Instituto Politécnico Nacional

Escuela Superior de Computo

PROYECTO APRENDIZAJE MAQUINA

Inteligencia Artificial

Profesor:

Catalán Salgado Edgar Armando

Alumnos:

Briones Cruz Juan Carlos

Espinosa Vergara David Daniel

Hernández Reyes Julio Cesar

Vargas Velez Angel Isaac

Fecha:

12 Enero 2024

1. Debe describir cada uno de los atributos al momento(si son mas de 10, solo describir 10), indicando:
    a. Tipo Dato(Numerico, categorico)
        i. En caso de ser numerico, min, max, promedio y desviacion estandar
        ii. En caso de ser categorico, las categorias

Lee un conjunto de datos desde un archivo CSV usando la biblioteca panda. Muestra información sobre los atributos y etiquetas del conjunto de datos, incluyendo tipos de datos, estadísticas descriptivas para atributos numéricos y categorías únicas para atributos categóricos.

```
################################################################################

Punto 1:

Atributo 1: LargoSepalo
Tipo de dato: float64
Mínimo: 4.3
Máximo: 7.9
Promedio: 5.84
Desviación Estándar: 0.83

Atributo 2: AnchoSepalo
Tipo de dato: float64
Mínimo: 2.0
Máximo: 4.4
Promedio: 3.05
Desviación Estándar: 0.43

Atributo 3: LargoPetalo
Tipo de dato: float64
Mínimo: 1.0
Máximo: 6.9
Promedio: 3.76
Desviación Estándar: 1.76

Atributo 4: AnchoPetalo
Tipo de dato: float64
Mínimo: 0.1
Máximo: 2.5
Promedio: 1.20
Desviación Estándar: 0.76

Etiqueta Y:
Tipo de dato: object
Categorías: ['Iris-setosa' 'Iris-versicolor' 'Iris-virginica']
```

```python
# Describir cada uno de los atributos al momento
def punto1(dataset_path):

    # Cargar el conjunto de datos usando pandas, indicando que la primera columna es el índice
    df = pd.read_csv(dataset_path)

    # Obtener los atributos y la etiqueta
    X = df.iloc[:, :-1]
    y = df.iloc[:, -1]

    # Descripción de los atributos del vector de entrada X
    for idx, column in enumerate(X.columns):
        print(f"\nAtributo {idx + 1}: {column}")
        print(f"Tipo de dato: {X[column].dtype}")

        if X[column].dtype == 'object':
            # Si es categórico
            print(f"Categorías: {X[column].unique()}")
        else:
            # Si es numérico
            print(f"Mínimo: {X[column].min()}")
            print(f"Máximo: {X[column].max()}")
            print(f"Promedio: {X[column].mean():.2f}")
            print(f"Desviación Estándar: {X[column].std():.2f}")

    # Descripción de la etiqueta de salida Y
    print(f"\nEtiqueta Y:")
    print(f"Tipo de dato: {y.dtype}")
    print(f"Categorías: {y.unique()}")

    return df, X, y
```

2. Definir los atributos del vector de entrada X y de salida(clase) Y
    a. Por cada clase obtener:
        i. Max, min, prom y desviacion estandar de cada uno de los atributos en el vector de entrada
        ii. Las categorias en caso de los datos categoricos

Utiliza el conjunto de datos cargado en el Punto 1 para mostrar estadísticas descriptivas separadas por clase. Muestra información sobre atributos numéricos y categóricos para cada clase.

```
##################################################

Punto 2:

Estadísticas para la Clase Iris-setosa:

Atributo 1: LargoSepalo
Máximo: 5.8
Mínimo: 4.3
Promedio: 5.01
Desviación Estándar: 0.35

Atributo 2: AnchoSepalo
Máximo: 4.4
Mínimo: 2.3
Promedio: 3.42
Desviación Estándar: 0.38

Atributo 3: LargoPetalo
Máximo: 1.9
Mínimo: 1.0
Promedio: 1.46
Desviación Estándar: 0.17

Atributo 4: AnchoPetalo
Máximo: 0.6
Mínimo: 0.1
Promedio: 0.24
Desviación Estándar: 0.11
```

```
Estadísticas para la Clase Iris-versicolor:

Atributo 1: LargoSepalo
Máximo: 7.0
Mínimo: 4.9
Promedio: 5.94
Desviación Estándar: 0.52

Atributo 2: AnchoSepalo
Máximo: 3.4
Mínimo: 2.0
Promedio: 2.77
Desviación Estándar: 0.31

Atributo 3: LargoPetalo
Máximo: 5.1
Mínimo: 3.0
Promedio: 4.26
Desviación Estándar: 0.47

Atributo 4: AnchoPetalo
Máximo: 1.8
Mínimo: 1.0
Promedio: 1.33
Desviación Estándar: 0.20
```

```
Estadísticas para la Clase Iris-virginica:

Atributo 1: LargoSepalo
Máximo: 7.9
Mínimo: 4.9
Promedio: 6.59
Desviación Estándar: 0.64

Atributo 2: AnchoSepalo
Máximo: 3.8
Mínimo: 2.2
Promedio: 2.97
Desviación Estándar: 0.32

Atributo 3: LargoPetalo
Máximo: 6.9
Mínimo: 4.5
Promedio: 5.55
Desviación Estándar: 0.55

Atributo 4: AnchoPetalo
Máximo: 2.5
Mínimo: 1.4
Promedio: 2.03
Desviación Estándar: 0.27
```

```
Etiqueta Y:
Tipo de dato: object
Categorías: ['Iris-setosa' 'Iris-versicolor' 'Iris-virginica']
```

```python
# Definir los atributos del vector de entrada X y de salida(clase) Y
def punto2(df):

    # Obtener los atributos y la etiqueta
    X = df.iloc[:, :-1]
    y = df.iloc[:, -1]

    # Por cada clase en Y
    clases = y.unique()
    for clase in clases:
        print(f"\nEstadísticas para la Clase {clase}:")
        # Filtrar las instancias correspondientes a la clase actual
        instancias_clase = df[df.iloc[:, -1] == clase].iloc[:, :-1]

        # Obtener las estadísticas
        for idx, column in enumerate(instancias_clase.columns):
            print(f"\nAtributo {idx + 1}: {column}")
            print(f"Máximo: {instancias_clase[column].max()}")
            print(f"Mínimo: {instancias_clase[column].min()}")
            print(f"Promedio: {instancias_clase[column].mean():.2f}")
            print(f"Desviación Estándar: {instancias_clase[column].std():.2f}")

            if instancias_clase[column].dtype == 'object':
                # Si es categórico
                print(f"Categorías: {instancias_clase[column].unique()}")

    # Descripción de la etiqueta de salida Y
    print(f"\nEtiqueta Y:")
    print(f"Tipo de dato: {y.dtype}")
    print(f"Categorías: {y.unique()}")
```

