Tarea 5 - Filtrado de datos y distancias

Alumno : Axel Daniel Malváez Flores

No. Cta: 318315126

Antecedentes

Sea un conjunto de vectores $X = \{x_i | x_i \in \mathbb{R}^n e 1 \leq i \leq N\}$, donde $x_i = (x_{i,1}, x_{i,2}, \ldots, x_{i,n})^T$. Una distancia $d(a, b)$ entre dos vectores $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{R}^n$ se puede considerar como la *longitud* de la
trayectoria que une a ambos vectores ***a*** y ***b***. Existen varias definiciones de distancia:

Distancia de Manhattan (norma L_1):

$$d(\boldsymbol{a}, \boldsymbol{b}) = \sum_{i=1}^{n} |a_i - b_i|$$

Distancia Euclidiana (norma L_2):

$$d(\boldsymbol{a}, \boldsymbol{b}) = \left(\sum_{i=1}^{n} (a_i - b_i)^2 \right)^{1/2}$$

Distancia de Minkowski:

$$d(\boldsymbol{a}, \boldsymbol{b}) = \left(\sum_{i=1}^{n} |a_i - b_i|^p \right)^{1/p}$$

Actividades

a).- **Filtrado de datos**: Realice la lectura del archivo “titanic3.csv” y responda las siguientes preguntas:

Utilizando las librerías las cuales nos ayudarán a leer nuestro archivo y a manipular nuestros datos:

```
import numpy as np
import pandas as pd

titanic = pd.read_csv('titanic3.csv')
titanic.head()
```

	pclass	survived	name	sex	age	sibsp	parch	ticket	fare	cabin	embarked	boat	body	home.dest
0	1	1	Allen, Miss. Elisabeth Walton	female	29.0000	0	0	24160	211.3375	B5	S	2	NaN	St Louis, MO
1	1	1	Allison, Master. Hudson Trevor	male	0.9167	1	2	113781	151.5500	C22 C26	S	11	NaN	Montreal, PQ / Chesterville, ON
2	1	0	Allison, Miss. Helen Loraine	female	2.0000	1	2	113781	151.5500	C22 C26	S	NaN	NaN	Montreal, PQ / Chesterville, ON
3	1	0	Allison, Mr. Hudson Joshua Creighton	male	30.0000	1	2	113781	151.5500	C22 C26	S	NaN	135.0	Montreal, PQ / Chesterville, ON
4	1	0	Allison, Mrs. Hudson J C (Bessie Waldo Daniels)	female	25.0000	1	2	113781	151.5500	C22 C26	S	NaN	NaN	Montreal, PQ / Chesterville, ON

1. **¿Cuántos cuerpos fueron encontrados?**

```
# We assume that 1 is yes and 0 is no
found_bodies = titanic[titanic['body'].notnull()]
print(f'Total cuerpos encontrados fueron : {len(found_bodies)}')

>> Total cuerpos encontrados fueron : 121
```

2. **¿Cuántos de ellos fueron hombres mayores de 40 años?**

```
men_40 = found_bodies[(found_bodies['age'] > 40) &
                      (found_bodies['sex'] == 'male')]
print(f'Total hombres sin vida mayores a 40 : {len(men_40)}')

>> Total hombres sin vida mayores a 40 : 39
```

3. **¿Cuántas mujeres desaparecieron entre las edades de 15 a 35 años?**

```
disappeared_women = titanic[(titanic['sex'] == 'female') &
                            (titanic['survived'] == 0) &
                            (titanic['body'].isnull()) &
                            (titanic['age'] >= 15) & (titanic['age'] <= 35)]
print(f'Total de mujeres desaparecidas entre 15 y 35 años : {len(disappeared_women)}')

>> Total de mujeres desaparecidas entre 15 y 35 años : 51
```

4. **¿Cuántos hombres mayores de 20 años sobrevivieron?**

```
survivor_20 = titanic[(titanic['sex'] == 'male') &
                      (titanic['age'] > 20) &
                      (titanic['survived'] == 1) ]

print(f'Total de hombres mayores de 20 años que sobrevivieron : {len(survivor_20)}')

>> Total de hombres mayores de 20 años que sobrevivieron : 96
```

5. **¿Cuántas mujeres menores de 25 años sobrevivieron?**

```
survivor_25 = titanic[(titanic['sex'] == 'female') &
                      (titanic['age'] < 25) &
                      (titanic['survived'] == 1) ]

print(f'Total de mujeres menores de 25 años que sobrevivieron : {len(survivor_25)}')

>> Total de mujeres menores de 25 años que sobrevivieron : 124
```

Además, genere una copia del conjunto de datos y rellene los datos faltantes (NA's) 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 (por ejemplo, la *edad* y la *tarifa*).

