Práctica 5

Néstor Iván Martínez Ostoa
Visualización de la Información - 0605
I.I.M.A.S. - U.N.A.M.

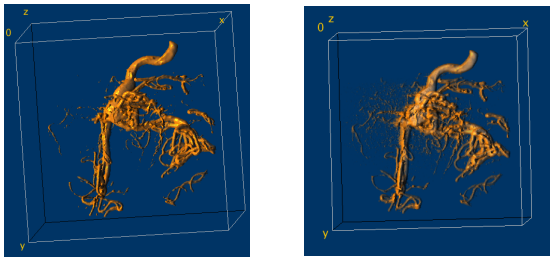
21 de marzo del 2021

---

1. Visualización de Campos Escalares

Para los siguientes conjuntos de datos genere renderizados volumétricos.

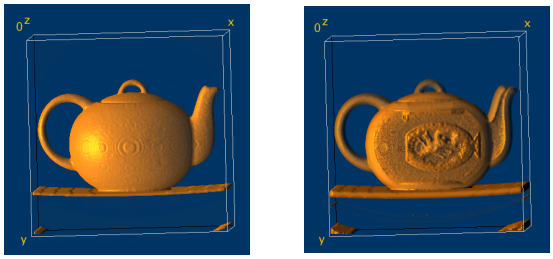
- *aneurism.raw*



(a) Función 1D (b) Función 2D

Figura 1: Comparación entre imágenes utilizando funciones de transferencia 1D y 2D

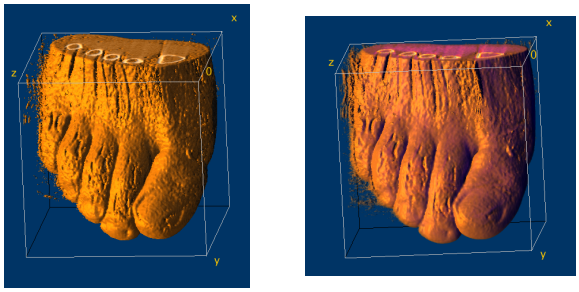
- *BostonTeapoot.raw*



(a) Función 1D (b) Función 2D

Figura 2: Comparación entre imágenes utilizando funciones de transferencia 1D y 2D

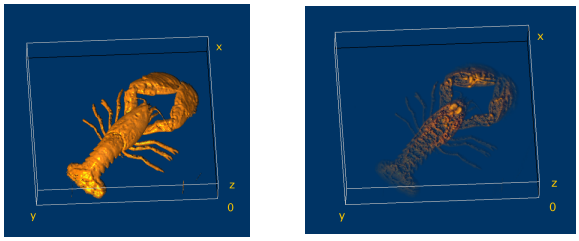
- *foot.raw*



(a) Función 1D (b) Función 2D

Figura 3: Comparación entre imágenes utilizando funciones de transferencia 1D y 2D

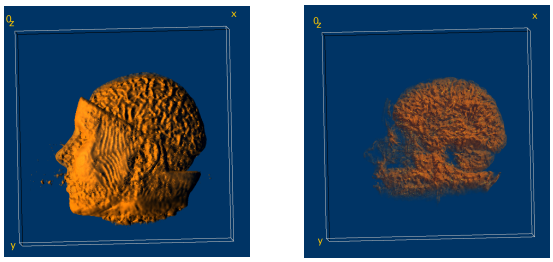
- *lobster.raw*



(a) Función 1D (b) Función 2D

Figura 4: Comparación entre imágenes utilizando funciones de transferencia 1D y 2D

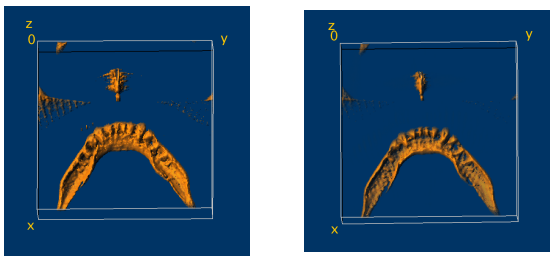
- *mrbrain*



(a) Función 1D (b) Función 2D

Figura 5: Comparación entre imágenes utilizando funciones de transferencia 1D y 2D

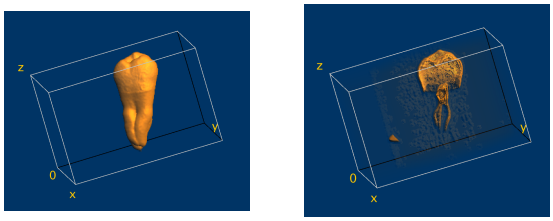
- *skull.raw*



(a) Función 1D (b) Función 2D

Figura 6: Comparación entre imágenes utilizando funciones de transferencia 1D y 2D