3. En caso de ser necesario hacer un preprocesamiento a la base de datos, describirlo

Realiza un preprocesamiento básico del conjunto de datos, incluyendo la imputación de datos faltantes para atributos numéricos, la codificación de variables categóricas y el escalado de características. Luego, divide el conjunto de datos en conjuntos de entrenamiento y prueba.

```
################################################

Punto 3:
```

```python
# En caso de ser necesario hacer un preprocesamiento a la base de datos, describirlo
def punto3(df):
    # Manejo de datos faltantes
    df_imputed = df.copy()
    for column in df_imputed.columns:
        if df_imputed[column].dtype == np.float64:
            # Calcular la media de la columna
            mean_value = df_imputed[column].mean()
            # Reemplazar los valores faltantes con la media
            df_imputed[column] = df_imputed[column].fillna(mean_value)

    # Codificación de variables categóricas (si es necesario)
    df_encoded = df_imputed.copy()
    for column in df_encoded.columns:
        if df_encoded[column].dtype == 'object':
            # Mapear valores únicos a números
            unique_values = df_encoded[column].unique()
            mapping = {value: index for index, value in enumerate(unique_values)}
            df_encoded[column] = df_encoded[column].map(mapping)

    # Escalado de características (si es necesario)
    df_scaled = df_encoded.copy()
    for column in df_scaled.columns[:-1]:  # Excluyendo la columna de etiquetas
        # Calcular la media y la desviación estándar de la columna
        mean_value = df_scaled[column].mean()
        std_dev = df_scaled[column].std()
        # Estandarizar la columna
        df_scaled[column] = (df_scaled[column] - mean_value) / std_dev

    # Dividir los datos en conjuntos de entrenamiento y prueba
    np.random.seed(42)  # Establecer la semilla para reproducibilidad
    mask = np.random.rand(len(df_scaled)) < 0.8
    train_data = df_scaled[mask]
    test_data = df_scaled[~mask]
    X_train = train_data.iloc[:, :-1].to_numpy()
    y_train = train_data.iloc[:, -1].to_numpy()
    X_test = test_data.iloc[:, :-1].to_numpy()
    y_test = test_data.iloc[:, -1].to_numpy()
    return X_train, X_test, y_train, y_test
```

4. Obtener el porcentaje de eficiencia y error utilizando el clasificador de minima distancia y cada uno de los metodos de validacion:
    a. Entrenamiento y prueba
    b. K fold cross validation
    c. Bootstrap

Implementa un clasificador de mínima distancia utilizando el conjunto de datos de entrenamiento y evalúa su rendimiento en el conjunto de prueba. También realiza validación cruzada K-fold y bootstrap para evaluar la robustez del clasificador.

```
##############################################

Punto 4:

   Resultados Clasificador de Mínima Distancia:
    Métrica utilizada: euclidean_distance

   a. Entrenamiento y Prueba:
        Precisión: 80.65%
        Tasa de Error: 19.35%

   b. K-fold Cross Validation:
        Precisión Promedio: 88.89%
        Tasa de Error Promedio: 11.11%

   c. Bootstrap:
        Precisión Promedio: 80.65%
        Tasa de Error Promedio: 19.35%
```

```python
# Punto 4: Clasificador de Mínima Distancia
def punto4(X_train, X_test, y_train, y_test, distance_func=euclidean_distance):
    # a. Entrenamiento y Prueba
    predictions_train_test = min_distance_classifier(X_train, y_train, X_test, distance_func)
    accuracy_train_test = calculate_accuracy(y_test, predictions_train_test)
    error_train_test = calculate_error(y_test, predictions_train_test)

    # b. K-fold Cross Validation (por ejemplo, con K=5)
    k_fold = 30
    fold_size = len(X_train) // k_fold
    accuracies_cv = []
    errors_cv = []

    for i in range(k_fold):
        start_idx = i * fold_size
        end_idx = (i + 1) * fold_size

        # Conjunto de prueba actual
        cv_X_test = X_train[start_idx:end_idx]
        cv_y_test = y_train[start_idx:end_idx]

        # Conjunto de entrenamiento actual
        cv_X_train = np.concatenate([X_train[:start_idx], X_train[end_idx:]])
        cv_y_train = np.concatenate([y_train[:start_idx], y_train[end_idx:]])

        # Clasificación
        predictions_cv = min_distance_classifier(cv_X_train, cv_y_train, cv_X_test, distance_func)

        # Métricas
        accuracy_cv = calculate_accuracy(cv_y_test, predictions_cv)
        error_cv = calculate_error(cv_y_test, predictions_cv)

        accuracies_cv.append(accuracy_cv)
        errors_cv.append(error_cv)

    # c. Bootstrap
    num_bootstrap_samples = 100
    accuracies_bootstrap = []
    errors_bootstrap = []
```

```python
# c. Bootstrap
num_bootstrap_samples = 100
accuracies_bootstrap = []
errors_bootstrap = []

for _ in range(num_bootstrap_samples):
    # Muestreo bootstrap
    bootstrap_indices = np.random.choice(len(X_train), len(X_train), replace=True)
    bootstrap_X_train = X_train[bootstrap_indices]
    bootstrap_y_train = y_train[bootstrap_indices]

    # Clasificación
    predictions_bootstrap = min_distance_classifier(bootstrap_X_train, bootstrap_y_train, X_test, distance_func)

    # Métricas
    accuracy_bootstrap = calculate_accuracy(y_test, predictions_bootstrap)
    error_bootstrap = calculate_error(y_test, predictions_bootstrap)

    accuracies_bootstrap.append(accuracy_bootstrap)
    errors_bootstrap.append(error_bootstrap)

# Resultados para el clasificador de mínima distancia
print("\n   Resultados Clasificador de Mínima Distancia:")
print(f"    Métrica utilizada: {distance_func.__name__}")
print("\n   a. Entrenamiento y Prueba:")
print(f"        Precisión: {accuracy_train_test:.2%}")
print(f"        Tasa de Error: {error_train_test:.2%}")

print("\n   b. K-fold Cross Validation:")
print(f"        Precisión Promedio: {np.mean(accuracies_cv):.2%}")
print(f"        Tasa de Error Promedio: {np.mean(errors_cv):.2%}")

print("\n   c. Bootstrap:")
print(f"        Precisión Promedio: {np.mean(accuracies_bootstrap):.2%}")
print(f"        Tasa de Error Promedio: {np.mean(errors_bootstrap):.2%}")
```

