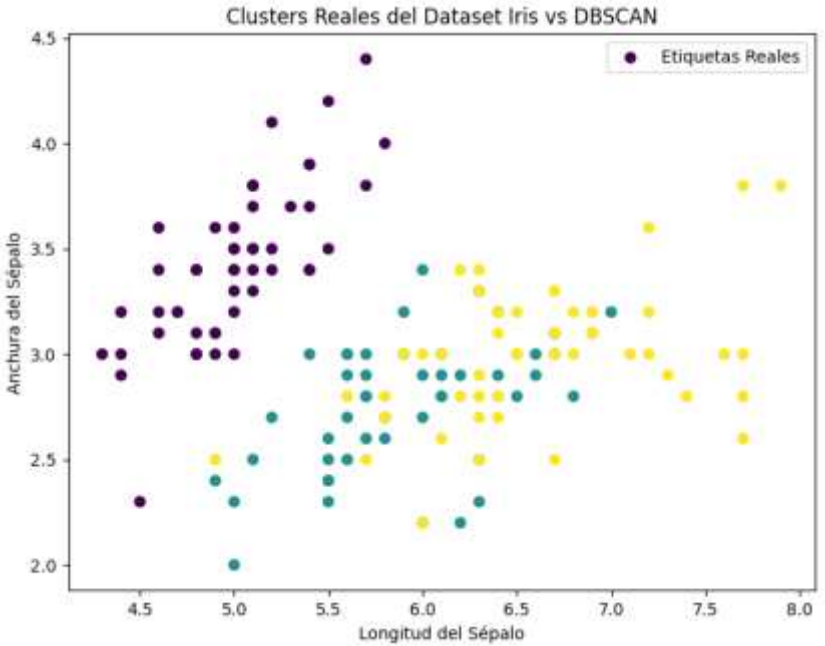
El gráfico representa los datos reales del conjunto Iris distribuidos en dos dimensiones: longitud y anchura del sépalo, donde los puntos están coloreados según su especie real (Setosa, Versicolor y Virginica).
Se observa que los centroides están razonablemente bien ubicados respecto a los grupos reales de datos, lo que indica que K-Means pudo identificar aproximadamente el centro de cada cluster. Sin embargo, K-Means no coincide perfectamente con las clases reales: algunos puntos están lejos del centroide esperado para su especie, lo que refleja limitaciones del algoritmo. La dispersión y solapamiento entre Versicolor y Virginica (colores verde y amarillo) explica por qué el modelo puede confundirlas.
En general: **K-Means** logra una segmentación aproximada de los datos reales de Iris, pero debido a la variabilidad natural entre especies, algunos datos se agrupan incorrectamente.

4.2. Codigo de para la Visualización de la comparación de los clusters encontrados por DBSCAN vs etiquetas reales:

```
# 4. Visualizar los clusters de DBSCAN con etiquetas reales.
# Visualizamos la comparación: clusters encontrados por DBSCAN vs etiquetas reales
plt.figure(figsize=(8, 6))
plt.scatter(X_vis[:, 0], X_vis[:, 1], c=y, cmap='viridis', label='Etiquetas Reales')
plt.title('Clusters Reales del Dataset Iris vs DBSCAN')
plt.xlabel('Longitud del Sépalo')
plt.ylabel('Anchura del Sépalo')
plt.legend()
plt.show()
```

Este código genera un gráfico que muestra los **datos reales del conjunto Iris**, usando las dos primeras características (longitud y anchura del sépalo). Los puntos están coloreados según la especie real de cada flor.

4.2.1.Resulatdo de la comparación visual de clusteres encontrados por DBSCAN Vs etiquetas reales:



4.2.1.1.Conclusiones de la comparación visual de clusteres encontrados por DBSCAN Vs etiquetas reales:

- El grafico muestran los clusters reales del dataset Iris (con las dos primeras características: longitud y anchura del sépalo), usando la codificación de color según las etiquetas reales.
- Aunque el gráfico de las etiquetas reales muestra bien definidos los tres grupos de especies, **DBSCAN no logra replicar esta separación usando solo dos características**. Esto se debe a que DBSCAN es sensible a la densidad y puede verse afectado por la distribución de los datos y los parámetros usados.