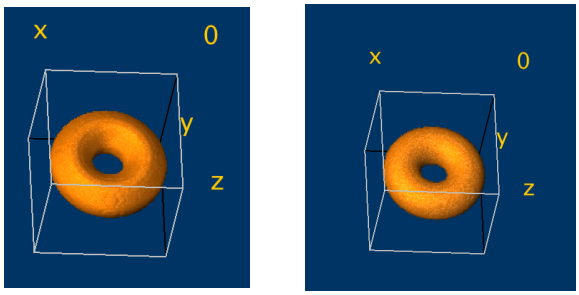
- *tooth.mrc*



(a) Función 1D (b) Función 2D

Figura 7: Comparación entre imágenes utilizando funciones de transferencia 1D y 2D

- *torus.mrc*



(a) Función 1D (b) Función 2D

Figura 8: Comparación entre imágenes utilizando funciones de transferencia 1D y 2D

¿Qué efecto tiene sobre los datos que se despliegan en pantalla el uso de estas funciones de transferencia?

Ambas controlan la opacidad de los vóxeles dentro de una imagen, sin embargo, encontré que la función de transferencia en 2D permite un control mucho más detallado pues dentro de una misma línea de voxeles, permite ocultar la opacidad de alguno de ellos, mientras que la función de 1D o muestra toda la línea o no la muestra.

Describa el proceso que tendría que llevar a cabo para visualizar una estructura específica (por ejemplo, los ventrıículos cerebrales) en los datos de resonancia magnética de la cabeza.

1. Identificar dentro de una imagen las áreas de interés
2. Ocupando los planos sagital, coronal y transversal dentro de *ImageJ*, seleccionar las áreas de interés y notar el punto marcado dentro del área de función de transferencia
3. Dentro del área de transferencia, subrayar el punto marcado (usualmente en azul) para incrementar la opacidad de ese punto dentro de la imagen
4. Repetir estos pasos hasta encontrar la imagen deseada

¿Se podría lograr esto únicamente usando funciones de transferencia? Explique.

El único impedimento que encontré a la hora de emplear funciones de transferencia con el método descrito anteriormente, es que cuando uno selecciona un área en particular, la función de transferencia dentro de la imagen muestra otras áreas que tal vez no sean deseadas. Es decir, incrementar la opacidad de un punto usualmente también incrementa la opacidad de otras áreas dentro de la imagen. Entonces, es complicado únicamente con una función de transferencia lograr mostrar solo un parte de la imagen.

2. Filtrado de datos

1. Haga la lectura del fichero *titanic3.csv*. Según este *dataset*:

   a) ¿Cuántos cuerpos fueron encontrados? ¿Cuántos de estos eran hombres mayores de 40 años?

   Se consideraron dentro del dataset de *titanic.csv* aquellos registros cuyo campo *body* es nulo pues este campo representa un identificador de cuerpo por lo que estoy asumiendo que si un registro sí tiene un identificador de cuerpo es porque fue encontrado y que todos los sobrevivientes no tienen uno.

```
found_bodies_df = titanic_df[(titanic_df.body.notna())]
survivors_df = titanic_df[titanic_df.survived == 1]
survivors_count = survivors_df.shape[0]
found_bodies_count = found_bodies_df.shape[0] + survivors_count
```

   Resultado: Se encontraron 621 cuerpos. De los cuales, 121 se encontraron sin vida y 500 sobrevivieron

   Con respecto a los hombres mayores de 40 años:

```
male_plus40_survivors_df = survivors_df[(survivors_df.sex == 'male') & (
    survivors_df.age > 40)]
found_male_plus40_df = found_bodies_df[(found_bodies_df.sex == 'male') & (
    found_bodies_df.age > 40)]
male_plus40_count = male_plus40_survivors_df.shape[0] + found_male_plus40_df.
    shape[0]
```

   Respuesta: Número hombres mayores de 40 años: 61. De los cuales, 25 sobrevivieron y 36 no.

   b) ¿Cuántas mujeres desaparecidas con edades entre 15 y 35 años?

```
missing_women_df = titanic_df[(titanic_df.body.isna()) & (titanic_df.sex == '
    female')]
missing_women_15_30 = missing_women_df[(missing_women_df.age >= 15) & (
    missing_women_df.age <= 30)].shape[0]