Primero creamos una copia de nuestros datos, para directamente modificar esta copia y no nuestros datos originales. Veremos de qué tipo es cada columna y los NA's que contiene:

```
# We copy the original data
titanic_filled = titanic.copy()

# Verify the data types
titanic_filled.dtypes

# Verify the nulls in each column
titanic_filled.isna().sum()
```

Posteriormente llenamos todos nuestros campos nulos o faltantes como se nos indica:

```
# ID's with 0
titanic_filled['body'] = titanic_filled['body'].fillna(0)

# Strings with 'desconocido'
titanic_filled[['boat', 'embarked', 'home.dest', 'cabin']] = titanic_filled[['boat', 'embarked', 'home.dest', 'cabin']].fillna('desconoc

# Numeric with mean
titanic_filled[['age','fare']] = titanic_filled[['age','fare']].fillna(titanic_filled[['age','fare']].mean())

# Verify the nulls now
titanic_filled.isna().sum()

>> pclass       0
   survived     0
   name         0
   sex          0
   age          0
   sibsp        0
   parch        0
   ticket       0
   fare         0
   cabin        0
   embarked     0
   boat         0
   body         0
   home.dest    0
   dtype: int64
```

Finalmente, de los campos "*age*" y "*fare*" agregue columnas al conjunto de datos que contengan los *valores normalizados*. Elija la normalización tipo

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

para el caso de que la variable tenga una *distribución normal* y utilice la *normalización* tipo

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

en cualquier otro caso.

En esta parte haremos las pruebas de normalidad con la función **shapiro()** de la librería **scipy.stats**. El test de Shapiro-Wilk evalúa la hipótesis nula de que una muestra proviene de una población con una distribución normal. Si el *pvalue* es menor a 0.05 (esto porque queremos un intervalo de confianza del 95%) rechazamos la hipótesis nula y por lo tanto los datos no se distribuyen normalmente, de otro modo no podemos rechazar y por lo tanto si se distribuyen normalmente.

```
from scipy.stats import shapiro

# Realizing shapiro test for both age and fate columns
test_statistic_age, p_value_age = shapiro(titanic_filled['age'])
test_statistic_fare, p_value_fare = shapiro(titanic_filled['fare'])

# Our confidence
alpha = 0.05

if p_value_age > alpha:
    print('La columna age sigue una distribución normal (p-value = {})'.format(p_value_age))
else:
    print('La columna age no sigue una distribución normal (p-value = {})'.format(p_value_age))


if p_value_fare > alpha:
    print('La columna age sigue una distribución normal (p-value = {})'.format(p_value_fare))
else:
    print('La columna age no sigue una distribución normal (p-value = {})'.format(p_value_fare))


>> La columna age no sigue una distribución normal (p-value = 4.709205401713391e-19)
>> La columna age no sigue una distribución normal (p-value = 0.0)
```

Dado que ninguna de las columnas sigue una distribución normal, entonces normalizamos la columna con la segunda fórmula:

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

Creamos una función que nos normaliza los valores de alguna columna utilizando esta fórmula:

```
def normalize_column(column):
    '''
    Function that normalize a column given as a Series type from Pandas
    '''
    n = len(column)
    x_min = min(column)
    x_max = max(column)
    vals = []
    for x in column:
        vals.append((x - x_min) / (x_max - x_min))
    ser = pd.Series(vals, index=[i for i in range(n)])
    return ser
```

Así normalizamos tanto las columnas *age* y *fare* y las pegamos hasta el final de nuestra tabla copiada:

```
titanic_norm = pd.concat([titanic_filled,
                          normalize_column(titanic_filled['age'])],
                          axis=1)
titanic_norm = pd.concat([titanic_norm,
                          normalize_column(titanic_filled['fare'])],
                          axis=1)
titanic_norm.columns = titanic_filled.columns.tolist() +
                       ['age_normalized', 'fare_normalized']
```