5. Obtener el porcentaje de eficiencia y error utilizando el clasificador K NN y cada uno de los metodos de validacion:
    a. Entrenamiento y prueba
    b. K fold cross validation
    c. Bootstrap

Implementa un clasificador KNN (k vecinos más cercanos) utilizando el conjunto de datos de entrenamiento y evalúa su rendimiento en el conjunto de prueba. También realiza validación cruzada K-fold y bootstrap para evaluar la robustez del clasificador.

```
################################################

Punto 5:

   Resultados Clasificador KNN:
    Métrica utilizada: euclidean_distance

    a. Entrenamiento y Prueba:
        Precisión: 96.77%
        Tasa de Error: 3.23%

   b. K-fold Cross Validation:
        Precisión Promedio: 96.00%
        Tasa de Error Promedio: 4.00%

   c. Bootstrap:
        Precisión Promedio: 93.90%
        Tasa de Error Promedio: 6.10%
```

```python
# Punto 5: Clasificador KNN
def punto5(X_train, X_test, y_train, y_test,distance_func=euclidean_distance):

    # a. Entrenamiento y Prueba

    k_fold = 20
    fold_size = len(X_train) // k_fold

    k_value = 10  # Puedes ajustar este valor según tu elección
    predictions_knn_train_test = knn_classifier(X_train, y_train, X_test, k_value, distance_func)
    accuracy_knn_train_test = calculate_accuracy(y_test, predictions_knn_train_test)
    error_knn_train_test = calculate_error(y_test, predictions_knn_train_test)

    # b. K-fold Cross Validation (usando el mismo valor de k)
    accuracies_knn_cv = []
    errors_knn_cv = []

    for i in range(k_fold):
        start_idx = i * fold_size
        end_idx = (i + 1) * fold_size

        # Conjunto de prueba actual
        cv_X_test = X_train[start_idx:end_idx]
        cv_y_test = y_train[start_idx:end_idx]

        # Conjunto de entrenamiento actual
        cv_X_train = np.concatenate([X_train[:start_idx], X_train[end_idx:]])
        cv_y_train = np.concatenate([y_train[:start_idx], y_train[end_idx:]])

        # Clasificación
        predictions_knn_cv = knn_classifier(cv_X_train, cv_y_train, cv_X_test, k_value, distance_func)

        # Métricas
        accuracy_knn_cv = calculate_accuracy(cv_y_test, predictions_knn_cv)
        error_knn_cv = calculate_error(cv_y_test, predictions_knn_cv)

        accuracies_knn_cv.append(accuracy_knn_cv)
        errors_knn_cv.append(error_knn_cv)
```

```python
# c. Bootstrap (usando el mismo valor de k)
accuracies_knn_bootstrap = []
errors_knn_bootstrap = []

num_bootstrap_samples = 100

for _ in range(num_bootstrap_samples):
    # Muestreo bootstrap
    bootstrap_indices = np.random.choice(len(X_train), len(X_train), replace=True)
    bootstrap_X_train = X_train[bootstrap_indices]
    bootstrap_y_train = y_train[bootstrap_indices]

    # Clasificación
    predictions_knn_bootstrap = knn_classifier(bootstrap_X_train, bootstrap_y_train, X_test, k_value, distance_func)

    # Métricas
    accuracy_knn_bootstrap = calculate_accuracy(y_test, predictions_knn_bootstrap)
    error_knn_bootstrap = calculate_error(y_test, predictions_knn_bootstrap)

    accuracies_knn_bootstrap.append(accuracy_knn_bootstrap)
    errors_knn_bootstrap.append(error_knn_bootstrap)

# Resultados para el clasificador KNN
print("\n   Resultados Clasificador KNN:")
print(f"    Métrica utilizada: {distance_func.__name__}")
print("\n    a. Entrenamiento y Prueba:")
print(f"        Precisión: {accuracy_knn_train_test:.2%}")
print(f"        Tasa de Error: {error_knn_train_test:.2%}")

print("\n   b. K-fold Cross Validation:")
print(f"        Precisión Promedio: {np.mean(accuracies_knn_cv):.2%}")
print(f"        Tasa de Error Promedio: {np.mean(errors_knn_cv):.2%}")

print("\n   c. Bootstrap:")
print(f"        Precisión Promedio: {np.mean(accuracies_knn_bootstrap):.2%}")
print(f"        Tasa de Error Promedio: {np.mean(errors_knn_bootstrap):.2%}")
```

6. Elegir dos de los atributos utilizando algun criterio o razon que debe describir por la que se cree que su eliminacion puede mejorar la eficiencia y posteriormente:
    a. Eliminar uno de los dos atributos elegidos y corrobar la teoria mediante los 3 metodos de validacion vistos
    b. Eliminar el otro de los atributos elegidos y corrobar la teoria mediante los 3 metodos de validacion vistos
    c. Eliminar los dos atributos elegidos y corrobar la teoria mediante los 3 metodos de validacion vistos