Preguntas para discusión:

1. ¿Cuál modelo supervisado tuvo mejor precisión? ¿Por qué?:

R/. El modelo con mejor precisión fue el **SVM con kernel RBF**, con una precisión aproximada del **96%**. Esto se debe a que SVM es muy eficaz para encontrar fronteras de decisión complejas y separar clases que no son linealmente separables, especialmente con el kernel RBF que introduce no linealidad.

2. ¿Cómo se relacionan los clusters de K-Means con las clases reales?:

R/. K-Means encuentra 3 clusters que, aunque aproximan a las clases reales, no son perfectamente coincidentes. Esto se observa en los gráficos donde los centroides de K-Means están bien posicionados, pero algunos puntos quedan mal clasificados. Además, los indicadores (ARI=0.60, Homogeneity=0.65) muestran una correspondencia moderada.
K-Means intenta optimizar la cohesión interna, pero no tiene en cuenta las etiquetas reales.

3. ¿Qué utilidad tiene DBSCAN en comparación con K-Means?:

R/. DBSCAN es útil para encontrar clusters de forma arbitraria y manejar bien los **outliers** (puntos de ruido), mientras que K-Means tiende a forzar todos los puntos en clusters. En este caso, DBSCAN tuvo **bajo rendimiento (ARI=0.00, Homogeneity=0.01)**, indicando que no agrupó bien según las clases reales. Sin embargo, en datasets con densidades variables o clusters no esféricos, **DBSCAN** podría ser más efectivo.

4. ¿Qué ventajas y desventajas tiene un árbol de decisión frente a un Random Forest?:

R/.

- ***Árbol de decisión:***
- Ventajas: Fácil de interpretar, visualización clara, rápido entrenamiento.
- Desventajas: Alta varianza, propenso al overfitting.

- ***Random Forest:***
- Ventajas: Combina varios árboles para reducir varianza y mejorar generalización, alta precisión.
- Desventajas: Menos interpretabilidad, más costoso computacionalmente.

5. ¿Cómo ayuda la visualización del árbol para interpretar el modelo?

La visualización del árbol muestra claramente **las decisiones basadas en características** (ej: longitud del pétalo, ancho del sépalo) y las **ramificaciones que llevan a cada clase**.
Esto permite **identificar las reglas de decisión** y comprender cómo se clasifican los datos. En contraste, modelos más complejos como Random Forest o SVM no ofrecen visualizaciones tan interpretables.

Explicando esto de manera mas profunda puedo decir lo siguiente :

✓ La primera división se basa en la longitud del pétalo (petal length (cm) <= 2.45):
- Si es verdadera: todas son Setosa (pureza total, gini=0).
- Si es falsa: pasamos al siguiente nodo, donde se decide entre Versicolor y Virginica.

Se siguen aplicando reglas (e.g., ancho del pétalo, ancho del sépalo) hasta que se llega a un nodo hoja, donde no hay más división posible.

✓ Interpretación de cada elemento del nodo:
- Regla de decisión: e.g., petal length (cm) <= 2.45.
- Gini: mide impureza; más cerca de 0, más "puro" es el nodo.
- Samples: número de muestras (flores) que cumplen la regla.
- Value: distribución de clases en ese nodo.
- Class: la clase mayoritaria del nodo.

En **resumen** como lo hemos visto anteriormente, la visualización del **árbol de decisión** nos ayuda mucho a entender como se clasifica el dataset Iris mediante reglas basadas en características del pétalo y sépalo de **manera mas interactiva y eficiente**. Su estructura es fácil de interpretar y nos permite ver visualmente cómo se toman las decisiones de clasificación. Los nodos muestran pureza (gini), distribución de clases y número de muestras. Esta visualización facilita entender el comportamiento del modelo.

✓ **Bonus (opcional):**

1. Experimenta con diferentes valores de eps y min_samples en DBSCAN:

1.1.Codigo para modelo DBSCAN y probar eps y min_samples diferentes a eps=0.5 y min_samples=5:

