Clustering con K-means en el Dataset Iris

Big Data e Inteligencia Artificial

Yeray Hurtado Dragón

Noviembre 2025

Índice

1. Introducción

En esta práctica se trabaja con el algoritmo **K-means**, una técnica de *clustering* muy utilizada en el aprendizaje no supervisado. Su objetivo es agrupar datos que se parezcan entre sí, sin necesidad de conocer las etiquetas o clases reales de antemano. De esta forma, el modelo aprende a descubrir patrones y estructuras dentro de los datos de manera autónoma.

Para el desarrollo del proyecto se utilizó el clásico **dataset Iris**, que contiene medidas de tres especies de flores: *Setosa*, *Versicolor* y *Virginica*. Cada flor está descrita por cuatro características numéricas (longitud y ancho del sépalo, y longitud y ancho del pétalo). El objetivo será dividir este conjunto de datos en **tres grupos** o *clusters*, y comprobar si el algoritmo logra agrupar correctamente las flores de la misma especie.

A lo largo del trabajo se realizaron las siguientes tareas principales:

1. Implementar **K-means desde cero en Python**, para entender en detalle cada paso del algoritmo (inicialización de centroides, asignación de puntos, actualización, etc.).

2. Aplicar **K-means con librerías en Python** (usando `scikit-learn`), para comparar los resultados con una versión optimizada y más rápida.

3. Repetir ambos procesos en **R** (tanto manualmente como con la función `kmeans()`), asegurando condiciones equivalentes para comparar ambos lenguajes.

Además, se evaluaron los resultados con dos métricas: la **precisión**, que mide qué tan bien los clusters se corresponden con las clases reales del dataset, y la **inercia**, que indica qué tan compactos son los grupos creados. También se incluyeron visualizaciones mediante **PCA (Análisis de Componentes Principales)** para observar gráficamente los clusters y su coherencia.

El propósito principal de la práctica no es solo obtener buenos resultados, sino comprender cómo funciona K-means, cómo influyen sus parámetros, y cómo se comporta al implementarlo en distintos lenguajes. De esta forma, se busca un aprendizaje más completo: tanto teórico como práctico.

2. Definición del algoritmo K-Means

El algoritmo **K-Means** es un método de *clustering* o agrupamiento no supervisado que tiene como objetivo dividir un conjunto de datos en K grupos o *clusters*, de manera que los elementos pertenecientes a un mismo grupo presenten una alta similitud entre sí y una baja similitud con los elementos de otros grupos. Cada grupo se representa mediante un punto denominado **centroide**, que corresponde al promedio de los elementos que lo conforman.

Matemáticamente, el algoritmo busca minimizar la suma de las distancias cuadráticas entre cada punto y el centroide del cluster al que pertenece. La función objetivo que se desea minimizar se expresa como:

$$J = \sum_{k=1}^{K} \sum_{i:c_i=k} \|x_i - \mu_k\|^2 \tag{1}$$

donde:

- K es el número de clusters definidos.
- x_i representa un punto de datos.
- μ_k es el centroide del cluster k.
- c_i indica el cluster asignado al punto x_i.

El algoritmo K-Means sigue los siguientes pasos principales:

1. **Inicialización:** se eligen aleatoriamente K centroides iniciales.
2. **Asignación de clusters:** cada punto x_i se asigna al cluster cuyo centroide esté más cercano, utilizando generalmente la distancia euclidiana:

$$c_i = \arg\min_k \|x_i - \mu_k\|^2 \tag{2}$$

3. **Actualización de centroides:** se recalculan los centroides de cada cluster como la media de los puntos asignados a él:

$$\mu_k = \frac{1}{N_k} \sum_{i:c_i=k} x_i \tag{3}$$

   donde N_k es el número de puntos pertenecientes al cluster k.

4. **Iteración:** se repiten los pasos de asignación y actualización hasta que los centroides no varíen significativamente o se alcance un número máximo de iteraciones.

En resumen, K-Means intenta encontrar la mejor partición de los datos minimizando la variabilidad interna dentro de cada grupo y maximizando la separación entre los diferentes clusters. Es un algoritmo ampliamente utilizado por su simplicidad, eficiencia y capacidad de adaptación a distintos tipos de datos, aunque su desempeño depende de la correcta elección del número de clusters K y de la inicialización de los centroides.

3. K-Means en Python

3.1. Preparación del Dataset

En esta sección se realiza la preparación del conjunto de datos *Iris* antes de aplicar el algoritmo de *K-means*. El objetivo es disponer de los datos en un formato adecuado y observar su distribución inicial mediante técnicas de reducción de dimensionalidad.

3.1.1. Importación de librerías necesarias

Para el desarrollo de esta práctica se utilizaron librerías estándar del lenguaje Python. En concreto, se emplearon `pandas` para la manipulación de datos, `scikit-learn` para la carga del dataset y la aplicación del *Análisis de Componentes Principales* (PCA), y `matplotlib` para la representación gráfica.

```
import pandas as pd
from sklearn.datasets import load_iris
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
import numpy as np
from sklearn.cluster import KMeans
```

3.1.2. Lectura y preparación del dataset

El dataset *Iris* se encuentra disponible directamente en la librería `scikit-learn`. Se cargaron los datos y se creó un `DataFrame` con las cuatro características numéricas (longitud y ancho del sépalo, longitud y ancho del pétalo), omitiendo la columna de clase, ya que no se utilizará en el proceso de clustering.

```
# Cargar el Dataset de Iris
iris = load_iris()

X = iris.data
y = iris.target
```

3.1.3. Visualización inicial de los datos

Antes de aplicar el algoritmo de *K-means*, se realizó una visualización exploratoria para observar la distribución de los datos. Dado que el conjunto de datos tiene cuatro dimensiones, se aplicó un *Análisis de Componentes Principales* (PCA) para reducirlo a dos y tres dimensiones, permitiendo su representación gráfica. En ambas figuras los puntos aparecen en color gris, sin etiquetar, ya que el objetivo es observar la estructura natural de los datos sin influencias de las clases originales.

La Figura 1 muestra las proyecciones del dataset Iris en dos y tres dimensiones utilizando PCA. En ambos gráficos se pueden observar al menos dos agrupaciones principales de puntos, aunque al no estar coloreados según su especie, no es posible identificar claramente cada grupo. Ni la proyección en 2D ni la en 3D permiten una visualización más clara que la otra; ambas simplemente muestran la distribución general de los datos antes de aplicar el algoritmo de *K-means*.