Permite al usuario eliminar uno o dos atributos del conjunto de datos para evaluar cómo afecta la eficiencia de los clasificadores de mínima distancia y KNN.

```
################################################################################

Punto 6:

 DataFrame Original:
     LargoSepalo  AnchoSepalo  LargoPetalo  AnchoPetalo          Clase
0            5.1          3.5          1.4          0.2     Iris-setosa
1            4.9          3.0          1.4          0.2     Iris-setosa
2            4.7          3.2          1.3          0.2     Iris-setosa
3            4.6          3.1          1.5          0.2     Iris-setosa
4            5.0          3.6          1.4          0.2     Iris-setosa
..           ...          ...          ...          ...             ...
145          6.7          3.0          5.2          2.3  Iris-virginica
146          6.3          2.5          5.0          1.9  Iris-virginica
147          6.5          3.0          5.2          2.0  Iris-virginica
148          6.2          3.4          5.4          2.3  Iris-virginica
149          5.9          3.0          5.1          1.8  Iris-virginica

[150 rows x 5 columns]

   Tienes los siguientes atributos:
    Atributos 0: LargoSepalo
    Atributos 1: AnchoSepalo
    Atributos 2: LargoPetalo
    Atributos 3: AnchoPetalo

    Ingresa el numero del primer atributo a eliminar:0

    Ingresa el numero del segundo atributo a eliminar:1

 a.Eliminar el primer Atributo
     AnchoSepalo  LargoPetalo  AnchoPetalo          Clase
0            3.5          1.4          0.2     Iris-setosa
1            3.0          1.4          0.2     Iris-setosa
2            3.2          1.3          0.2     Iris-setosa
3            3.1          1.5          0.2     Iris-setosa
4            3.6          1.4          0.2     Iris-setosa
..           ...          ...          ...             ...
145          3.0          5.2          2.3  Iris-virginica
146          2.5          5.0          1.9  Iris-virginica
147          3.0          5.2          2.0  Iris-virginica
148          3.4          5.4          2.3  Iris-virginica
```

```
Resultados Clasificador de Mínima Distancia:
 Métrica utilizada: euclidean_distance

a. Entrenamiento y Prueba:
     Precisión: 87.10%
     Tasa de Error: 12.90%

b. K-fold Cross Validation:
     Precisión Promedio: 91.11%
     Tasa de Error Promedio: 8.89%

c. Bootstrap:
     Precisión Promedio: 90.45%
     Tasa de Error Promedio: 9.55%

Resultados Clasificador KNN:
 Métrica utilizada: euclidean_distance

 a. Entrenamiento y Prueba:
     Precisión: 93.55%
     Tasa de Error: 6.45%

b. K-fold Cross Validation:
     Precisión Promedio: 96.00%
     Tasa de Error Promedio: 4.00%

c. Bootstrap:
     Precisión Promedio: 95.42%
     Tasa de Error Promedio: 4.58%
```

```
b.Eliminar el segundo Atributo
     LargoSepalo  LargoPetalo  AnchoPetalo           Clase
0            5.1          1.4          0.2     Iris-setosa
1            4.9          1.4          0.2     Iris-setosa
2            4.7          1.3          0.2     Iris-setosa
3            4.6          1.5          0.2     Iris-setosa
4            5.0          1.4          0.2     Iris-setosa
..           ...          ...          ...             ...
145          6.7          5.2          2.3  Iris-virginica
146          6.3          5.0          1.9  Iris-virginica
147          6.5          5.2          2.0  Iris-virginica
148          6.2          5.4          2.3  Iris-virginica
149          5.9          5.1          1.8  Iris-virginica

[150 rows x 4 columns]
```

```
Resultados Clasificador de Mínima Distancia:
 Métrica utilizada: euclidean_distance

a. Entrenamiento y Prueba:
     Precisión: 77.42%
     Tasa de Error: 22.58%

b. K-fold Cross Validation:
     Precisión Promedio: 93.33%
     Tasa de Error Promedio: 6.67%

c. Bootstrap:
     Precisión Promedio: 80.81%
     Tasa de Error Promedio: 19.19%

Resultados Clasificador KNN:
 Métrica utilizada: euclidean_distance

 a. Entrenamiento y Prueba:
     Precisión: 90.32%
     Tasa de Error: 9.68%

b. K-fold Cross Validation:
     Precisión Promedio: 96.00%
     Tasa de Error Promedio: 4.00%

c. Bootstrap:
     Precisión Promedio: 91.48%
     Tasa de Error Promedio: 8.52%
```

```
c.Eliminar los dos atributos
     LargoPetalo  AnchoPetalo           Clase
0            1.4          0.2     Iris-setosa
1            1.4          0.2     Iris-setosa
2            1.3          0.2     Iris-setosa
3            1.5          0.2     Iris-setosa
4            1.4          0.2     Iris-setosa
..           ...          ...             ...
145          5.2          2.3  Iris-virginica
146          5.0          1.9  Iris-virginica
147          5.2          2.0  Iris-virginica
148          5.4          2.3  Iris-virginica
149          5.1          1.8  Iris-virginica

[150 rows x 3 columns]
```

```
Resultados Clasificador de Mínima Distancia:
 Métrica utilizada: euclidean_distance

a. Entrenamiento y Prueba:
     Precisión: 96.77%
     Tasa de Error: 3.23%

b. K-fold Cross Validation:
     Precisión Promedio: 96.67%
     Tasa de Error Promedio: 3.33%

c. Bootstrap:
     Precisión Promedio: 97.52%
     Tasa de Error Promedio: 2.48%

Resultados Clasificador KNN:
 Métrica utilizada: euclidean_distance

 a. Entrenamiento y Prueba:
     Precisión: 96.77%
     Tasa de Error: 3.23%

b. K-fold Cross Validation:
     Precisión Promedio: 96.00%
     Tasa de Error Promedio: 4.00%

c. Bootstrap:
     Precisión Promedio: 96.71%
     Tasa de Error Promedio: 3.29%
```

```python
# Eliminacion de 2 atributos para mejorar la eficiencia
def punto6(df,atributos, clase):
    print("\n DataFrame Original:")
    print(df)

    print("\n   Tienes los siguientes atributos:")
    # Asigna un número a cada columna
    numeros_columnas = {i: columna for i, columna in enumerate(atributos.columns)}

    # Imprime la relación entre el número y el nombre de la columna
    for numero, columna in numeros_columnas.items():
        print(f'    Atributos {numero}: {columna}')

    colAeliminar1 = int(input("\n    Ingresa el numero del primer atributo a eliminar:"))
    colAeliminar2 = int(input("\n    Ingresa el numero del segundo atributo a eliminar:"))


    print("\n a.Eliminar el primer Atributo")
    # Verifica que el índice sea válido
    if colAeliminar1 < len(atributos.columns):
        # Utiliza el método drop para eliminar la columna por su índice
        newdf1 = df.drop(df.columns[colAeliminar1], axis=1)
        # Muestra el DataFrame después de eliminar la columna
        print(newdf1)
    else:
        print(f"El índice {colAeliminar1} no es válido para las columnas del DataFrame.")

    X_train, X_test, y_train, y_test = punto3(newdf1)
    # Punto 4
    punto4(X_train, X_test, y_train, y_test, distance_func=euclidean_distance)
    # Punto 5
    punto5(X_train, X_test, y_train, y_test,distance_func=euclidean_distance)


    print("\nb.Eliminar el segundo Atributo")

    # Verifica que el índice sea válido
    if colAeliminar2 < len(atributos.columns):
        # Utiliza el método drop para eliminar la columna por su índice
```

```
    newdf2 = df.drop(df.columns[colAeliminar2], axis=1)
    # Muestra el DataFrame después de eliminar la columna
    print(newdf2)
else:
    print(f"El índice {colAeliminar2} no es válido para las columnas del DataFrame.")


X_train, X_test, y_train, y_test = punto3(newdf2)
# Punto 4
punto4(X_train, X_test, y_train, y_test, distance_func=euclidean_distance)
# Punto 5
punto5(X_train, X_test, y_train, y_test,distance_func=euclidean_distance)

print("\nc.Eliminar los dos atributos")
# Verifica que el índice sea válido
if colAeliminar1 < len(atributos.columns) and colAeliminar2 < len(atributos.columns):
    # Utiliza el método drop para eliminar la columna por su índice
    newdf = df.drop(df.columns[colAeliminar1], axis=1)
    newdf3 = newdf.drop(df.columns[colAeliminar2], axis=1)
    # Muestra el DataFrame después de eliminar la columna
    print(newdf3)
else:
    print(f"El índice {colAeliminar1}  o el {colAeliminar2} no son válidos para las columnas del DataFrame.")


X_train, X_test, y_train, y_test = punto3(newdf3)
# Punto 4
punto4(X_train, X_test, y_train, y_test, distance_func=euclidean_distance)
# Punto 5
punto5(X_train, X_test, y_train, y_test,distance_func=euclidean_distance)
```