```

   Respuesta: Mujeres desaparecidas entre 15 y 30 años: 181

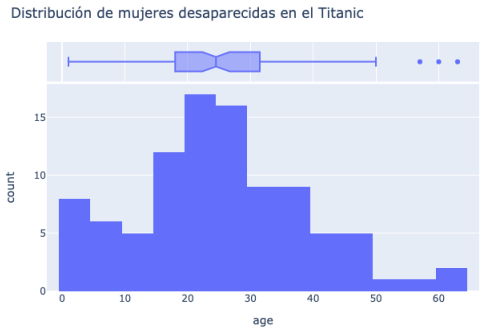


Figura 9: Distribución de mujeres no encontradas

   c) ¿Cuántos hombres mayores de 20 años sobrevivieron?

```
survivors_male_20_df = survivors_df[(survivors_df.age > 20) & (survivors_df.sex
    == 'male')]
num_survivors_male20 = survivors_male_20_df.shape[0]

```

   Respuesta: Hombres sobrevivientes mayores a 20 años: 96 - 22 % de sobrevivientes.

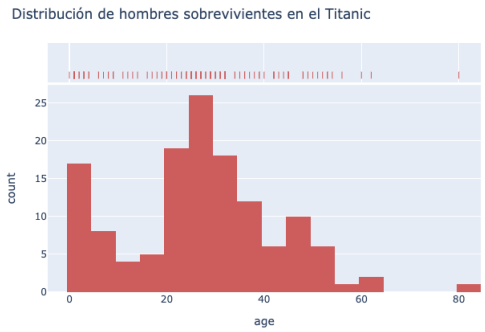


Figura 10: Distribución de hombres sobrevivientes

d) ¿Cuántas mujeres menores de 25 años sobrevivieron?

```
num_female_25 = female_survivors_df[female_survivors_df.age < 25].shape[0]

```

Respuesta: Mujeres sobrevivientes menores a 25 años: 124 - 29 %

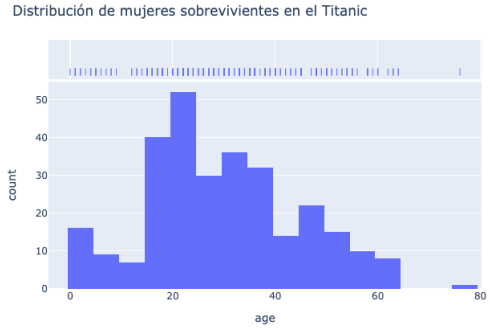


Figura 11: Distribución de mujeres sobrevivientes

2. Genere una copia del *dataset* y rellene los datos faltantes (*NAs*) con un valor de 0 en el caso de datos numéricos usados como identificador, la palabra "desconocido" en el caso de datos tipo cadena de caracteres y en el caso de variables numéricas use el promedio de los valores de esa columna (p.e., la edad y la tarifa)

   a) Identificación de campos con datos faltantes:

```
df_copy = titanic_df.copy()
df_copy.isna().any()
```

```
pclass       False
survived     False
name         False
sex          False
age           True
sibsp        False
parch        False
ticket       False
fare          True
cabin         True
embarked      True
boat          True
body          True
home.dest     True
dtype: bool
```

	cabin	embarked	boat
0	B5	S	2
1	C22 C26	S	11
2	C22 C26	S	NaN
3	C22 C26	S	NaN
4	C22 C26	S	NaN

Figura 12: Verificación de existencia de campos **nan**

b) Reemplazamiento de datos faltantes

```
df_copy.age = df_copy.age.fillna(int(np.mean(df_copy.age)))
df_copy.fare = df_copy.fare.fillna(int(np.mean(df_copy.fare)))
df_copy.cabin = df_copy.cabin.fillna('Desconocido')
df_copy.embarked = df_copy.embarked.fillna('Desconocido')
df_copy.boat = df_copy.boat.fillna(0)
df_copy.body = df_copy.body.fillna(int(np.mean(df_copy.body)))
df_copy['home.dest'] = df_copy['home.dest'].fillna('Desconocido')
```

c) Verificación:

```
pclass       False
survived     False
name         False
sex          False
age          False
sibsp        False
parch        False
ticket       False
fare         False
cabin        False
embarked     False
boat         False
body         False
home.dest    False
dtype: bool
```

	cabin	embarked	boat
0	B5	S	2
1	C22 C26	S	11
2	C22 C26	S	0
3	C22 C26	S	0
4	C22 C26	S	0

Figura 13: Verificación de resultados

3. De los campos *age* y *fare* agregue columnas al *dataset* que contenga los valores normalizados. Elija la normalización de tipo

$$\frac{x_i - \bar{x}}{\sigma}$$

en el caso de que la variable tenga una distribución normal, y la normalización tipo

$$\frac{x_i - x_{min}}{x_{max} - x_{min}}$$

en otro caso.

