PREGUNTA N°2

1)Elgoritmo K-Means es una técnica de agrupamiento que se utiliza para dividir un conjunto de datos en kkk grupos distintos (clusters)

PASOS:

**Asignación de Clusters**:

**Recalculación de Centroides**:

l algoritmo K-Means converge cuando los centroides ya no se mueven significativamente

2) limpeiza de datos

Transformacion de datos

Reduccion de dimensionalidad

3) METODO DEL CODO : Este método implica ejecutar K-Means para diferentes valores de kkk y calcular la suma de los errores cuadráticos internos (inertia) para cada kkk. La inertia mide cuán dispersos están los puntos dentro de un cluster.

METODO DE LA SILUETA: Calcula el coeficiente de silueta para diferentes valores de kkk. El coeficiente de silueta mide cuán similar es un punto a su propio cluster en comparación con otros clusters.

METODO DE LA BRECHA : Compara la variación dentro de los clusters con la variación esperada bajo una referencia nula aleatoria

4)

5) **Cluster 0**: Podría representar un grupo de postulantes con ciertas características comunes, como una puntuación de examen específica, un rango de edad similar, etc.

**Cluster 1**: Otro grupo con características diferentes pero similares entre sí, como un promedio de notas específico, otro rango de edad, etc.

**Cluster 2**: Otro grupo distinto con sus propias características comunes.

Otro interpretación:

Cluster 0: Podría agrupar a postulantes con notas altas y edades más jóvenes.

Cluster 1: Podría agrupar a postulantes con notas medias y edades intermedias.

Cluster 2: Podría agrupar a postulantes con notas bajas y edades más avanzadas.

CODIGO :

# Importar librerías necesarias

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.cluster import KMeans

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import silhouette\_score

from sklearn.ensemble import RandomForestClassifier

from sklearn.decomposition import PCA

# Leer el dataset

archivo = "Postulantes\_UNI.csv"

df = pd.read\_csv(archivo, encoding='latin1')

# Limpieza de datos

df.drop\_duplicates(inplace=True)

# Reemplazar valores no numéricos con NaN

df.replace('?', np.nan, inplace=True)

# Identificar valores nulos

print("Valores nulos por columna:")

print(df.isnull().sum())

# Imputar valores nulos en columnas numéricas

numeric\_columns = df.select\_dtypes(include=[np.number]).columns

df[numeric\_columns] = df[numeric\_columns].fillna(df[numeric\_columns].mean())

# Verificar nuevamente valores nulos

print("Valores nulos por columna después de la imputación:")

print(df.isnull().sum())

# Identificación y manejo de outliers

def detectar\_outliers\_iqr(df):

    outliers = pd.DataFrame()

    for col in df.select\_dtypes(include=[np.number]).columns:

        Q1 = df[col].quantile(0.25)

        Q3 = df[col].quantile(0.75)

        IQR = Q3 - Q1

        lower\_bound = Q1 - 1.5 \* IQR

        upper\_bound = Q3 + 1.5 \* IQR

        outliers[col] = ((df[col] < lower\_bound) | (df[col] > upper\_bound))

    return outliers

outliers = detectar\_outliers\_iqr(df)

print("Número de outliers por columna:")

print(outliers.sum())

# Opcional: eliminar outliers

# df = df[~outliers.any(axis=1)]

# Estandarización de los datos

scaler = StandardScaler()

df\_scaled = scaler.fit\_transform(df.select\_dtypes(include=[np.number]))

# Método del Codo y de la Silueta para determinar el número óptimo de clusters

inertia = []

silhouette\_scores = []

K = range(2, 11)

for k in K:

    kmeans = KMeans(n\_clusters=k, random\_state=0)

    kmeans.fit(df\_scaled)

    inertia.append(kmeans.inertia\_)

    labels = kmeans.labels\_

    silhouette\_scores.append(silhouette\_score(df\_scaled, labels))

# Gráfico del Método del Codo

plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)

plt.plot(K, inertia, 'bx-')

plt.xlabel('Número de Clusters (k)')

plt.ylabel('Inertia')

plt.title('Método del Codo para encontrar el k óptimo')

# Gráfico del Método de la Silueta

plt.subplot(1, 2, 2)

plt.plot(K, silhouette\_scores, 'bx-')

plt.xlabel('Número de Clusters (k)')

plt.ylabel('Coeficiente de Silueta')

plt.title('Método de la Silueta para encontrar el k óptimo')

plt.show()

# Aplicar K-Means con el número óptimo de clusters (por ejemplo, k=3)

k\_optimo = 3  # Ajustar según los resultados obtenidos de los gráficos

kmeans = KMeans(n\_clusters=k\_optimo, random\_state=0)

kmeans.fit(df\_scaled)

labels = kmeans.labels\_

# Añadir las etiquetas de los clusters al DataFrame original

df['Cluster'] = labels

# Mostrar los centroides de los clusters

centroids = kmeans.cluster\_centers\_

print("Centroides de los clusters:")

print(centroids)

# Visualización de resultados (si es aplicable)

plt.figure(figsize=(8, 6))

plt.scatter(df\_scaled[:, 0], df\_scaled[:, 1], c=labels, cmap='viridis')

plt.scatter(centroids[:, 0], centroids[:, 1], s=300, c='red')

plt.xlabel('Feature 1')

plt.ylabel('Feature 2')

plt.title('K-Means Clustering')

plt.show()

# Análisis de Correlación

plt.figure(figsize=(10, 8))

sns.heatmap(df.corr(), annot=True, cmap='coolwarm')

plt.title('Mapa de Calor de Correlación')

plt.show()

# Importancia de Características con Random Forest

rf = RandomForestClassifier()

rf.fit(df.drop(columns=['Cluster']), df['Cluster'])

importances = rf.feature\_importances\_

indices = np.argsort(importances)[::-1]

plt.figure(figsize=(10, 6))

plt.title("Importancia de Características")

plt.bar(range(df.shape[1] - 1), importances[indices], align="center")

plt.xticks(range(df.shape[1] - 1), df.columns[indices], rotation=90)

plt.xlim([-1, df.shape[1] - 1])

plt.show()

# Reducción de Dimensionalidad con PCA

pca = PCA(n\_components=2)

principal\_components = pca.fit\_transform(df\_scaled)

plt.figure(figsize=(10, 6))

plt.scatter(principal\_components[:, 0], principal\_components[:, 1], c=labels, cmap='viridis')

plt.xlabel('Componente Principal 1')

plt.ylabel('Componente Principal 2')

plt.title('Reducción de Dimensionalidad con PCA')

plt.show()

print("Variancia explicada por cada componente principal:")

print(pca.explained\_variance\_ratio\_)

# Evaluación de Clusters

for cluster\_num in range(k\_optimo):

    print(f"Cluster {cluster\_num}:")

    print(df[df['Cluster'] == cluster\_num].describe())

# Análisis de Silueta por Característica

from sklearn.metrics import silhouette\_samples

silhouette\_vals = silhouette\_samples(df\_scaled, labels)

df['Silhouette'] = silhouette\_vals

for column in df.columns:

    if column != 'Cluster' and column != 'Silhouette':

        print(f"Silhouette para {column}: {df.groupby('Cluster')[column].mean().std() / df[column].std()}")