7. Eliminar uno o mas de las muestras utilizando algun criterio que debe describir, puede ser utilizando la totalidad de los atributos o una version reducida de estos. Buscando mejorar la eficiencia, corrobar la teoria mediante los 3 metodos de validacion vistos

Permite al usuario crear un nuevo conjunto de datos seleccionando un subconjunto específico de muestras (filas) y atributos (columnas) del conjunto de datos original.

```
################################################################################

Punto 7:
(150, 4)
Tenemos la siguiente cantidad de muestras 150
Use la notacion de segmentación, Ej. [X:N], [:N], [X:], , etc
Habilite los atributos (filas) que desea tener en su vector de salida.
0:149
Conjunto de datos creado:

[[5.1 3.5 1.4 0.2 'Iris-setosa']
 [4.9 3.0 1.4 0.2 'Iris-setosa']
 [4.7 3.2 1.3 0.2 'Iris-setosa']
 [4.6 3.1 1.5 0.2 'Iris-setosa']
 [5.0 3.6 1.4 0.2 'Iris-setosa']
 [5.4 3.9 1.7 0.4 'Iris-setosa']
 [4.6 3.4 1.4 0.3 'Iris-setosa']
 [5.0 3.4 1.5 0.2 'Iris-setosa']
 [4.4 2.9 1.4 0.2 'Iris-setosa']
 [4.9 3.1 1.5 0.1 'Iris-setosa']
 [5.4 3.7 1.5 0.2 'Iris-setosa']
 [4.8 3.4 1.6 0.2 'Iris-setosa']
 [4.8 3.0 1.4 0.1 'Iris-setosa']
 [4.3 3.0 1.1 0.1 'Iris-setosa']
 [5.8 4.0 1.2 0.2 'Iris-setosa']
 [5.7 4.4 1.5 0.4 'Iris-setosa']
 [5.4 3.9 1.3 0.4 'Iris-setosa']
 [5.1 3.5 1.4 0.3 'Iris-setosa']
 [5.7 3.8 1.7 0.3 'Iris-setosa']
 [5.1 3.8 1.5 0.3 'Iris-setosa']
 [5.4 3.4 1.7 0.2 'Iris-setosa']
 [5.1 3.7 1.5 0.4 'Iris-setosa']
 [4.6 3.6 1.0 0.2 'Iris-setosa']
 [5.1 3.3 1.7 0.5 'Iris-setosa']
 [4.8 3.4 1.9 0.2 'Iris-setosa']
 [5.0 3.0 1.6 0.2 'Iris-setosa']
 [5.0 3.4 1.6 0.4 'Iris-setosa']
 [5.2 3.5 1.5 0.2 'Iris-setosa']
 [5.2 3.4 1.4 0.2 'Iris-setosa']
 [4.7 3.2 1.6 0.2 'Iris-setosa']
 [4.8 3.1 1.6 0.2 'Iris-setosa']
 [5.4 3.4 1.5 0.4 'Iris-setosa']
 [5.2 4.1 1.5 0.1 'Iris-setosa']
```

```python
# Eliminacion de uno o mas muestras para mejorar la eficiencia
def punto7(df,atributos, y, subsets_por_clase):
    new_x = np.concatenate(subsets_por_clase)
    print(new_x.shape)

    # Solicita al usuario los tamaños de salida
    print(f"Tenemos la siguiente cantidad de muestras {new_x.shape[0]}")
    print(f"Use la notacion de segmentación, Ej. [X:N], [:N], [X:], , etc")
    output_size = (input("Habilite los atributos (filas) que desea tener en su vector de salida.\n"))

    filas_seleccionadas = output_size.split(":")
    filas_seleccionadas = [int(splits) for splits in filas_seleccionadas]

    output_dataMod = y[filas_seleccionadas[0]: filas_seleccionadas[1]+1]
    new_x= new_x[filas_seleccionadas[0]: filas_seleccionadas[1]+1]
    #print(new_xMod)



    print("Conjunto de datos creado: \n")
    nuevodf = np.column_stack((new_x, output_dataMod))
    print(nuevodf)
```