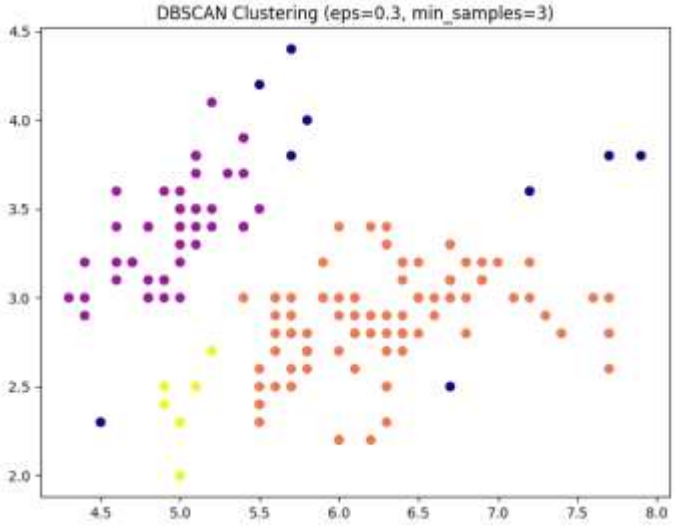
1.2.Codigo para Visualización de las pruebas con eps y min_samples diferentes a eps=0.5 y min_samples=5:

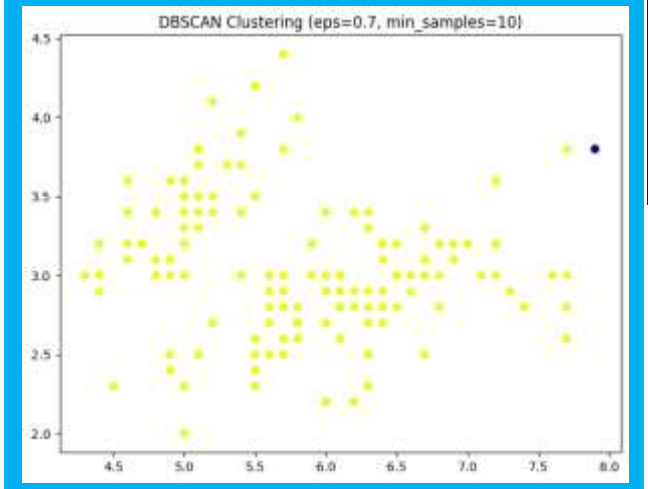
```
# Puedes graficar para cada caso:
plt.figure(figsize=(8, 6))
plt.scatter(X_vis[:, 0], X_vis[:, 1], c=labels_alt1, cmap='plasma')
plt.title('DBSCAN Clustering (eps=0.3, min_samples=3)')
plt.show()

plt.figure(figsize=(8, 6))
plt.scatter(X_vis[:, 0], X_vis[:, 1], c=labels_alt2, cmap='plasma')
plt.title('DBSCAN Clustering (eps=0.7, min_samples=10)')
plt.show()
```

1.3. Codigo para Evaluar cuantitativamene los cluster obtenidos de las pruebas con eps y min_samples diferentes a eps=0.5 y min_samples=5 Vs. las etiquetas reales:

```
# Evaluar con etiquetas reales
print("Evaluación para eps=0.3, min_samples=3")
print("ARI:", adjusted_rand_score(y, labels_alt1))
print("Homogeneity:", homogeneity_score(y, labels_alt1))

print("Evaluación para eps=0.7, min_samples=10")
print("ARI:", adjusted_rand_score(y, labels_alt2))
print("Homogeneity:", homogeneity_score(y, labels_alt2))
```

1.2.1.Resultados de las Visualizaciónes de modelo DBSCAN con eps y min_samples diferentes a eps=0.5 y min_samples=5 :





1.3.1.Resultado de la evaluación de la presición del modelo DBSCAN de los cluster obtenidos de las pruebas con eps y min_samples diferentes a eps=0.5 y min_samples=5 Vs las etiquetas reales:

```
Evaluación para eps=0.3, min_samples=3
ARI: 0.49483116976697095
Homogeneity: 0.5675194459526063
Evaluación para eps=0.7, min_samples=10
ARI: 0.0
Homogeneity: 0.006707485288362729
```

Esta es un comparación cuantitativa de los clusteres obtenidos por modelo DBSCAN y las etiquetas ***Reales***, tanto para un eps=0.3 y 0.7, con mi_samples=3 y 10, con esto podemos observar la presición y que tanto se aproxima la predicción del modelo a los datos reales.

1.4. Conclusiones de las pruebas modelo DBSCAN con eps y min_samples diferentes a eps=0.5 y min_samples=5 :

✓ *Configuración eps=0.3 y min_samples=3*

- **Visualización**: Los datos se agrupan en varios clusters con algunos puntos dispersos (ruido). Se observa una estructura que, aunque con algo de dispersión, logra identificar subgrupos en el dataset.
- **Métricas**:
  - *Adjusted Rand Index (ARI)*: **0.49** (valor intermedio), lo que indica que el clustering tiene cierta similitud con las etiquetas reales, pero no es perfecto.
  - *Homogeneity Score*: **0.57**, reflejando que los clusters son razonablemente puros en términos de pertenencia a una sola clase.