La nueva table **titanic_norm** se ve como:

	pclass	survived	name	sex	age	sibsp	parch	ticket	fare	cabin	embarked	boat	body	home.dest	age_normalized	fare_normalized
0	1	1	Allen, Miss. Elisabeth Walton	female	29.000000	0	0	24160	211.3375	B5	S	2	0.0	St Louis, MO	0.361169	0.412503
1	1	1	Allison, Master. Hudson Trevor	male	0.916700	1	2	113781	151.5500	C22 C26	S	11	0.0	Montreal, PQ / Chesterville, ON	0.009395	0.295806
2	1	0	Allison, Miss. Helen Loraine	female	2.000000	1	2	113781	151.5500	C22 C26	S	desconocido	0.0	Montreal, PQ / Chesterville, ON	0.022964	0.295806
3	1	0	Allison, Mr. Hudson Joshua Creighton	male	30.000000	1	2	113781	151.5500	C22 C26	S	desconocido	135.0	Montreal, PQ / Chesterville, ON	0.373695	0.295806
4	1	0	Allison, Mrs. Hudson J C (Bessie Waldo Daniels)	female	25.000000	1	2	113781	151.5500	C22 C26	S	desconocido	0.0	Montreal, PQ / Chesterville, ON	0.311064	0.295806
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1304	3	0	Zabour, Miss. Hileni	female	14.500000	1	0	2665	14.4542	desconocido	C	desconocido	328.0	desconocido	0.179540	0.028213
1305	3	0	Zabour, Miss. Thamine	female	29.881135	1	0	2665	14.4542	desconocido	C	desconocido	0.0	desconocido	0.372206	0.028213
1306	3	0	Zakarian, Mr. Mapriededer	male	26.500000	0	0	2656	7.2250	desconocido	C	desconocido	304.0	desconocido	0.329854	0.014102
1307	3	0	Zakarian, Mr. Ortin	male	27.000000	0	0	2670	7.2250	desconocido	C	desconocido	0.0	desconocido	0.336117	0.014102
1308	3	0	Zimmerman, Mr. Leo	male	29.000000	0	0	315082	7.8750	desconocido	S	desconocido	0.0	desconocido	0.361169	0.015371

b).- **Distancias**: Utilizando el archivo “movies.csv” construya una o varias funciones que permitan calcular una *matriz de distancias* para los datos numéricos en el *dataFrame*. La función debe permitir construir la matriz de distancia usando las distancias de Manhattan, Euclideana y de Minkowski (para *p* igual a 3).

```
movies = pd.read_csv('movies.csv', sep=';')
movies.head()
```

	user_id	star_wars	lord_of_the_rings	harry_potter
0	1	1.2	4.9	2.1
1	2	2.1	8.1	7.9
2	3	7.4	3.0	9.9
3	4	5.6	0.5	1.8
4	5	1.5	8.3	2.6

Una *matriz de distancia* es una matriz cuadrada qué contiene las distancias entre los elementos de un conjunto (medidas un par a la vez).

Obtenemos las columnas que son numéricas y que no indican o aportan información relevante tales como ID’s:

```
numeric_movies = movies.loc[:,['star_wars', 'lord_of_the_rings', 'harry_potter']]
```

Definimos las distancias:

```
def manhattan_distance(a,b):
    '''
    Function that calculates the manhattan distance
    '''
    return sum(abs(a-b))

def euclidean_distance(a,b):
    '''
    Function that calculates the euclidean distance
```

```
    '''
    return np.sqrt(sum((a - b)**2))

def minkowski_distance(a,b,p):
    '''
    Function that calculates the minkowski distance
    '''
    return sum(abs(a - b)**p)**(1/p)
```

Creamos nuestra función **distance_matrix** propia:

```
def distance_matrix_me(data, dist = 'euclidean'):
    '''
    Function that calculates the matrix of distances
    '''
    n = len(data)
    matrix = np.zeros((n,n))
    for i in range(n):
        for j in range(n):
            if dist == 'euclidean':
                matrix[i][j] = euclidean_distance(data.iloc[i], data.iloc[j])
            elif dist == 'manhattan':
                matrix[i][j] = manhattan_distance(data.iloc[i], data.iloc[j])
            else:
                matrix[i][j] = minkowski_distance(data.iloc[i], data.iloc[j], 3)
    return matrix
```