Para determinar si alguno de los dos campos seguían una distribución normal apliqué a ambos la prueba de *Shapiro*. Estos fueron los resultados:

- *p-value* para *age*: $8{,}97 \times 10^{-20}$
- *p-value* para *fare*: $8{,}97 \times 10^{-20}$

Por lo que podemos concluir que ninguno de los dos sigue una distribución normal.

```
def normalize(values):
    min_ = min(values)
    max_ = max(values)
    return (values - min_) / (max_ - min_)

df_copy['age_norm'] = normalize(age)
df_copy['fare_norm'] = normalize(fare)

```

	age	fare	age_norm	fare_norm
0	29.0	211.3375	0.3625	0.412503
1	1.0	151.5500	0.0125	0.295806
2	2.0	151.5500	0.0250	0.295806
3	30.0	151.5500	0.3750	0.295806
4	25.0	151.5500	0.3125	0.295806

Figura 14: Columnas normalizadas

3. Distancias

Usando los datos de *movies.csv* contesta lo siguiente:

1. Construya una o varias funciones que permitan calcular la matriz de distancias para los datos numéricos en el *dataFrame*. La función debe permitir construir la matriz de distancia usando las medidas de Manhattan, Euclideana y de Minkowski (para $p \geq 3$).

   - **Distancia Manhattan**

```
X = movies_df.to_numpy()
own_distance_matrix(X, X, distance='manhattan')
```

	0	1	2	3	4	5	6	7	8	9
0	0.0	-10.9	-14.1	-2.7	-8.2	-9.5	-16.5	-14.4	-7.2	-9.3
1	10.9	0.0	-3.2	8.2	2.7	1.4	-5.6	-3.5	3.7	1.6
2	14.1	3.2	0.0	11.4	5.9	4.6	-2.4	-0.3	6.9	4.8
3	2.7	-8.2	-11.4	0.0	-5.5	-6.8	-13.8	-11.7	-4.5	-6.6
4	8.2	-2.7	-5.9	5.5	0.0	-1.3	-8.3	-6.2	1.0	-1.1
5	9.5	-1.4	-4.6	6.8	1.3	0.0	-7.0	-4.9	2.3	0.2
6	16.5	5.6	2.4	13.8	8.3	7.0	0.0	2.1	9.3	7.2
7	14.4	3.5	0.3	11.7	6.2	4.9	-2.1	0.0	7.2	5.1
8	7.2	-3.7	-6.9	4.5	-1.0	-2.3	-9.3	-7.2	0.0	-2.1
9	9.3	-1.6	-4.8	6.6	1.1	-0.2	-7.2	-5.1	2.1	0.0

Figura 15: Distancia Manhattan

   - **Distancia Euclideana**

```
X = movies_df.to_numpy()
own_distance_matrix(X, X, distance='euclidean')
```

	0	1	2	3	4	5	6	7	8	9
0	0.000000	6.759438	10.338762	6.914478	5.282045	6.891299	9.406912	8.630180	8.786353	9.009439
1	6.759438	0.000000	7.687652	10.546089	6.122908	6.122091	5.037857	7.006426	10.641898	10.357606
2	10.338762	7.687652	0.000000	8.723531	10.963120	6.712675	8.611620	11.189727	10.342630	12.019151
3	6.914478	10.546089	8.723531	0.000000	8.904493	6.777905	11.231207	10.731729	6.093439	8.408329
4	5.282045	6.122908	10.963120	8.904493	0.000000	6.194352	6.250600	3.701351	7.811530	6.168468
5	6.891299	6.122091	6.712675	6.777905	6.194352	0.000000	5.049752	6.312686	4.956813	5.969925
6	9.406912	5.037857	8.611620	11.231207	6.250600	5.049752	0.000000	4.272002	8.704596	7.742093
7	8.630180	7.006426	11.189727	10.731729	3.701351	6.312686	4.272002	0.000000	7.833262	5.485435
8	8.786353	10.641898	10.342630	6.093439	7.811530	4.956813	8.704596	7.833262	0.000000	3.442383
9	9.009439	10.357606	12.019151	8.408329	6.168468	5.969925	7.742093	5.485435	3.442383	0.000000