✓ *Configuración eps=0.7 y min_samples=10*

- **Visualización**: La mayoría de los datos se agrupan en un solo cluster (color predominante), y hay varios puntos marcados como ruido. La estructura del clustering es pobre y no logra diferenciar bien los subgrupos.
- **Métricas**:
  - *ARI*: **0.0**, indicando que no hay concordancia entre los clusters formados y las clases reales.
  - *Homogeneity*: **0.0067**, casi nula, lo que confirma la baja calidad del clustering en esta configuración.

✓ **Interpretación General**

- **DBSCAN es sensible a los parámetros eps y min_samples**. Un valor pequeño de eps (como 0.3) permite identificar subgrupos más compactos, pero también puede fragmentar clusters reales. Por otro lado, un eps demasiado grande (como 0.7) tiende a agrupar la mayoría de los puntos en un solo cluster y marca muchos puntos como ruido, lo que reduce la calidad del clustering.
- **DBSCAN tiene dificultades para datasets con distribuciones complejas o con solapamiento entre clases** como es el caso del dataset Iris, donde los tres grupos no están claramente separados cuando se usan solo las dos primeras características (longitud y anchura del sépalo).
- **Las métricas ARI y Homogeneity son útiles para evaluar el clustering**, ya que reflejan numéricamente la similitud entre los clusters encontrados y las clases reales.
- **DBSCAN es útil cuando se quiere identificar clusters de formas arbitrarias y con presencia de ruido**, pero no es tan efectivo en datasets donde los clusters tienen formas convexas y bien separadas, como espera K-Means.

2. Cambia el kernel de SVM a linear y compara resultados:

Las diferencias entre los resultados obtenidos al aplicar SVM con kernel lineal y SVM con kernel RBF, y por qué sus precisiones pueden ser distintas.

2.1.Codigo para crear y entrenar el modelo con Kernel de SVM Lineal:

```
# Entrenar SVM con kernel linear
svm_linear = SVC(kernel='linear', random_state=42)
svm_linear.fit(x_train, y_train)
```

✓ 0.0s

```
SVC
SVC(kernel='linear', random_state=42)
```

2.2. Codigo para predicción y evaluación del modelo con Kernel de SVM Lineal:

```
#predecir con SVM lineal
y_pred_linear = svm_linear.predict(x_test)
# Evaluar la precisión del modelo SVM lineal
accuracy_linear = accuracy_score(y_test, y_pred_linear)
print(f"Precisión del SVM (linear): {accuracy_linear:.2f}")
```

✓ 0.0s

2.3.Resultado de la precisión del modelo SVM con Kernerl lineal:

```
Precisión del SVM (linear): 1.00
```

2.4.Diferencias observadas:

-SVM con kernel lineal tiene una precisión de 1.00 (100%).
-SVM con kernel RBF tiene una precisión de 0.96 (96%).

2.5.Explicación:

2.5.1. Kernel lineal:

- Se utiliza un modelo de SVM que intenta trazar una línea recta (o plano en dimensiones superiores) para separar las clases.
- El dataset Iris tiene datos bien separados linealmente (especialmente entre la clase setosa y las otras).
- Por eso, en algunos casos, el modelo lineal puede clasificar perfectamente, logrando 100% de precisión.
- Además, si los datos de entrenamiento están bien divididos (sin demasiadas clases mezcladas), la separación es clara y lineal.

2.5.2. Kernel RBF (Radial Basis Function):

- Utiliza una transformación no lineal del espacio de características, creando fronteras curvas.
- Sin embargo, para un dataset como Iris, que es mayormente linealmente separable, este exceso de flexibilidad puede llevar a sobreajuste o errores leves, y por eso la precisión es del 96%.

2.6. Conclusiones generales de las diferencia de Kernel:

- ✓ **Kernel lineal** puede ser más efectivo para el dataset Iris porque las clases son linealmente separables (en especial con las dos primeras características).
- ✓ **Kernel RBF** es más flexible y útil cuando los datos no son linealmente separables, pero en este caso puede ser innecesario y generar más errores.
- ✓ **La diferencia en precisión (1.00 vs 0.96)** muestra que la complejidad adicional del kernel RBF **no mejora el rendimiento** para este caso, y puede incluso afectar ligeramente al modelo.