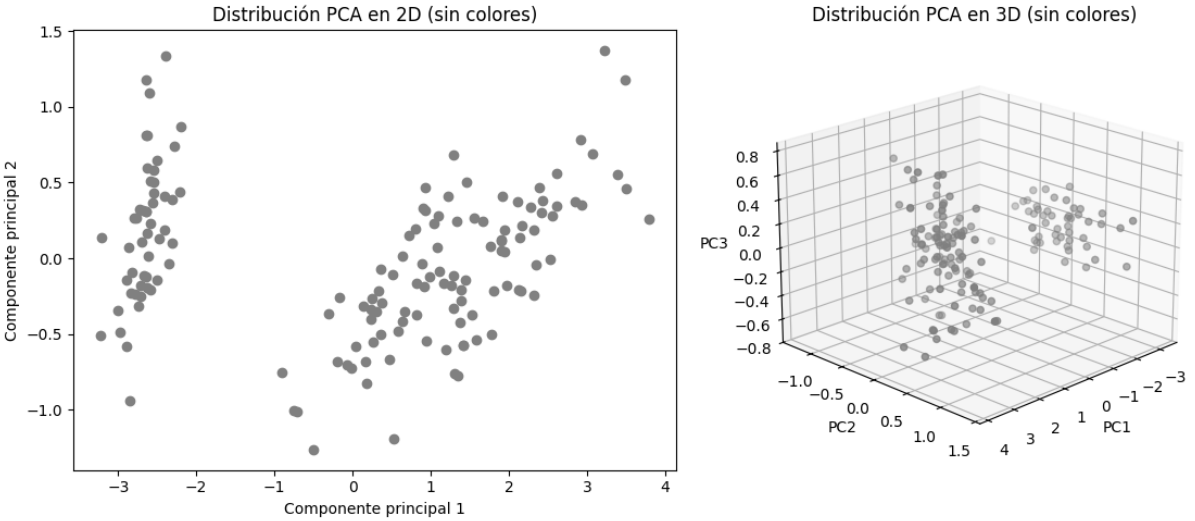


Figura 1: Distribución inicial del dataset Iris mediante PCA en 2D (izquierda) y 3D (derecha).

3.2. Implementación de K-means desde cero

En esta sección se presenta la implementación del algoritmo **K-means** en Python, desarrollado desde cero sin utilizar librerías específicas de clustering. El objetivo es entender cómo funciona el algoritmo paso a paso, dividiendo el proceso en funciones independientes que cumplen un rol específico dentro del clustering.

3.2.1. Función `euclidean_distance`

Esta función calcula la distancia euclidiana entre dos puntos en un espacio n-dimensional. Se utiliza para determinar qué tan cerca está cada punto de los centroides.

```
def euclidean_distance(point1, point2):
    return np.sqrt(np.sum((point1 - point2) ** 2))
```

3.2.2. Función `initialize_centroids`

Selecciona aleatoriamente k puntos del dataset para ser los centroides iniciales. Se utiliza una semilla (`seed`) para garantizar que los resultados sean reproducibles.

```
def initialize_centroids(data, k):
    np.random.seed(42)  # Para reproducibilidad
    random_indices = np.random.choice(data.shape[0], k, replace=
        False)
    centroids = data[random_indices]
    return centroids
```

3.2.3. Función `assign_clusters`

Asigna cada punto al centroide más cercano utilizando la distancia euclidiana. Devuelve un array que indica a qué cluster pertenece cada punto.

```
def assign_clusters(data, centroids):
    clusters = []
    for point in data:
        distances = [euclidean_distance(point, centroid) for
            centroid in centroids]
        cluster = np.argmin(distances)  # ndice del centroide
            m s cercano
        clusters.append(cluster)
    return np.array(clusters)
```

3.2.4. Función `calculate_centroids`

Calcula el nuevo centroide de cada cluster como el promedio de todos los puntos asignados a ese cluster. Si un cluster no tiene puntos asignados, se selecciona un punto aleatorio del dataset como centroide.

```
def calculate_centroids(data, clusters, k):
    centroids = []
    for i in range(k):
        cluster_points = data[clusters == i]
        if len(cluster_points) > 0:
            centroids.append(np.mean(cluster_points, axis=0))
        else:
            centroids.append(data[np.random.randint(0, data.shape
                [0])])
    return np.array(centroids)
```

3.2.5. Función principal `kmeans`

Integra todas las funciones anteriores para ejecutar el algoritmo completo de K-means. Se repite el proceso de asignar clusters y recalcular centroides hasta que los centroides no cambien o se alcance el número máximo de iteraciones.

```
def kmeans(data, k, max_iters=100):
    centroids = initialize_centroids(data, k)

    for _ in range(max_iters):
        clusters = assign_clusters(data, centroids)
        new_centroids = calculate_centroids(data, clusters, k)

        if np.all(centroids == new_centroids):
            break

        centroids = new_centroids

    return centroids, clusters
```

3.2.6. Selección del número óptimo de clusters (Método del Codo)

Antes de ejecutar K-means, es necesario elegir el número adecuado de clusters k. Para ello, se utiliza el **método del codo**, que calcula la inercia (suma de las distancias cuadradas dentro de cada cluster) para distintos valores de k y busca el punto donde la mejora deja de ser significativa.

```
def calcular_inercia(X, centroids, clusters):
    inercia = 0
    for i in range(len(X)):
        centroide = centroids[clusters[i]]
        dist = sum((X[i][j] - centroide[j])**2 for j in range(len(X
            [i])))
        inercia += dist
    return inercia

inertias = []
K = range(1, 10)
for k in K:
    centroids, clusters = kmeans(X, k)
    inertias.append(calcular_inercia(X, centroids, clusters))
```

En la Figura 2 se aprecia que la inercia disminuye rápidamente hasta $k = 3$, a partir de donde la mejora se estabiliza. Por tanto, se selecciona $k = 3$ como el número óptimo de clusters para continuar con el análisis.

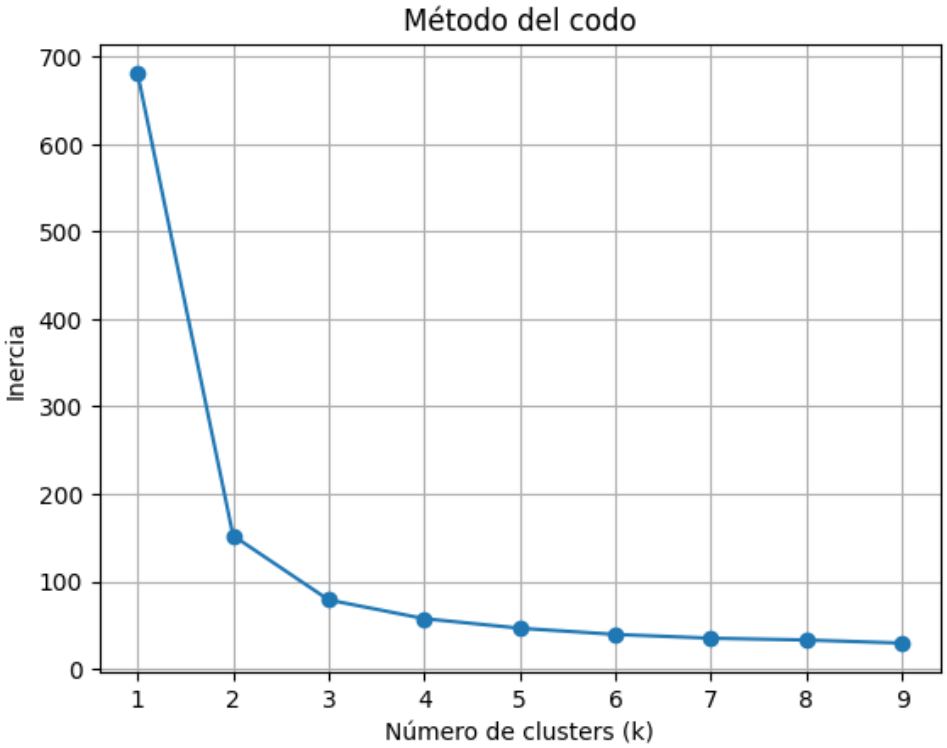


Figura 2: Curva del método del codo para determinar el número óptimo de clusters.