Compare sus resultados con los que se obtienen por medio del método **distance_matrix** de **scipy.spatial**.

```
# Function made by us to calculate the distance matrix
matrix_euclidean = distance_matrix_me(numeric_movies, 'euclidean')
matrix_euclidean
```

```
array([[ 0.        ,  6.68505797, 10.14347081,  6.22976725,  3.44963766,
         4.74236228,  7.24499827,  5.04777179,  3.63318042,  0.41231056],
       [ 6.68505797,  0.        ,  7.6223356 , 10.35470907,  5.33760246,
         4.63465209,  0.6164414 ,  3.6180105 ,  8.01560977,  6.57875368],
       [10.14347081,  7.6223356 ,  0.        ,  8.66602562, 10.77914653,
         6.00499792,  7.62627039, 10.01049449,  8.42436941,  9.77036335],
       [ 6.22976725, 10.35470907,  8.66602562,  0.        ,  8.84816365,
         6.47610994, 10.82312339,  9.95841353,  3.48281495,  5.89067059],
       [ 3.44963766,  5.33760246, 10.77914653,  8.84816365,  0.        ,
         6.11310069,  5.92199291,  2.16794834,  6.70969448,  3.61247837],
       [ 4.74236228,  4.63465209,  6.00499792,  6.47610994,  6.11310069,
         0.        ,  4.94974747,  5.98748695,  3.94588393,  4.43170396],
       [ 7.24499827,  0.6164414 ,  7.62627039, 10.82312339,  5.92199291,
         4.94974747,  0.        ,  4.15331193,  8.47171765,  7.13722635],
       [ 5.04777179,  3.6180105 , 10.01049449,  9.95841353,  2.16794834,
         5.98748695,  4.15331193,  0.        ,  7.76916984,  5.10783712],
       [ 3.63318042,  8.01560977,  8.42436941,  3.48281495,  6.70969448,
         3.94588393,  8.47171765,  7.76916984,  0.        ,  3.29393382],
       [ 0.41231056,  6.57875368,  9.77036335,  5.89067059,  3.61247837,
         4.43170396,  7.13722635,  5.10783712,  3.29393382,  0.        ]])
```

```
from scipy.spatial import distance_matrix
matrix_euclidean_s = distance_matrix(numeric_movies, numeric_movies, p=2)
matrix_euclidean_s
```

```
array([[ 0.        ,  6.68505797, 10.14347081,  6.22976725,  3.44963766,
         4.74236228,  7.24499827,  5.04777179,  3.63318042,  0.41231056],
       [ 6.68505797,  0.        ,  7.6223356 , 10.35470907,  5.33760246,
         4.63465209,  0.6164414 ,  3.6180105 ,  8.01560977,  6.57875368],
       [10.14347081,  7.6223356 ,  0.        ,  8.66602562, 10.77914653,
         6.00499792,  7.62627039, 10.01049449,  8.42436941,  9.77036335],
       [ 6.22976725, 10.35470907,  8.66602562,  0.        ,  8.84816365,
         6.47610994, 10.82312339,  9.95841353,  3.48281495,  5.89067059],
       [ 3.44963766,  5.33760246, 10.77914653,  8.84816365,  0.        ,
         6.11310069,  5.92199291,  2.16794834,  6.70969448,  3.61247837],
       [ 4.74236228,  4.63465209,  6.00499792,  6.47610994,  6.11310069,
         0.        ,  4.94974747,  5.98748695,  3.94588393,  4.43170396],
       [ 7.24499827,  0.6164414 ,  7.62627039, 10.82312339,  5.92199291,
         4.94974747,  0.        ,  4.15331193,  8.47171765,  7.13722635],
       [ 5.04777179,  3.6180105 , 10.01049449,  9.95841353,  2.16794834,
         5.98748695,  4.15331193,  0.        ,  7.76916984,  5.10783712],
       [ 3.63318042,  8.01560977,  8.42436941,  3.48281495,  6.70969448,
         3.94588393,  8.47171765,  7.76916984,  0.        ,  3.29393382],
       [ 0.41231056,  6.57875368,  9.77036335,  5.89067059,  3.61247837,
         4.43170396,  7.13722635,  5.10783712,  3.29393382,  0.        ]])
```

Notamos que efectivamente la función que hicimos nos da los mismos resultados que la de **scipy.spatial.distance_matrix.**

Además, usando los métodos “*dendrogram*” y “*linkage*” construya un diagrama en forma de árbol (*dendrograma*) para el conjunto de datos en “movies.csv”.
Repita el proceso ahora usando algún esquema de *normalización* del rango de los datos.