2.7.Tabla comparativa (resumen comparativo del Kernel Lineal y Kernel RBF).

Característica	SVM con kernel lineal	SVM con kernel RBF
Kernel usado	Lineal	Radial Basis Function
Precisión	**1.00 (100%)**	**0.96 (96%)**
Separabilidad de clases	Muy buena (lineal)	También buena, pero más compleja
Complejidad del modelo	Baja	Alta
Riesgo de sobreajuste	Bajo	Moderado
Adecuado para datos lineales	Sí	A veces innecesario
Tiempo de entrenamiento	Menor	Mayor

3. Intenta usar todas las características para clustering y observa qué cambia:

Usare todas las características del dataset Iris. para aplica **K-Means** y **DBSCAN**, a su vez **c**alculare el **ARI** y **Homogeneity Score** para comparar con las etiquetas reales y poder realizar la comparación con el que realice de dos caracteristicas veamos el bloque de codigo para los modelos:

a. K-means.
b. DBSCAN.

a.3.1.Codigo crear, entrenar, predecir y evaluar el modelo K_mean con todas las caracteristicas para los clustering:

```
# Usar todas las características
X_all_features = X  # Dataset completo (4 características)

# Creamos modelo K-Means con todas las características y entrenamos.
kmeans_all = KMeans(n_clusters=3, random_state=42)
kmeans_all_labels = kmeans_all.fit_predict(X_all_features)
centroids_all = kmeans_all.cluster_centers_

# Evaluar K-Means
ari_kmeans_all = adjusted_rand_score(y, kmeans_all_labels)
homogeneity_kmeans_all = homogeneity_score(y, kmeans_all_labels)

print(f"K-Means usando todas las características:")
print(f"Adjusted Rand Index: {ari_kmeans_all:.2f}")
print(f"Homogeneity Score: {homogeneity_kmeans_all:.2f}")
```

✓ 0.0s

a.3.1.2.Resultado del codigo anterior el modelo K_mean con todas las caracteristicas para los clustering:

```
K-Means usando todas las características:
Adjusted Rand Index: 0.72
Homogeneity Score: 0.74
```

a.3.2.Codigo para reducir a 2D con PCA para visualización de K-mean con todas las caracteristicas:

```
# Reducir a 2D con PCA para visualización
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_all_features)
centroids_pca = pca.transform(centroids_all)
```

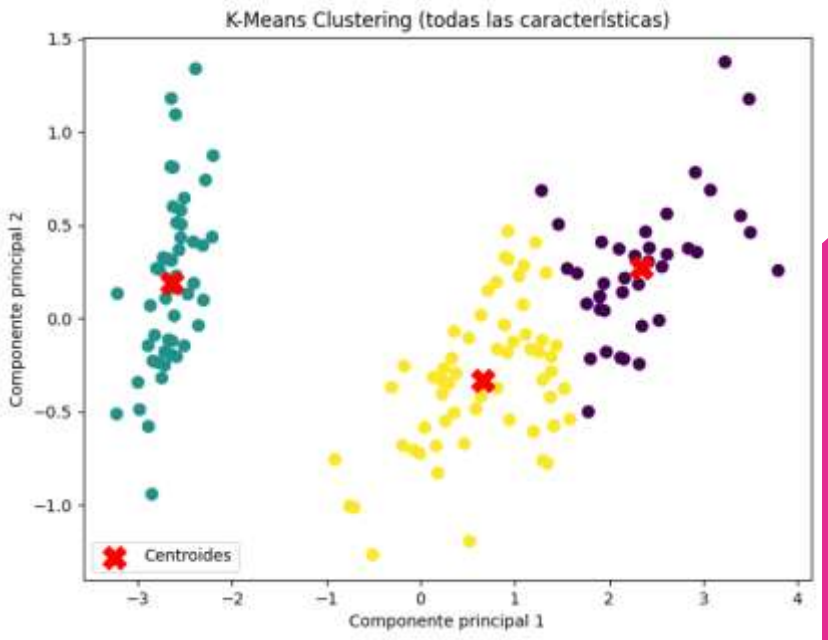
- Ante la necesidad de visualizar el K-mean teniendo las 4 caracteristicas inlcuidas en la clasificacion de los datos y como solo podemos visualizar espacios de 2D (planos) o 3D (como un cubo), ya que Un espacio de 4D es matemáticamente posible, pero no podemos verlo directamente, por eso fue necesario reducir las dimensiones a 2D o 3D para crear gráficos comprensibles, para ello tenemos a **PCA (Análisis de Componentes Principales)** toma todos los datos en 4D y los proyecta en un plano 2D, el cual Elige las **dos combinaciones de características (componentes)** que capturan la mayor varianza (información) posible.