3.2.7. Ejecución del modelo

Con el número de clusters determinado ($K = 3$), se ejecuta el algoritmo K-means sobre el dataset *Iris*.

```
centroids, clusters = kmeans(X, k=3, max_iters=100)
```

El algoritmo inicializa tres centroides aleatorios y los actualiza iterativamente hasta que las posiciones se estabilizan o se alcanza el número máximo de iteraciones. Como resultado, se obtienen los centroides finales y la asignación de cada punto a su respectivo cluster.

3.2.8. Visualización de los Clusters

Para visualizar los resultados, se aplicó nuevamente el *Análisis de Componentes Principales* (PCA) para reducir las dimensiones a 2D y 3D. En la Figura 3 se muestran los clusters obtenidos con el valor óptimo $K = 3$, junto con los centroides calculados.

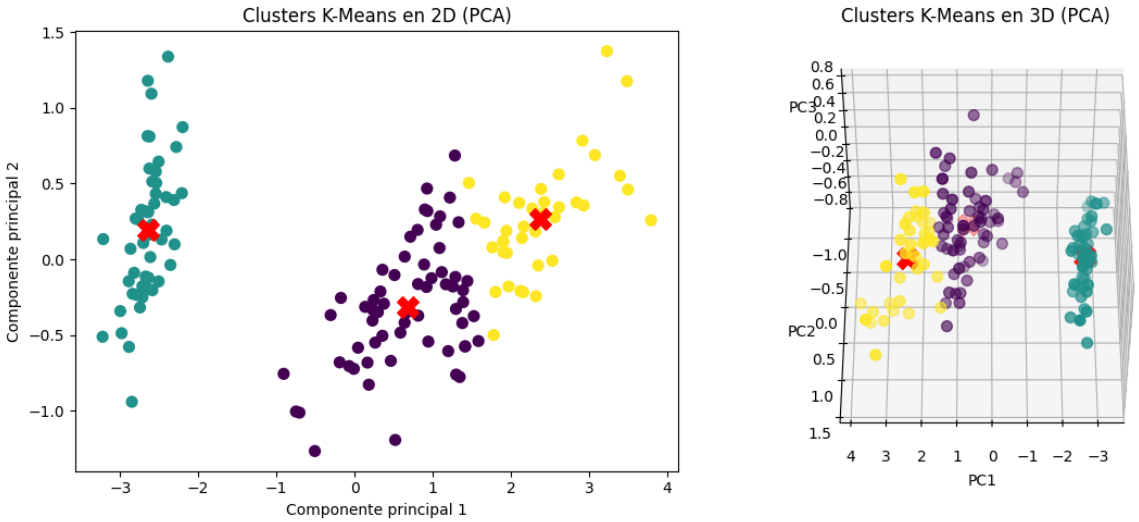


Figura 3: Visualización de los clusters obtenidos mediante K-means implementado desde cero. Los colores representan los grupos formados automáticamente, mientras que las "X-ojas marcan los centroides finales. En la vista 3D se aprecia con claridad la separación entre los tres conjuntos, especialmente el grupo más aislado, correspondiente a la especie *Setosa*.

En la Figura 3 se observa que el algoritmo logra identificar tres grupos principales. Uno de ellos aparece claramente separado del resto, mientras que los otros dos presentan cierta superposición, lo cual refleja la similitud existente entre las especies *Versicolor* y *Virginica*. En conjunto, los resultados muestran que la implementación manual del algoritmo produce una agrupación coherente con la estructura real del dataset.

3.2.9. Comparación con las clases reales

A continuación, se comparan los clusters generados por K-means (con $K = 3$) frente a las clases reales del dataset *Iris*. Ambos gráficos se obtuvieron tras aplicar una reducción de dimensionalidad mediante *PCA* a dos componentes principales, para representar los datos en un plano 2D.

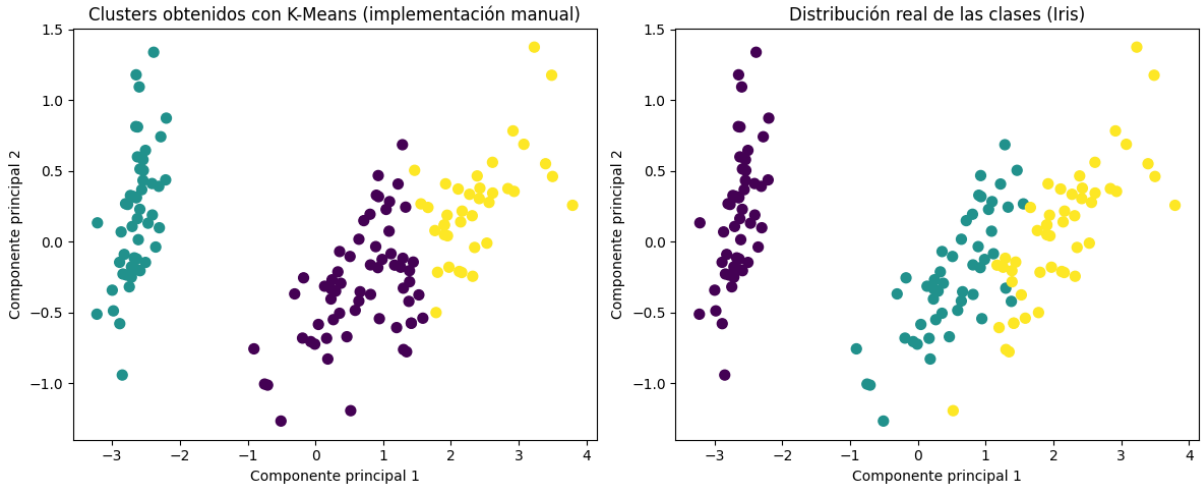


Figura 4: Comparación entre los clusters generados por la implementación manual de K-means (izquierda) y las clases reales del dataset Iris (derecha). Los colores representan los distintos grupos. Se observa que la especie *Setosa* se separa claramente del resto, mientras que *Versicolor* y *Virginica* presentan cierta superposición debido a sus similitudes morfológicas.