CODIGO

A continuacion esta todo el código:

```python
import numpy as np
import pandas as pd

# Describir cada uno de los atributos al momento
def punto1(dataset_path):

    # Cargar el conjunto de datos usando pandas, indicando que la primera columna es el
índice
    df = pd.read_csv(dataset_path)

    # Obtener los atributos y la etiqueta
    X = df.iloc[:, :-1]
    y = df.iloc[:, -1]

    # Descripción de los atributos del vector de entrada X
    for idx, column in enumerate(X.columns):
        print(f"\nAtributo {idx + 1}: {column}")
        print(f"Tipo de dato: {X[column].dtype}")

        if X[column].dtype == 'object':
            # Si es categórico
            print(f"Categorías: {X[column].unique()}")
```

```
        else:
            # Si es numérico
            print(f"Mínimo: {X[column].min()}")
            print(f"Máximo: {X[column].max()}")
            print(f"Promedio: {X[column].mean():.2f}")
            print(f"Desviación Estándar: {X[column].std():.2f}")

    # Descripción de la etiqueta de salida Y
    print(f"\nEtiqueta Y:")
    print(f"Tipo de dato: {y.dtype}")
    print(f"Categorías: {y.unique()}")

    return df, X, y

# Definir los atributos del vector de entrada X y de salida(clase) Y
def punto2(df):

    # Obtener los atributos y la etiqueta
    X = df.iloc[:, :-1]
    y = df.iloc[:, -1]

    # Por cada clase en Y
    clases = y.unique()
    for clase in clases:
        print(f"\nEstadísticas para la Clase {clase}:")
        # Filtrar las instancias correspondientes a la clase actual
        instancias_clase = df[df.iloc[:, -1] == clase].iloc[:, :-1]

        # Obtener las estadísticas
        for idx, column in enumerate(instancias_clase.columns):
            print(f"\nAtributo {idx + 1}: {column}")
            print(f"Máximo: {instancias_clase[column].max()}")
            print(f"Mínimo: {instancias_clase[column].min()}")
            print(f"Promedio: {instancias_clase[column].mean():.2f}")
            print(f"Desviación Estándar: {instancias_clase[column].std():.2f}")

            if instancias_clase[column].dtype == 'object':
                # Si es categórico
                print(f"Categorías: {instancias_clase[column].unique()}")

    # Descripción de la etiqueta de salida Y
    print(f"\nEtiqueta Y:")
    print(f"Tipo de dato: {y.dtype}")
    print(f"Categorías: {y.unique()}")

# En caso de ser necesario hacer un preprocesamiento a la base de datos, describirlo
def punto3(df):
    # Manejo de datos faltantes
    df_imputed = df.copy()
    for column in df_imputed.columns:
```

```
        if df_imputed[column].dtype == np.float64:
            # Calcular la media de la columna
            mean_value = df_imputed[column].mean()
            # Reemplazar los valores faltantes con la media
            df_imputed[column] = df_imputed[column].fillna(mean_value)

    # Codificación de variables categóricas (si es necesario)
    df_encoded = df_imputed.copy()
    for column in df_encoded.columns:
        if df_encoded[column].dtype == 'object':
            # Mapear valores únicos a números
            unique_values = df_encoded[column].unique()
            mapping = {value: index for index, value in enumerate(unique_values)}
            df_encoded[column] = df_encoded[column].map(mapping)

    # Escalado de características (si es necesario)
    df_scaled = df_encoded.copy()
    for column in df_scaled.columns[:-1]:  # Excluyendo la columna de etiquetas
        # Calcular la media y la desviación estándar de la columna
        mean_value = df_scaled[column].mean()
        std_dev = df_scaled[column].std()
        # Estandarizar la columna
        df_scaled[column] = (df_scaled[column] - mean_value) / std_dev

    # Dividir los datos en conjuntos de entrenamiento y prueba
    np.random.seed(42)  # Establecer la semilla para reproducibilidad
    mask = np.random.rand(len(df_scaled)) < 0.8
    train_data = df_scaled[mask]
    test_data = df_scaled[~mask]
    X_train = train_data.iloc[:, :-1].to_numpy()
    y_train = train_data.iloc[:, -1].to_numpy()
    X_test = test_data.iloc[:, :-1].to_numpy()
    y_test = test_data.iloc[:, -1].to_numpy()
    return X_train, X_test, y_train, y_test

def euclidean_distance(x1, x2):
    # Calcular la distancia euclidiana entre dos puntos
    return np.linalg.norm(x1 - x2)

def manhattan_distance(x1, x2):
    # Calcular la distancia de Manhattan entre dos puntos
    return np.sum(np.abs(x1 - x2))

# def min_distance_classifier(train_X, train_y, test_X, distance_func=euclidean_distance):
#     # Implementación del clasificador de mínima distancia
#     predictions = []
#     for sample in test_X:
#         # Calcular la distancia con cada instancia de entrenamiento
#         distances = [distance_func(sample, train_sample) for train_sample in train_X]
#         # Obtener la clase de la instancia más cercana
```

```
#         predicted_class = train_y[np.argmin(distances)]
#         predictions.append(predicted_class)
#     return np.array(predictions)

def min_distance_classifier(train_X, train_y, test_X, distance_func=euclidean_distance):
    # Calcula el valor promedio para cada atributo en el conjunto de entrenamiento para cada
clase
    class_averages = {cls: np.mean(train_X[train_y == cls], axis=0) for cls in set(train_y)}

    # Clasificación
    predictions = []
    for sample in test_X:
        # Calcula la distancia con los valores promedio de cada clase
        distances = [distance_func(sample, class_averages[cls]) for cls in class_averages]
        # Asigna la clase cuyo valor promedio está más cerca
        predicted_class = list(class_averages.keys())[np.argmin(distances)]
        predictions.append(predicted_class)

    return np.array(predictions)

def knn_classifier(train_X, train_y, test_X, k, distance_func=euclidean_distance):
    # Implementación del clasificador KNN
    predictions = []
    for sample in test_X:
        # Calcular la distancia con cada instancia de entrenamiento
        distances = [distance_func(sample, train_sample) for train_sample in train_X]
        # Obtener las k instancias más cercanas
        nearest_neighbors_indices = np.argsort(distances)[:k]
        # Obtener las clases de las k instancias más cercanas
        nearest_neighbors_classes = train_y[nearest_neighbors_indices]
        # Obtener la clase más común entre las k instancias más cercanas
        predicted_class = np.bincount(nearest_neighbors_classes).argmax()