Figura 16: Distancia Euclideana

- **Distancia Minkowski**

```
X = movies_df.to_numpy()
own_distance_matrix(X, X, distance='minkowski', p=3)
```

	0	1	2	3	4	5	6	7	8	9
0	0.0	-10.9	-14.1	-2.7	-8.2	-9.5	-16.5	-14.4	-7.2	-9.3
1	10.9	0.0	-3.2	8.2	2.7	1.4	-5.6	-3.5	3.7	1.6
2	14.1	3.2	0.0	11.4	5.9	4.6	-2.4	-0.3	6.9	4.8
3	2.7	-8.2	-11.4	0.0	-5.5	-6.8	-13.8	-11.7	-4.5	-6.6
4	8.2	-2.7	-5.9	5.5	0.0	-1.3	-8.3	-6.2	1.0	-1.1
5	9.5	-1.4	-4.6	6.8	1.3	0.0	-7.0	-4.9	2.3	0.2
6	16.5	5.6	2.4	13.8	8.3	7.0	0.0	2.1	9.3	7.2
7	14.4	3.5	0.3	11.7	6.2	4.9	-2.1	0.0	7.2	5.1
8	7.2	-3.7	-6.9	4.5	-1.0	-2.3	-9.3	-7.2	0.0	-2.1
9	9.3	-1.6	-4.8	6.6	1.1	-0.2	-7.2	-5.1	2.1	0.0

Figura 17: Distancia Minkowski con $p = 3$

2. Compare sus resultados con los del método **distance_matrix** de **scipy.spatial**

```
X = movies_df.to_numpy()
dm_own = own_distance_matrix(X, X, distance='euclidean', p=2)
dm_scipy = pd.DataFrame(data=distance_matrix(X, X, p=2))
dm_own == dm_scipy
```

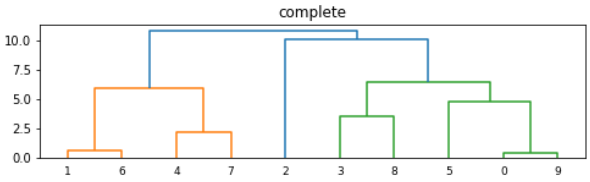
	0	1	2	3	4	5	6	7	8	9
0	True	True	True	True	True	True	True	True	True	True
1	True	True	True	True	True	True	True	True	True	True
2	True	True	True	True	True	True	True	True	True	True
3	True	True	True	True	True	True	True	True	True	True
4	True	True	True	True	True	True	True	True	True	True
5	True	True	True	True	True	True	True	True	True	True
6	True	True	True	True	True	True	True	True	True	True
7	True	True	True	True	True	True	True	True	True	True
8	True	True	True	True	True	True	True	True	True	True
9	True	True	True	True	True	True	True	True	True	True

Figura 18: Comparación para distancia Euclideana y **scipy.spatial.distance_matrix()**

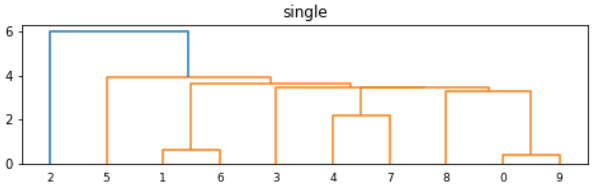
4. Clustering Jerárquico

En el anexo (sección 5.1) se encuentran todas las funciones empleadas para el código de esta sección.

1. Usando los métodos *dendrogram* y *linkage* construya un diagrama en forma de árbol (dendrograma) para el conjunto de datos en *movie.csv*



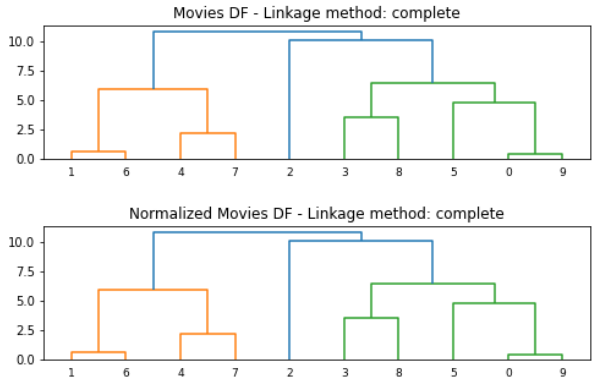
(a) Dendrograma con el método **complete**