Como se aprecia en la Figura 4, el algoritmo K-means con $K = 3$ logra capturar de forma bastante precisa la estructura natural del conjunto de datos. La separación de la especie *Setosa* es casi perfecta, mientras que *Versicolor* y *Virginica* presentan cierta superposición. Esto se debe a que **K-means** no utiliza las etiquetas reales, sino únicamente la distancia entre los puntos y los centroides. Aun así, los resultados son coherentes y demuestran que la implementación manual del algoritmo es funcional y produce una agrupación muy similar a la clasificación real.

3.3. Implementación de K-means con librerías

En esta sección se aplica el algoritmo **K-means** utilizando la implementación disponible en la librería `scikit-learn` de Python. A diferencia de la versión programada desde cero, esta versión realiza internamente todo el proceso de inicialización, asignación y actualización de centroides, lo que simplifica notablemente el código.

3.3.1. Ejecución del modelo

El modelo se entrena directamente sobre los datos del conjunto *Iris*, indicando el número de clusters $K = 3$, obtenido previamente mediante el método del codo. El parámetro `random_state` se establece en 42 para garantizar la reproducibilidad de los resultados.

```
# Aplicar K-means con scikit-learn
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(X)

# Obtener resultados
clusters_lib = kmeans.labels_
centroids_lib = kmeans.cluster_centers_
```

3.3.2. Visualización de los clusters

Para visualizar los resultados del clustering, se aplicó nuevamente el *Análisis de Componentes Principales* (PCA) para reducir las dimensiones a 2D y 3D. La Figura 5 muestra los tres grupos obtenidos con la versión de librerías.

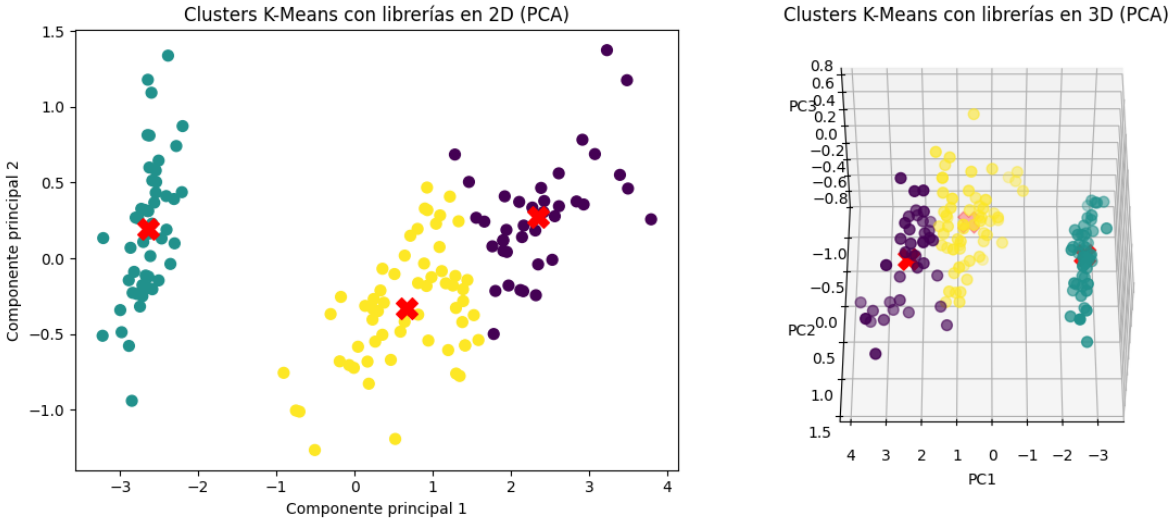


Figura 5: Clusters obtenidos mediante K-means con la librería `scikit-learn`. Se distinguen tres grupos principales, uno claramente separado (*Setosa*) y dos parcialmente solapados (*Versicolor* y *Virginica*). Las "X" marcan los centroides finales.

Los resultados son muy similares a los de la implementación manual, aunque la librería logra una convergencia más precisa gracias a su optimización interna.

3.3.3. Comparación con las clases reales

De igual forma que en la implementación manual, se realiza una comparación entre los clusters obtenidos por `scikit-learn` y las clases verdaderas del dataset. Esta comparación permite evaluar visualmente la calidad del agrupamiento.

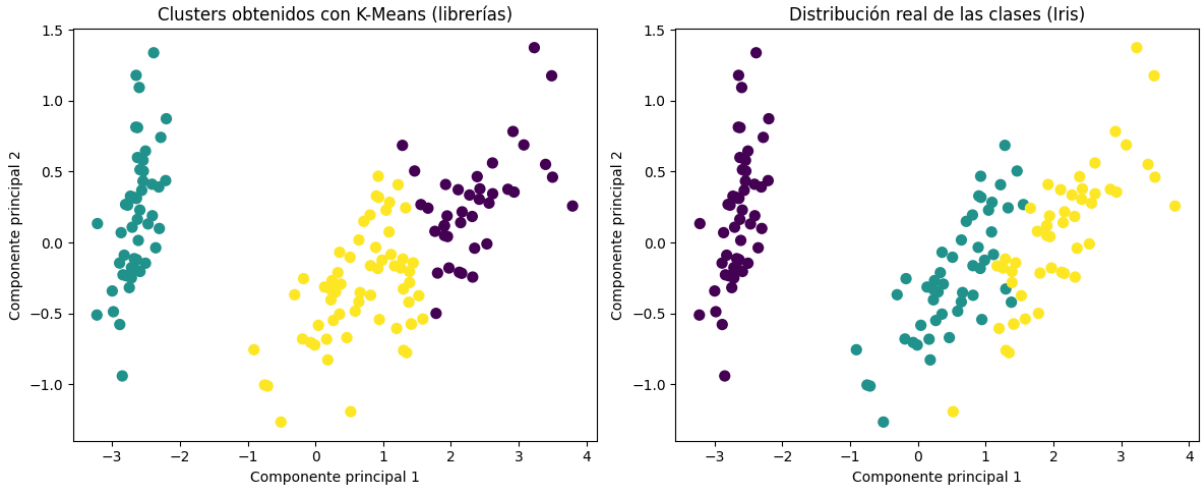


Figura 6: Comparación entre los clusters generados con la librería `scikit-learn` (izquierda) y las clases reales del dataset Iris (derecha). Se observa nuevamente una clara separación del grupo correspondiente a la especie *Setosa*, mientras que *Versicolor* y *Virginica* se mantienen parcialmente solapadas.

Los resultados obtenidos con la librería `scikit-learn` confirman la coherencia del modelo: el algoritmo detecta correctamente tres agrupaciones principales en los datos, que se corresponden con las especies reales. La ventaja principal de usar esta versión radica en la eficiencia computacional y en la simplicidad de su uso, sin necesidad de programar manualmente las funciones de distancia o actualización de centroides.

4. K-Means en R

4.1. Preparación del Dataset

En esta sección se repite el proceso de preparación del conjunto de datos *Iris*, esta vez utilizando el lenguaje R. El objetivo es disponer del mismo conjunto de datos y formato que se usó en la sección de Python, para garantizar que la comparación de resultados entre ambos lenguajes sea justa y coherente.