        predictions.append(predicted_class)
    return np.array(predictions)

def calculate_accuracy(y_true, y_pred):
    # Calcular la precisión del clasificador
    correct_predictions = np.sum(y_true == y_pred)
    total_samples = len(y_true)
    accuracy = correct_predictions / total_samples
    return accuracy

def calculate_error(y_true, y_pred):
    # Calcular la tasa de error del clasificador
    incorrect_predictions = np.sum(y_true != y_pred)
    total_samples = len(y_true)
    error_rate = incorrect_predictions / total_samples
    return error_rate

# Punto 4: Clasificador de Mínima Distancia
```

```python
def punto4(X_train, X_test, y_train, y_test, distance_func=euclidean_distance):
     # a. Entrenamiento y Prueba
    predictions_train_test = min_distance_classifier(X_train, y_train, X_test, distance_func)
    accuracy_train_test = calculate_accuracy(y_test, predictions_train_test)
    error_train_test = calculate_error(y_test, predictions_train_test)

    # b. K-fold Cross Validation (por ejemplo, con K=5)
    k_fold = 30
    fold_size = len(X_train) // k_fold
    accuracies_cv = []
    errors_cv = []

    for i in range(k_fold):
        start_idx = i * fold_size
        end_idx = (i + 1) * fold_size

        # Conjunto de prueba actual
        cv_X_test = X_train[start_idx:end_idx]
        cv_y_test = y_train[start_idx:end_idx]

        # Conjunto de entrenamiento actual
        cv_X_train = np.concatenate([X_train[:start_idx], X_train[end_idx:]])
        cv_y_train = np.concatenate([y_train[:start_idx], y_train[end_idx:]])

        # Clasificación
        predictions_cv = min_distance_classifier(cv_X_train, cv_y_train, cv_X_test,
distance_func)

        # Métricas
        accuracy_cv = calculate_accuracy(cv_y_test, predictions_cv)
        error_cv = calculate_error(cv_y_test, predictions_cv)

        accuracies_cv.append(accuracy_cv)
        errors_cv.append(error_cv)

    # c. Bootstrap
    num_bootstrap_samples = 100
    accuracies_bootstrap = []
    errors_bootstrap = []

    for _ in range(num_bootstrap_samples):
        # Muestreo bootstrap
        bootstrap_indices = np.random.choice(len(X_train), len(X_train), replace=True)
        bootstrap_X_train = X_train[bootstrap_indices]
        bootstrap_y_train = y_train[bootstrap_indices]

        # Clasificación
        predictions_bootstrap = min_distance_classifier(bootstrap_X_train, bootstrap_y_train,
X_test, distance_func)
```

```
        # Métricas
        accuracy_bootstrap = calculate_accuracy(y_test, predictions_bootstrap)
        error_bootstrap = calculate_error(y_test, predictions_bootstrap)

        accuracies_bootstrap.append(accuracy_bootstrap)
        errors_bootstrap.append(error_bootstrap)

    # Resultados para el clasificador de mínima distancia
    print("\n   Resultados Clasificador de Mínima Distancia:")
    print(f"    Métrica utilizada: {distance_func.__name__}")
    print("\n   a. Entrenamiento y Prueba:")
    print(f"        Precisión: {accuracy_train_test:.2%}")
    print(f"        Tasa de Error: {error_train_test:.2%}")

    print("\n   b. K-fold Cross Validation:")
    print(f"        Precisión Promedio: {np.mean(accuracies_cv):.2%}")
    print(f"        Tasa de Error Promedio: {np.mean(errors_cv):.2%}")

    print("\n   c. Bootstrap:")
    print(f"        Precisión Promedio: {np.mean(accuracies_bootstrap):.2%}")
    print(f"        Tasa de Error Promedio: {np.mean(errors_bootstrap):.2%}")

# Punto 5: Clasificador KNN
def punto5(X_train, X_test, y_train, y_test,distance_func=euclidean_distance):

    # a. Entrenamiento y Prueba

    k_fold = 20
    fold_size = len(X_train) // k_fold

    k_value = 10  # Puedes ajustar este valor según tu elección
    predictions_knn_train_test = knn_classifier(X_train, y_train, X_test, k_value,
distance_func)
    accuracy_knn_train_test = calculate_accuracy(y_test, predictions_knn_train_test)
    error_knn_train_test = calculate_error(y_test, predictions_knn_train_test)

    # b. K-fold Cross Validation (usando el mismo valor de k)
    accuracies_knn_cv = []
    errors_knn_cv = []

    for i in range(k_fold):
        start_idx = i * fold_size
        end_idx = (i + 1) * fold_size

        # Conjunto de prueba actual
        cv_X_test = X_train[start_idx:end_idx]
        cv_y_test = y_train[start_idx:end_idx]

        # Conjunto de entrenamiento actual
        cv_X_train = np.concatenate([X_train[:start_idx], X_train[end_idx:]])
```

```
        cv_y_train = np.concatenate([y_train[:start_idx], y_train[end_idx:]])

        # Clasificación
        predictions_knn_cv = knn_classifier(cv_X_train, cv_y_train, cv_X_test, k_value,
distance_func)

        # Métricas
        accuracy_knn_cv = calculate_accuracy(cv_y_test, predictions_knn_cv)
        error_knn_cv = calculate_error(cv_y_test, predictions_knn_cv)

        accuracies_knn_cv.append(accuracy_knn_cv)
        errors_knn_cv.append(error_knn_cv)

    # c. Bootstrap (usando el mismo valor de k)
    accuracies_knn_bootstrap = []
    errors_knn_bootstrap = []

    num_bootstrap_samples = 100

    for _ in range(num_bootstrap_samples):
        # Muestreo bootstrap
        bootstrap_indices = np.random.choice(len(X_train), len(X_train), replace=True)
        bootstrap_X_train = X_train[bootstrap_indices]
        bootstrap_y_train = y_train[bootstrap_indices]

        # Clasificación
        predictions_knn_bootstrap = knn_classifier(bootstrap_X_train, bootstrap_y_train,
X_test, k_value, distance_func)

        # Métricas
        accuracy_knn_bootstrap = calculate_accuracy(y_test, predictions_knn_bootstrap)
        error_knn_bootstrap = calculate_error(y_test, predictions_knn_bootstrap)

        accuracies_knn_bootstrap.append(accuracy_knn_bootstrap)
        errors_knn_bootstrap.append(error_knn_bootstrap)

    # Resultados para el clasificador KNN
    print("\n   Resultados Clasificador KNN:")
    print(f"    Métrica utilizada: {distance_func.__name__}")
    print("\n    a. Entrenamiento y Prueba:")
    print(f"        Precisión: {accuracy_knn_train_test:.2%}")
    print(f"        Tasa de Error: {error_knn_train_test:.2%}")

    print("\n   b. K-fold Cross Validation:")
    print(f"        Precisión Promedio: {np.mean(accuracies_knn_cv):.2%}")
    print(f"        Tasa de Error Promedio: {np.mean(errors_knn_cv):.2%}")

    print("\n   c. Bootstrap:")
    print(f"        Precisión Promedio: {np.mean(accuracies_knn_bootstrap):.2%}")