- Así, se preserva la **mayor parte de la estructura de los datos** en un gráfico fácil de entender.
- Cabe resaltar un que, en ciencia de datos, **las dimensiones** son las **características o variables** que describen cada dato. En el caso del dataset Iris: Cada flor tiene **4 características** (dimensiones):
    1. Longitud del sépalo
    2. Anchura del sépalo.
    3. Longitud del pétalo
    4. Anchura del pétalo
- Cada flor es como un punto en un espacio **de 4 dimensiones**. Cada eje (dimensión) representa una de las características.
- **En Resumen: en el clustering con todas las características, tenemos un espacio de 4 dimensiones, el PCA nos ayuda a convertir esos datos en 2 dimensiones para poder** visualizar cómo se distribuyen los clusters **en un gráfico y aunque** PCA reduce la cantidad de información, **nos da una idea clara de los agrupamientos.**

a.3.3.Codigo de visualización del modelo K-mean con todas las caracteristicas (4), reducido a 2D con PCA:

```
# Visualizar los clusters de K-Means con todas las características
plt.figure(figsize=(10, 6))
plt.figure(figsize=(8,6))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=kmeans_all_labels, cmap='viridis', s=50)
plt.scatter(centroids_pca[:, 0], centroids_pca[:, 1], c='red', marker='X', s=200, label='Centroides')
plt.title("K-Means Clustering (todas las características)")
plt.xlabel("Componente principal 1")
plt.ylabel("Componente principal 2")
plt.legend()
plt.show()
```

a.3.3.1.Visualización del modelo K-mean con todas las caracteristicas (4), reducido a 2D con PCA:



a.3.4. Conclusiones del modelo K-mean con todas las caracteristicas (cuatro):

- Al Usar todas las características (4) para clustering con K-Means, evaluación de la precisión frente a las etiquetas son las siguientes:

- **Adjusted Rand Index (ARI):** 0.72.
- **Homogeneity Score:** 0.74.
-

Como vemos la utilización de todas las características del dataset Iris (longitud y ancho de sépalo, longitud y ancho de pétalo) mejora considerablemente la calidad del clustering, adicionalmente se observa una mayor capacidad del modelo K-Means para identificar correctamente los grupos o clusters reales, en comparación con cuando solo se usan dos características.

-**ARI y Homogeneity** muestran un aumento notable respecto al clustering con solo dos características (aproximadamente 0.6), lo que indica que incluir más información relevante (características adicionales) ayuda a que los clusters encontrados se asemejen más a las verdaderas etiquetas.

a.3.4.1.Conclusiones de la Visualizacion del modelo K-mean con todas las caracteristicas reducido a 2D con PCA:

- La visualización con **PCA** permite representar las 4 dimensiones del dataset original en solo 2 componentes principales.
- Esto facilita la interpretación visual del clustering al proyectar las 4 dimensiones en un plano bidimensional.
- Aunque PCA reduce información, mantiene la varianza más importante, lo que permite observar patrones y separación entre clusters.
- Los centroides marcados en rojo indican los centros geométricos de los clusters encontrados por K-Means. La cercanía de los puntos a los centroides sugiere un buen desempeño del modelo.

"Es recomendable usar todas las características disponibles siempre que sea posible, ya que brindan más información sobre los datos"

b.3.1. Modelo DBSCAN con todas las características para clustering:

b.3.1.Codigo crear, entrenar, predecir y evaluar el DBSCAN con todas las caracteristicas para los clustering:

```
# DBSCAN con todas las características
dbscan_all = DBSCAN(eps=0.5, min_samples=5)
dbscan_all_labels = dbscan_all.fit_predict(X_all_features)

# Evaluar DBSCAN
ari_dbscan_all = adjusted_rand_score(y, dbscan_all_labels)
homogeneity_dbscan_all = homogeneity_score(y, dbscan_all_labels)
print(f"\nDBSCAN usando todas las características:")
print(f"Adjusted Rand Index: {ari_dbscan_all:.2f}")
print(f"Homogeneity Score: {homogeneity_dbscan_all:.2f}")
```

b.3.1.1.ResultadoCodigo crear, entrenar, predecir y evaluar el DBSCAN con todas las caracteristicas para los clustering:

```
DBSCAN usando todas las características:
Adjusted Rand Index: 0.52
Homogeneity Score: 0.56
```