4.1.1. Lectura y preparación del dataset

El dataset *Iris* está incluido de forma nativa en R y contiene información sobre tres especies de flores (*Setosa*, *Versicolor* y *Virginica*), con cuatro variables numéricas: `Sepal.Length`, `Sepal.Width`, `Petal.Length` y `Petal.Width`. Al igual que en Python, se trabajará únicamente con las variables numéricas, ya que la columna de especies se empleará solo para comparar los resultados finales del clustering.

```r
data(iris)
X <- iris[, 1:4]      # las columnas de datos num ricos
y <- iris$Species      # la variable de clase
```

4.1.2. Visualización inicial de los datos

Antes de aplicar *K-means* en R, se repite el **Análisis de Componentes Principales (PCA)** para garantizar que la estructura del dataset sea equivalente a la usada en Python. Esto permite comparar los resultados bajo las mismas condiciones. La Figura 7 muestra la proyección en 2D y 3D,confirmando que los datos están listos para el clustering.

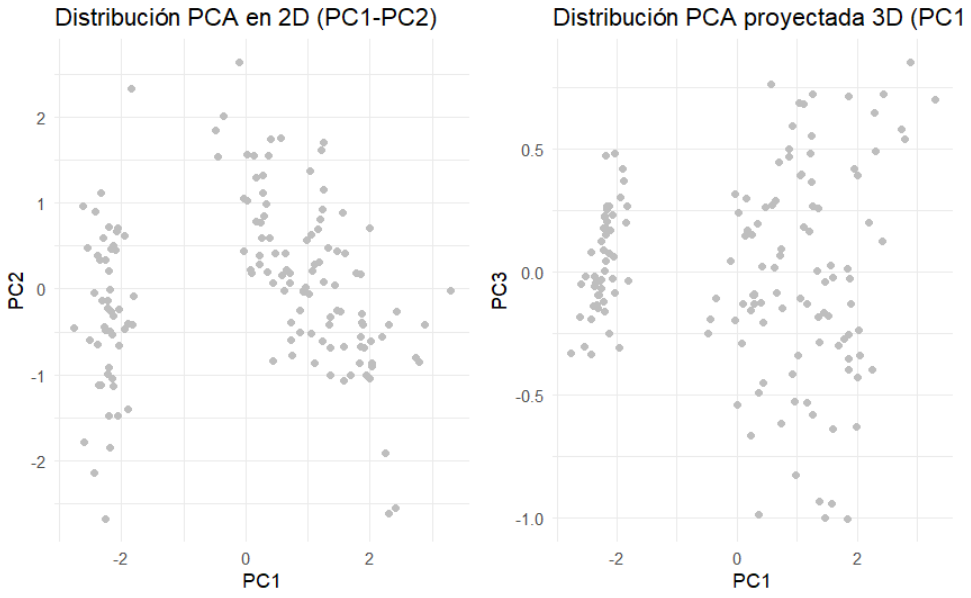


Figura 7: Distribución inicial del dataset *Iris* en R mediante PCA en 2D (izquierda) y 3D (derecha), con el mismo ángulo de visión que en la versión de Python.

4.2. Implementación de K-means desde cero

En esta sección se desarrolla una versión manual del algoritmo **K-means** en R, equivalente a la implementada en Python. Se utilizan únicamente funciones base del lenguaje, sin recurrir a librerías externas, para comprender el funcionamiento interno del proceso de agrupamiento.

4.2.1. Definición de funciones básicas

A continuación, se implementan las funciones principales: cálculo de distancia, inicialización de centroides, asignación de clusters y actualización de centroides. Cada función cumple el mismo propósito que en la versión Python, adaptada a la sintaxis del lenguaje R.

```r
# Calcular distancia euclidiana
euclidean_distance <- function(p1, p2) {
  sqrt(sum((p1 - p2)^2))
}

# Inicializar centroides aleatoriamente
initialize_centroids <- function(data, k) {
  set.seed(42)  # reproducibilidad
  indices <- sample(1:nrow(data), k)
  data[indices, ]
}

# Asignar cada punto al centroide m s cercano
assign_clusters <- function(data, centroids) {
  apply(data, 1, function(point) {
    distances <- apply(centroids, 1, function(c) euclidean_distance
       (point, c))
    which.min(distances)
  })
}

# Calcular nuevos centroides
calculate_centroids <- function(data, clusters, k) {
  centroids <- matrix(NA, nrow = k, ncol = ncol(data))
  for (i in 1:k) {
    cluster_points <- data[clusters == i, , drop = FALSE]
    if (nrow(cluster_points) > 0) {
      centroids[i, ] <- colMeans(cluster_points)
    } else {
      centroids[i, ] <- data[sample(1:nrow(data), 1), ]
    }
  }
  centroids
}
```

4.2.2. Función principal `kmeans_manual`

Finalmente, se define la función principal `kmeans_manual()`, que integra todos los pasos del algoritmo. El proceso se repite hasta que los centroides dejan de cambiar significativamente o se alcanza el número máximo de iteraciones.

```
kmeans_manual <- function(data, k, max_iters = 100) {
  centroids <- initialize_centroids(data, k)

  for (i in 1:max_iters) {
    clusters <- assign_clusters(data, centroids)
    new_centroids <- calculate_centroids(data, clusters, k)

    if (all(centroids == new_centroids)) break
    centroids <- new_centroids
  }

  list(centroids = centroids, clusters = clusters)
}

# Ejemplo de ejecuci n
result <- kmeans_manual(X, k = 3)
centroids <- result$centroids
clusters <- result$clusters
```

4.2.3. Visualización de resultados

Al igual que en la implementación realizada en Python, se aplicó el **Análisis de Componentes Principales (PCA)** para reducir las cuatro dimensiones del conjunto de datos a dos componentes principales y así representar gráficamente los resultados del clustering. El proceso de reducción y visualización se mantuvo idéntico al descrito anteriormente, garantizando que las comparaciones entre ambos lenguajes sean justas y consistentes.

En la Figura 8 se muestran los tres clusters obtenidos mediante la versión manual del algoritmo en R, junto con sus centroides finales marcados con una "X". Los colores representan los grupos formados automáticamente por el modelo, sin utilizar las etiquetas originales del dataset.