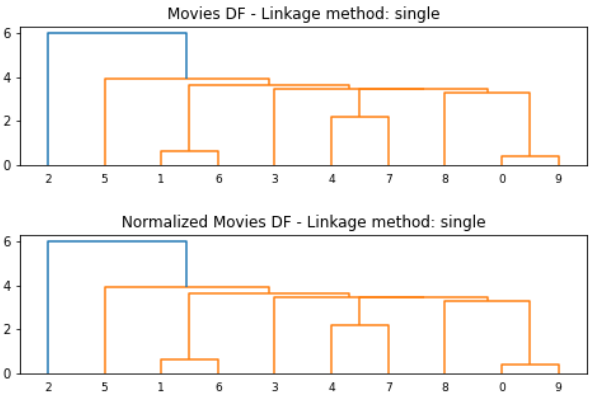
(b) Dendrograma con el método **single**

Figura 19: Dendrogramas para el dataset de *movies.csv*

2. Repita el proceso ahora usando algún esquema de normalización del rango de los datos



(a) Dendrogramas con el método **complete**



(b) Dendrogramas con el método **single**

Figura 20: Dendrogramas para el dataset de *movies.csv*

3. ¿Qué diferencias puede encontrar en los resultados previos?

Ninguna, ambos dendrogramas (utilizando los métodos de *complete* y *single*) son idénticos.

4. ¿En qué casos resulta importante llevar a cabo un proceso de normalización del rango de datos?

   Nos interesa normalizar un conjunto de datos cuando no queremos que una variable tenga una influencia mayor a la deseada simplemente por la escala de la medición original. Es decir, dentro del conjunto de datos podemos tener variables que estén muy alejados de la media y para un modelo, esto puede ser un problema pues tendrá más influencia sobre las demás variables.

5. Consulte los diferentes tipos de distancias que se pueden usar como parámetro en el método *linkage*. ¿En qué características de los datos se podría basar para elegir una determinada distancia? Los métodos que se pueden ocupar son:

   - single
   - complete
   - average
   - weighted
   - centroid
   - median
   - ward

   El método *single* el cual define a dos clusters como la distancia mínima entre dos de sus miembros es muy efectivo si queremos encontrar estructuras complejas dentro de los datos. Es decir, si los datos no son linealmente separados, este método nos funciona muy bien pues puede agrupar de manera correcta estructuras complejas. Por otro lado, si creemos que la suma de los cuadrados de nuestros datos es pequeña, podemos emplear el método de *Ward* pues cuantifica cuanto incrementará la suma de los cuadrados si agrupamos dos clústers.

5. Anexo

5.1. Código en Python para la sección de Clustering Jerárquico 4

```
from scipy.cluster import hierarchy
def plot_dendrogram(df, method, title=None):
    clusters = hierarchy.linkage(movies_df, method=method)
    plt.figure(figsize=(8,2))
    dendrogram = hierarchy.dendrogram(clusters, labels=None,orientation="top",
    leaf_font_size=9, leaf_rotation=360)
    title_str = method if title is None else title + ' - Linkage method: ' +method
    plt.title(title_str)
    plt.show()

def show_dendrograms(df, second_df=None, extended=False):
    methods = ['complete','single', 'average', 'ward','centroid', 'weighted', 'median']
    s = len(methods) if extended else 4
    for method in methods[:s]:
        if second_df is None:
            plot_dendrogram(df, method)
        else:
            plot_dendrogram(df, method, title='Original DF')
            plot_dendrogram(second_df, method, title='Second DF')

def normalize_df(original_df, start_col=0, end_col=None):
    if end_col is None:
        end_col = original_df.shape[1]
    norm_df = pd.DataFrame()
    columns = original_df.columns[start_col:end_col]
    for col in columns: norm_df[col] = normalize(original_df[col])
    return norm_df
```

Referencias

[1] Garduño E, "Notas de Visualización de la Información: Distancias", Marzo del 2021, Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas, Universidad Nacional Autónoma de México.

[2] Garduño E, "Notas de Visualización de la Información: Normalización", Marzo del 2021, Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas, Universidad Nacional Autónoma de México.

[3] SciPy, "scipy.cluster.hierarchy.linkage" [en línea] revisado el 20 de marzo del 2021 en: `https://docs.scipy.org/doc/scipy/reference/generated/scipy.cluster.hierarchy.linkage.html`

[4] Bruce P., Bruce A., Gedeck P., "Practical Statistics for Data Scientists". O'Reilly, Segunda Edición, 2020.