    print(f"        Tasa de Error Promedio: {np.mean(errors_knn_bootstrap):.2%}")
```

```python
# Eliminacion de 2 atributos para mejorar la eficiencia
def punto6(df,atributos, clase):
    print("\n DataFrame Original:")
    print(df)

    print("\n   Tienes los siguientes atributos:")
    # Asigna un número a cada columna
    numeros_columnas = {i: columna for i, columna in enumerate(atributos.columns)}

    # Imprime la relación entre el número y el nombre de la columna
    for numero, columna in numeros_columnas.items():
        print(f'    Atributos {numero}: {columna}')

    colAeliminar1 = int(input("\n    Ingresa el numero del primer atributo a eliminar:"))
    colAeliminar2 = int(input("\n    Ingresa el numero del segundo atributo a eliminar:"))


    print("\n a.Eliminar el primer Atributo")
    # Verifica que el índice sea válido
    if colAeliminar1 < len(atributos.columns):
        # Utiliza el método drop para eliminar la columna por su índice
        newdf1 = df.drop(df.columns[colAeliminar1], axis=1)
        # Muestra el DataFrame después de eliminar la columna
        print(newdf1)
    else:
        print(f"El índice {colAeliminar1} no es válido para las columnas del DataFrame.")

    X_train, X_test, y_train, y_test = punto3(newdf1)
    # Punto 4
    punto4(X_train, X_test, y_train, y_test, distance_func=euclidean_distance)
    # Punto 5
    punto5(X_train, X_test, y_train, y_test,distance_func=euclidean_distance)


    print("\nb.Eliminar el segundo Atributo")

    # Verifica que el índice sea válido
    if colAeliminar2 < len(atributos.columns):
        # Utiliza el método drop para eliminar la columna por su índice
        newdf2 = df.drop(df.columns[colAeliminar2], axis=1)
        # Muestra el DataFrame después de eliminar la columna
        print(newdf2)
    else:
        print(f"El índice {colAeliminar2} no es válido para las columnas del DataFrame.")


    X_train, X_test, y_train, y_test = punto3(newdf2)
    # Punto 4
    punto4(X_train, X_test, y_train, y_test, distance_func=euclidean_distance)
```

```
    # Punto 5
    punto5(X_train, X_test, y_train, y_test,distance_func=euclidean_distance)

    print("\nc.Eliminar los dos atributos")
    # Verifica que el índice sea válido
    if colAeliminar1 < len(atributos.columns) and colAeliminar2 < len(atributos.columns):
        # Utiliza el método drop para eliminar la columna por su índice
        newdf = df.drop(df.columns[colAeliminar1], axis=1)
        newdf3 = newdf.drop(df.columns[colAeliminar2], axis=1)
        # Muestra el DataFrame después de eliminar la columna
        print(newdf3)
    else:
        print(f"El índice {colAeliminar1}  o el {colAeliminar2} no son válidos para las
columnas del DataFrame.")


    X_train, X_test, y_train, y_test = punto3(newdf3)
    # Punto 4
    punto4(X_train, X_test, y_train, y_test, distance_func=euclidean_distance)
    # Punto 5
    punto5(X_train, X_test, y_train, y_test,distance_func=euclidean_distance)


# Eliminacion de uno o mas muestras para mejorar la eficiencia
def punto7(df,atributos, y, subsets_por_clase):
    new_x = np.concatenate(subsets_por_clase)
    print(new_x.shape)

    # Solicita al usuario los tamaños de salida
    print(f"Tenemos la siguiente cantidad de muestras {new_x.shape[0]}")
    print(f"Use la notacion de segmentación, Ej. [X:N], [:N], [X:], , etc")
    output_size = (input("Habilite los atributos (filas) que desea tener en su vector de
salida.\n"))

    filas_seleccionadas = output_size.split(":")
    filas_seleccionadas = [int(splits) for splits in filas_seleccionadas]

    output_dataMod = y[filas_seleccionadas[0]: filas_seleccionadas[1]+1]
    new_x= new_x[filas_seleccionadas[0]: filas_seleccionadas[1]+1]
    #print(new_xMod)



    print("Conjunto de datos creado: \n")
    nuevodf = np.column_stack((new_x, output_dataMod))
    print(nuevodf)
    """
    #return nuevodf, new_x, output_dataMod
    column_names = atributos.columns
    column_names = column_names.append('Clase')
```

```
    print(column_names)

    # Extrae los nombres de las columnas de la primera fila
    #column_names = nuevodf[0, :-1].tolist() + ['Clase']

    # Crea un DataFrame utilizando los nombres de las columnas y los datos restantes
    #df = pd.DataFrame(nuevodf[1:], columns=column_names)

    df = pd.DataFrame(nuevodf, columns=column_names)

    X_train, X_test, y_train, y_test = punto3(df)
    # Punto 4
    punto4(X_train, X_test, y_train, y_test, distance_func=euclidean_distance)
    # Punto 5
    punto5(X_train, X_test, y_train, y_test,distance_func=euclidean_distance)
    """
def obtener_clases(x, y):
    ## Obtenemos la clase con base en sus etiquetas [y]
    clases_unicas = np.unique(y)

    ## Fragmentamos las clases en arrays diferentes 1 x clase
    subsets = [x[y == cls] for cls in clases_unicas]
    return clases_unicas, subsets
##########################################################################################
##############
if __name__ == "__main__":

    # Ubicacion DataSet
    dataset_path = 'iris.data'


    # Punto 1
    print("\n#################################################################################")

    print("\nPunto 1:")
    df, atributos, clases= punto1(dataset_path)

    # Punto 2
    print("\n#################################################################################")

    print("\nPunto 2:")
    punto2(df)

    # Punto 3
    print("\n#################################################################################")

    print("\nPunto 3:")
    X_train, X_test, y_train, y_test = punto3(df)

    # Punto 4
```

```
print("\n#################################################################################")

print("\nPunto 4:")
punto4(X_train, X_test, y_train, y_test, distance_func=euclidean_distance)

# Punto 5
print("\n#################################################################################")

print("\nPunto 5:")
punto5(X_train, X_test, y_train, y_test,distance_func=euclidean_distance)

# Punto 6
print("\n#################################################################################")

print("\nPunto 6:")
punto6(df, atributos, clases)

# Punto 7
print("\n#################################################################################")

print("\nPunto 7:")
 # Obtener clases
allclases, subsets_por_clase = obtener_clases(atributos, clases)
punto7(df,atributos, clases, subsets_por_clase)
```