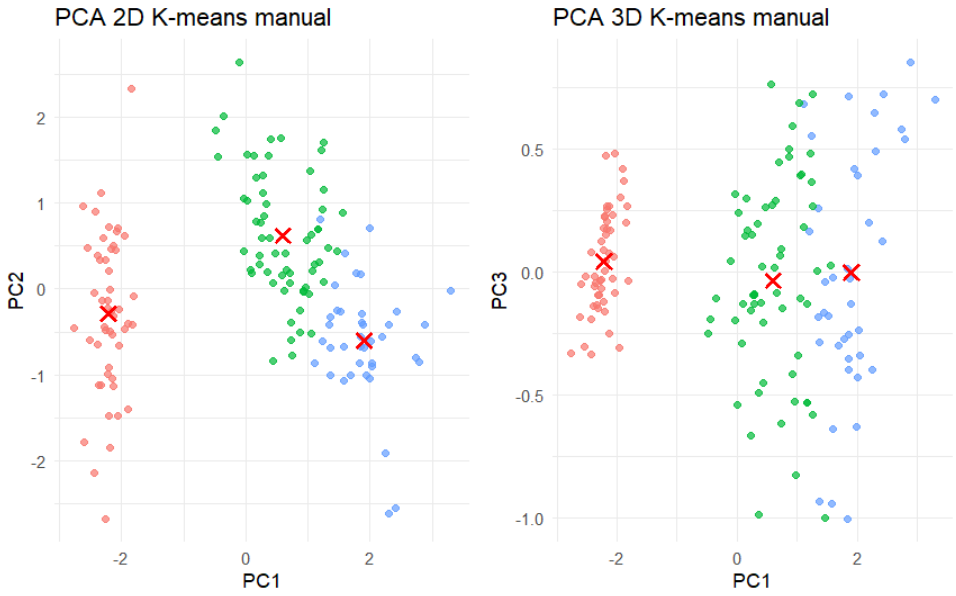


Figura 8: Visualización de los clusters obtenidos mediante la implementación manual de K-means en R. Los colores representan los grupos detectados automáticamente, y las "X" marcan los centroides finales calculados.

De forma similar a los resultados obtenidos en Python, se observa que el algoritmo identifica claramente tres agrupaciones principales. Uno de los grupos se encuentra perfectamente separado del resto, correspondiente a la especie *Setosa*, mientras que los otros dos presentan cierta superposición, reflejando la similitud entre *Versicolor* y *Virginica*. Estos resultados confirman que la implementación en R reproduce fielmente el comportamiento del algoritmo, mostrando una estructura de clusters coherente y consistente con la distribución natural de los datos del conjunto *Iris*.

4.2.4. Comparación con las clases reales

La comparación de los clusters obtenidos mediante la implementación manual de K-means en R con las especies reales del dataset *Iris* se realizó siguiendo la misma metodología que se describió para Python en la Sección 4. Se utilizó un **Análisis de Componentes Principales (PCA)** para reducir las dimensiones a 2D, permitiendo representar los clusters junto a las clases verdaderas en un plano.

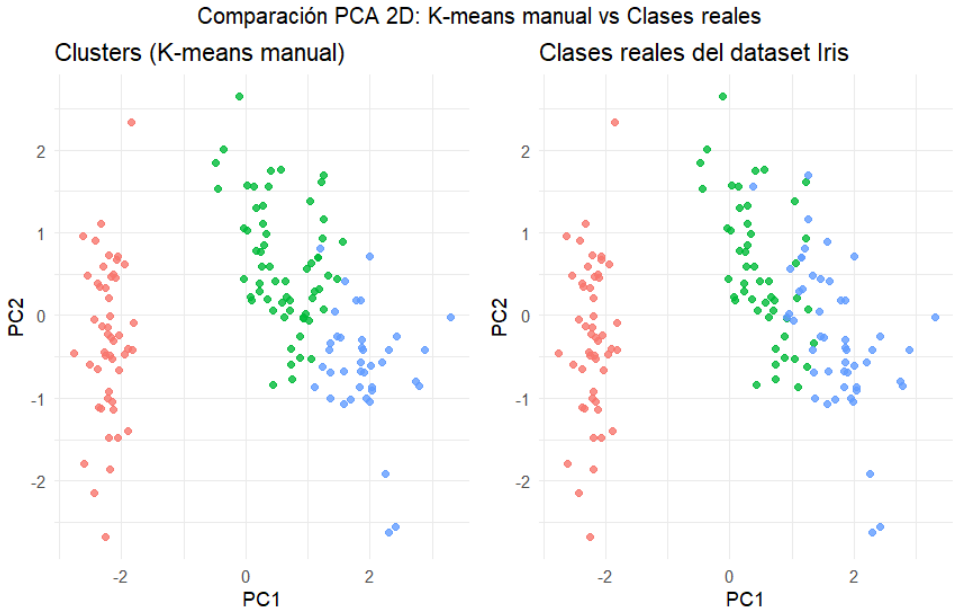


Figura 9: Comparación entre los clusters generados manualmente en R y las clases reales del dataset Iris. Los colores representan los distintos grupos. Se observa la misma tendencia que en Python: la especie *Setosa* se separa claramente, mientras que *Versicolor* y *Virginica* presentan cierta superposición.

Los resultados obtenidos en R son consistentes con los observados en Python, identificando claramente tres agrupaciones principales y reflejando la estructura natural del dataset. Esto confirma que la implementación manual del algoritmo en R funciona correctamente y produce resultados comparables a los de Python.

4.3. Implementación de K-means con librerías

En esta sección se aplica el algoritmo **K-means** utilizando la función `kmeans()` de R, que simplifica todo el proceso de inicialización, asignación y actualización de centroides. Se mantiene el mismo número de clusters ($K = 3$) utilizado en las implementaciones manuales y en Python, para permitir una comparación coherente de resultados.

4.3.1. Ejecución del modelo

El modelo se aplica directamente sobre las columnas numéricas del dataset *Iris*, generando los clusters automáticamente. Se emplea un parámetro `nstart=25` para garantizar una buena inicialización de los centroides y reproducibilidad del resultado.

```
set.seed(42)
kmeans_lib <- kmeans(X,centers = 3)

clusters_lib <- kmeans_lib$cluster
centroids_lib <- kmeans_lib$centers
```

4.3.2. Visualización de resultados

Para visualizar los clusters obtenidos con la función de librerías, se proyectaron nuevamente los datos al espacio PCA, siguiendo el mismo procedimiento descrito en los apartados anteriores de *Visualización de resultados* y *Comparación con las clases reales*. Esto permite comparar directamente los resultados de la versión manual y la de librerías, garantizando consistencia en la representación gráfica.