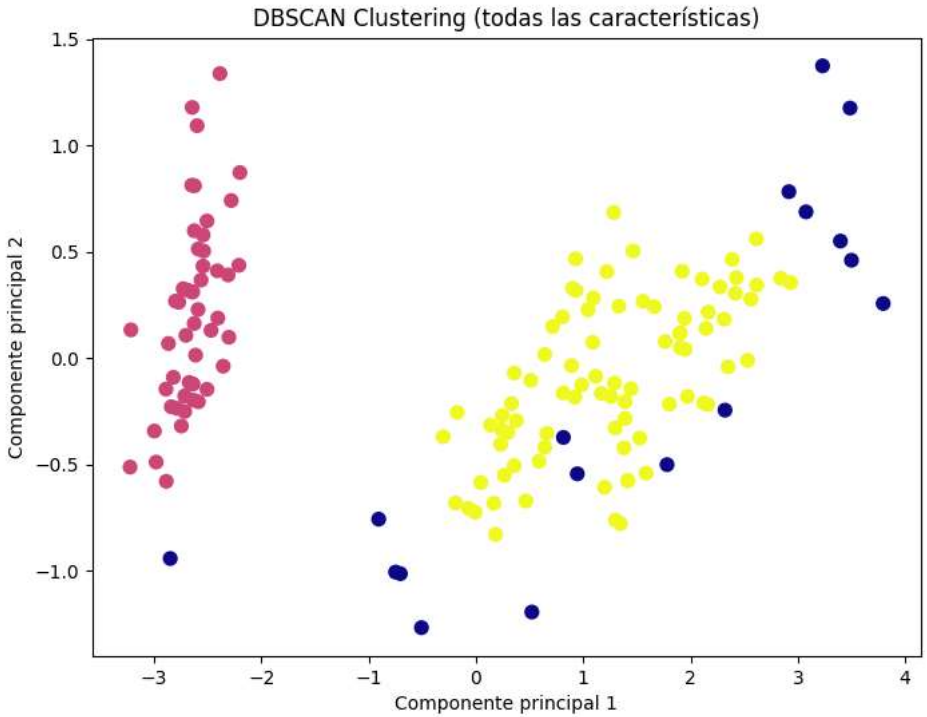
b.3.2.Codigo de visualización del modelo DBSCAN con todas las caracteristicas (4), reducido a 2D con PCA:

```
# Visualizar los clusters de DBSCAN con todas las características
plt.figure(figsize=(8,6))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=dbscan_all_labels, cmap='plasma', s=50)
plt.title("DBSCAN Clustering (todas las características)")
plt.xlabel("Componente principal 1")
plt.ylabel("Componente principal 2")
plt.show()
```

0.1s

- Se aplica la misma función PAC para poder obtener la visual en 2D como lo explique arriba en **a.3.2.**, lo que facilita la interpretación visual del clustering al proyectar las 4 dimensiones en un plano bidimensional.

b.3.2.1.Visualización del modelo DBSCAN con todas las caracteristicas (4), reducido a 2D con PCA:



b.3.3. Modelo DBSCAN con todas las características para clustering:

- Con ARI = 0.52 y Homogeneity Score = 0.56 como resultado del modelo con 4 caracteristica, es notoria la ***mejora*** respecto a **DBSCAN** con solo dos características, los valores siguen siendo ***inferiores*** a los obtenidos con **K-Means**, lo que indica que DBSCAN no logra identificar bien los clusters reales.
- En general se puede decir que DBSCAN con todas las características presenta una mejora notable, pero su rendimiento aún es inferior al de K-Means. Esto sugiere que DBSCAN no se ajusta tan bien al dataset Iris, posiblemente por la forma de los clusters y la distribución de los datos.
- DBSCAN es más sensible a la elección de parámetros (eps, min_samples) y no asume una forma particular para los clusters, por lo que puede ser menos efectivo en datos bien estructurados como Iris.

REFERENCIA BIBLIOGRAFICA:

- https://www.kranio.io/blog/fundamentos-del-machine-learning
- Aprendemachinelearning.com/comprende-principal-componet-analysis/