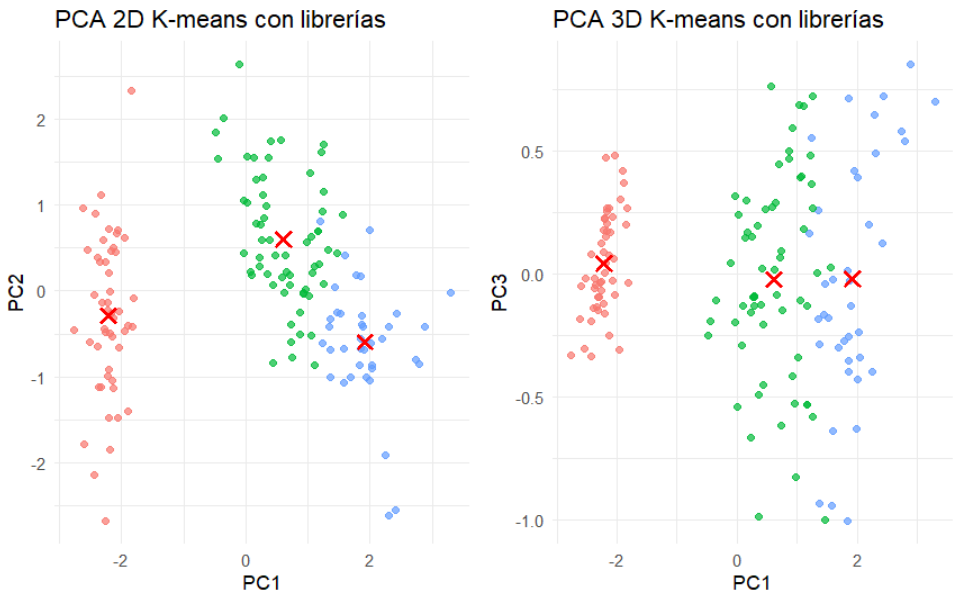


Figura 10: Clusters obtenidos mediante K-means con la función `kmeans()` de R. Los colores representan los grupos detectados automáticamente y las "X" rojas marcan los centroides finales. Se observa una clara separación de *Setosa*, mientras que *Versicolor* y *Virginica* presentan cierta superposición, consistente con los resultados manuales y con Python.

4.3.3. Comparación con las clases reales

Siguiendo la misma metodología empleada en los apartados anteriores (*Comparación con las clases reales*), se compararon los clusters generados por `kmeans()` con las especies reales del dataset. La Figura 11 muestra esta comparación en un espacio reducido a dos componentes principales mediante PCA.

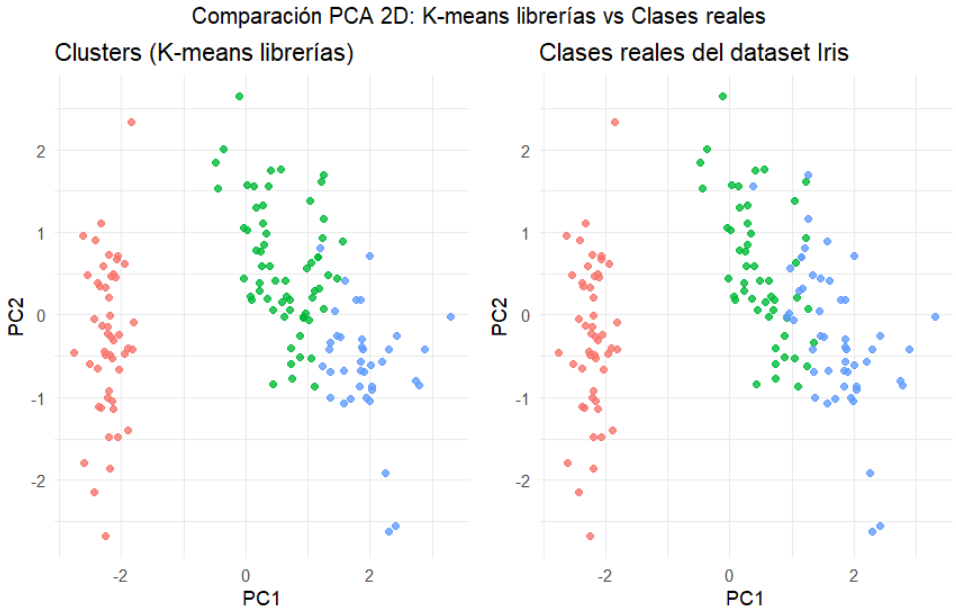


Figura 11: Comparación entre los clusters obtenidos con la librería `kmeans()` en R y las clases reales del dataset Iris. Se observa que la especie *Setosa* se separa claramente, mientras que *Versicolor* y *Virginica* mantienen una ligera superposición, replicando la tendencia observada en implementaciones manuales y en Python.

Los resultados confirman que la función `kmeans()` de R identifica correctamente las tres agrupaciones principales y proporciona resultados consistentes con las implementaciones manuales y la versión en Python, validando la coherencia del algoritmo a través de diferentes lenguajes e implementaciones.

5. Comparación global

En esta sección se comparan los resultados obtenidos por las distintas implementaciones del algoritmo K-means en Python y R, tanto en sus versiones manuales como utilizando librerías. Se analizan dos métricas principales: la **precisión**, que mide qué tan bien los clusters coinciden con las clases reales, y la **inercia**, que representa la compactación de los grupos formados. Además, se observa visualmente la coherencia de los clusters obtenidos.

5.1. Tabla comparativa de métricas

Implementación	**Precisión**	**Inercia**
Python manual	0.89	78.8
Python con scikit-learn	0.89	78.8
R manual	0.88	78.8
R con kmeans()	0.89	78.8

Cuadro 1: Comparación de precisión e inercia entre las implementaciones de K-means en Python y R. Los valores pueden variar ligeramente entre ejecuciones.

Los resultados mostrados en la Tabla 1 reflejan una gran similitud entre las distintas implementaciones, tanto en precisión como en inercia. Esto ocurre porque, a pesar de las diferencias en código o lenguaje, el **algoritmo de K-means sigue exactamente el mismo procedimiento matemático** en todos los casos: calcula distancias euclidianas, asigna puntos al centroide más cercano y actualiza los centroides repitiendo el proceso hasta converger.

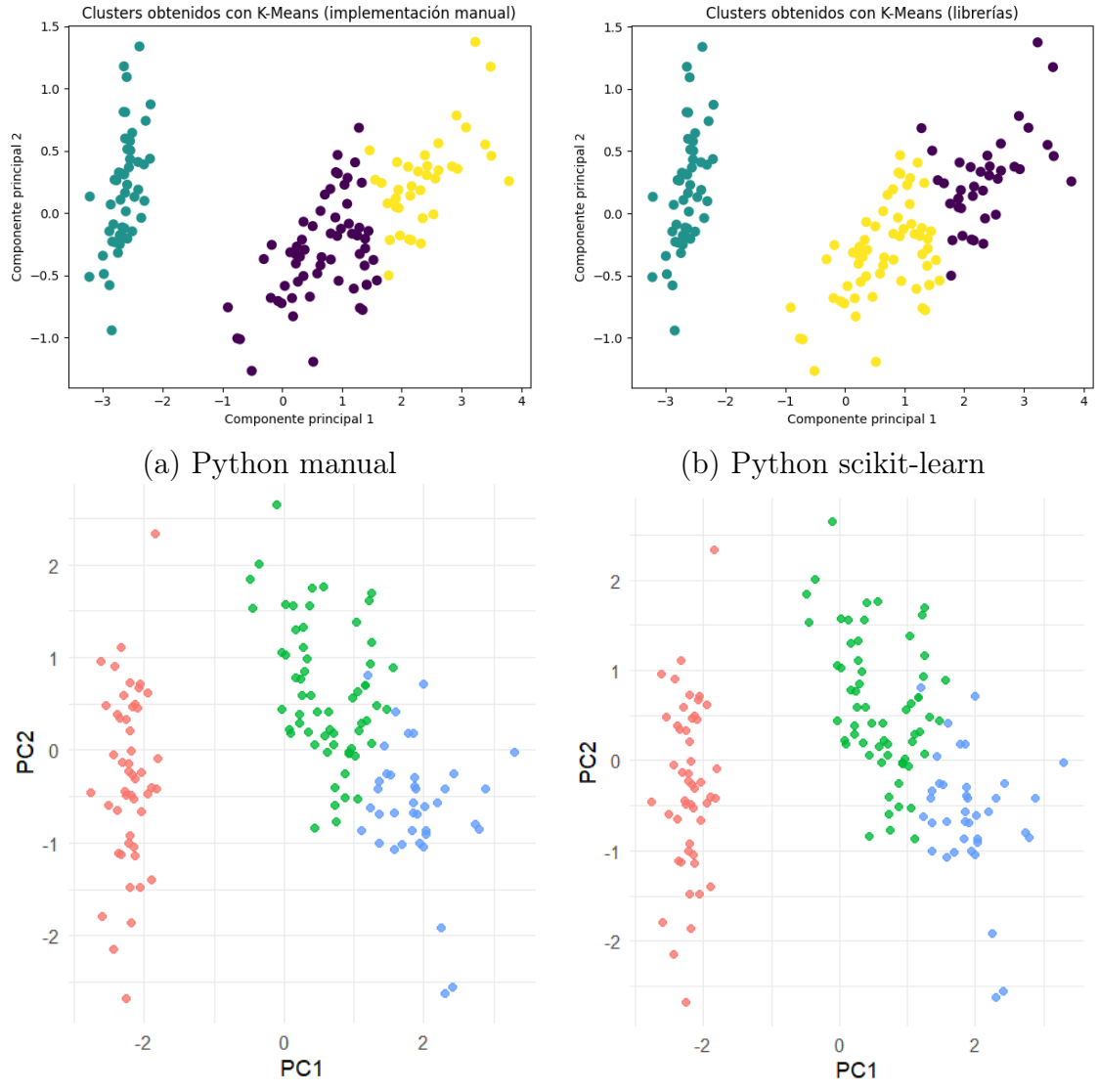
En otras palabras, tanto las versiones manuales como las de librería realizan las mismas operaciones, solo que las librerías lo hacen de forma más eficiente y optimizada. Por eso, las pequeñas diferencias (de milésimas) que pueden aparecer en la inercia o la precisión se deben principalmente a factores como:

- La **inicialización aleatoria de los centroides**, que puede generar ligeras variaciones si la semilla no es exactamente la misma.
- La **tolerancia numérica de los cálculos**, ya que cada lenguaje maneja las operaciones con pequeñas diferencias de redondeo.
- El **criterio de parada** (cuándo se considera que los centroides dejaron de moverse) puede variar levemente entre implementaciones.

Aun así, las diferencias son mínimas y los valores finales prácticamente coinciden. Esto demuestra que el algoritmo K-means es muy estable y que su comportamiento no depende del lenguaje o de la librería utilizada, sino de los propios datos y la forma en que están distribuidos. En el caso del dataset *Iris*, los tres grupos naturales (correspondientes a las especies de flores) están bien definidos, por lo que cualquier implementación de K-means tiende a encontrar los mismos clusters con resultados casi idénticos.

5.2. Comparación visual de clusters

Para entender mejor los resultados, se muestran los clusters en dos dimensiones (usando PCA) comparados con las clases reales del conjunto de datos *Iris*. En la Figura 12 se incluyen las representaciones para las cuatro versiones del algoritmo:



(a) Python manual

(b) Python scikit-learn

(c) R manual

(d) R kmeans()

Figura 12: Comparación visual de los clusters obtenidos en 2D mediante PCA frente a las clases reales del dataset *Iris*.

En todas las gráficas se observa que la especie *Setosa* forma un grupo completamente separado, mientras que *Versicolor* y *Virginica* se superponen parcialmente. Esto tiene sentido, ya que estas dos últimas especies son morfológicamente muy parecidas, lo que dificulta que un algoritmo no supervisado como K-means las distinga por completo.

5.3. Observaciones generales

- Las versiones manuales ayudan a entender mejor cómo funciona el algoritmo paso a paso (asignación de clusters, cálculo de centroides, etc.), aunque son un poco más lentas.
- Las versiones con librerías (`scikit-learn` en Python y `kmeans()` en R) son mucho más rápidas y estables, ya que están optimizadas internamente.
- En todas las implementaciones, el número de clusters $k = 3$ (obtenido con el método del codo) resultó ser el más adecuado, ya que coincide con las tres especies del dataset.
- La precisión y la inercia son prácticamente iguales en todos los casos, lo cual tiene sentido porque el algoritmo es el mismo y los datos también. La diferencia entre hacerlo "a mano" o con una librería solo afecta la velocidad y la forma en que se inicializan los centroides, pero no cambia el resultado final de forma importante.

En resumen, los resultados son muy parecidos porque K-means sigue exactamente los mismos pasos matemáticos en cualquier lenguaje. Tanto las versiones manuales como las de librería terminan agrupando los puntos de forma muy similar, ya que el conjunto de datos *Iris* tiene una estructura muy clara con tres grupos naturales. Esto demuestra que el algoritmo es bastante robusto y consistente, sin importar si se usa Python o R para implementarlo.

6. Conclusiones

Tras la realización de la práctica y el análisis de resultados, se pueden extraer las siguientes conclusiones:

1. **Eficacia del algoritmo K-means:** El algoritmo logró identificar correctamente las tres especies del dataset Iris, separando perfectamente la especie *Setosa* y agrupando de manera razonable a *Versicolor* y *Virginica*, aunque con cierta superposición debido a su similitud morfológica.

2. **Comparación Python vs R:** Los resultados entre Python y R son consistentes, tanto en implementaciones manuales como con librerías. Esto demuestra que K-means es robusto frente a diferentes lenguajes y entornos.

3. **Implementaciones manuales:** Son útiles para comprender el algoritmo, el cálculo de distancias, la asignación de clusters y la actualización de centroides, pero presentan un mayor costo computacional y requieren más líneas de código.

4. **Uso de librerías:** 'scikit-learn' y 'kmeans()' en R simplifican la implementación, optimizan la convergencia y ofrecen resultados precisos y reproducibles con menos esfuerzo. Para aplicaciones prácticas y datasets grandes, esta opción es preferible.

5. **Lecciones generales:** K-means es eficiente y sencillo para datos con clusters compactos y bien separados. Sin embargo, su desempeño depende de la correcta elección de k y puede verse limitado ante grupos solapados o no esféricos. La práctica refuerza la importancia de evaluar tanto métricas cuantitativas (precisión, inercia) como visuales (distribución de clusters) para validar los resultados.

En conclusión, la práctica permite comparar métodos manuales y automáticos de K-means, evaluando su precisión y eficiencia. La elección del enfoque dependerá del objetivo: aprendizaje profundo del algoritmo o aplicación rápida y eficiente a datos reales.