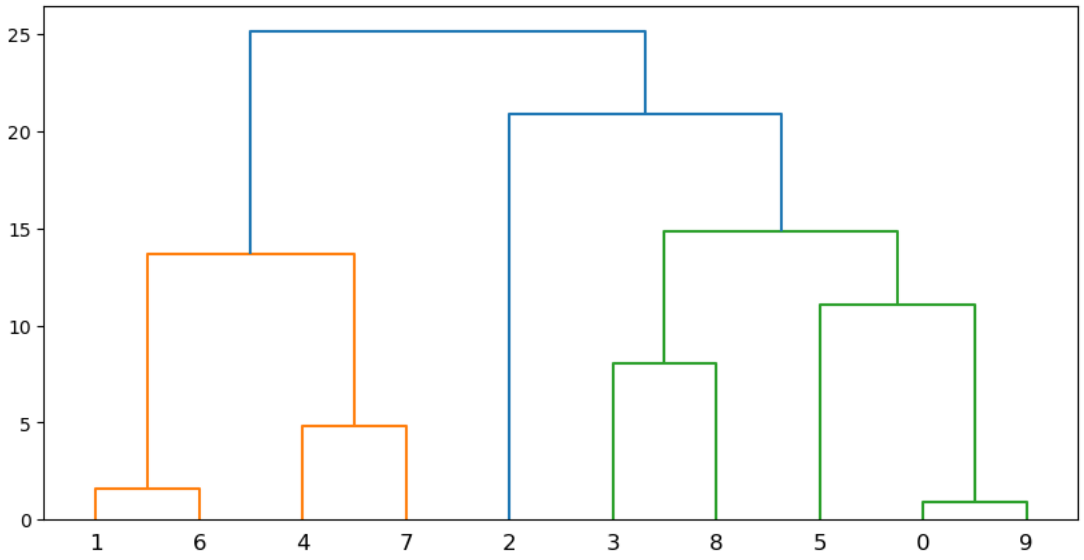
Utilizaremos las siguientes librerías para escalar y normalizar nuestros datos:

```
from sklearn.preprocessing import StandardScaler, Normalizer
n = Normalizer()
datos_norm = n.fit_transform(numeric_movies)
matrix_euclidean_norm = distance_matrix(datos_norm, datos_norm, p=2)
```

Dendrogram

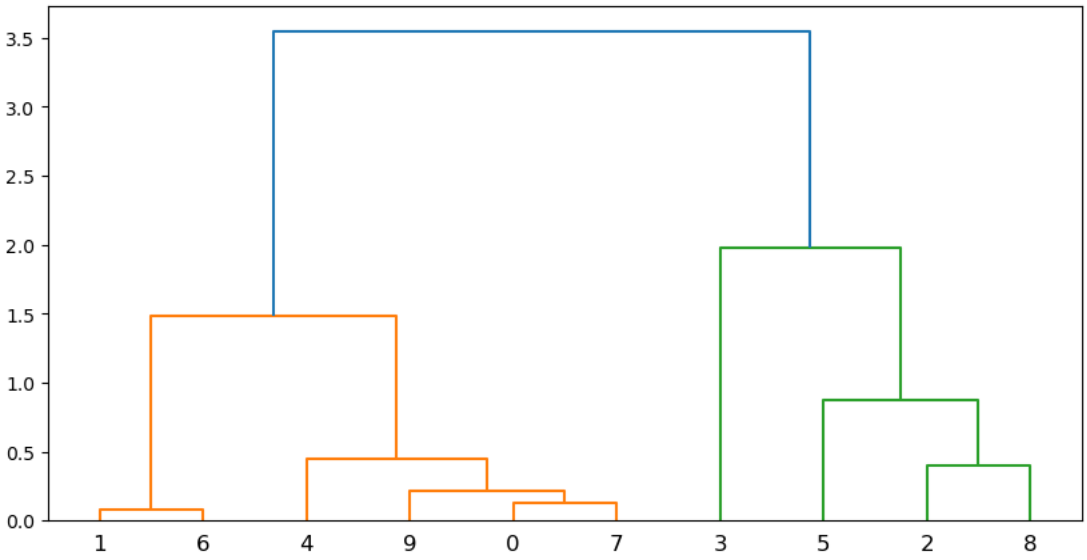
- Datos Originales

```
from scipy.cluster.hierarchy import linkage, dendrogram
import matplotlib.pyplot as plt

# Compute the linkage matrix using Ward's method
Z = linkage(matrix_euclidean, 'ward')

# Plot the dendrogram
plt.figure(figsize=(10, 5))
dendrogram(Z)
plt.show()
```



- Datos Normalizados

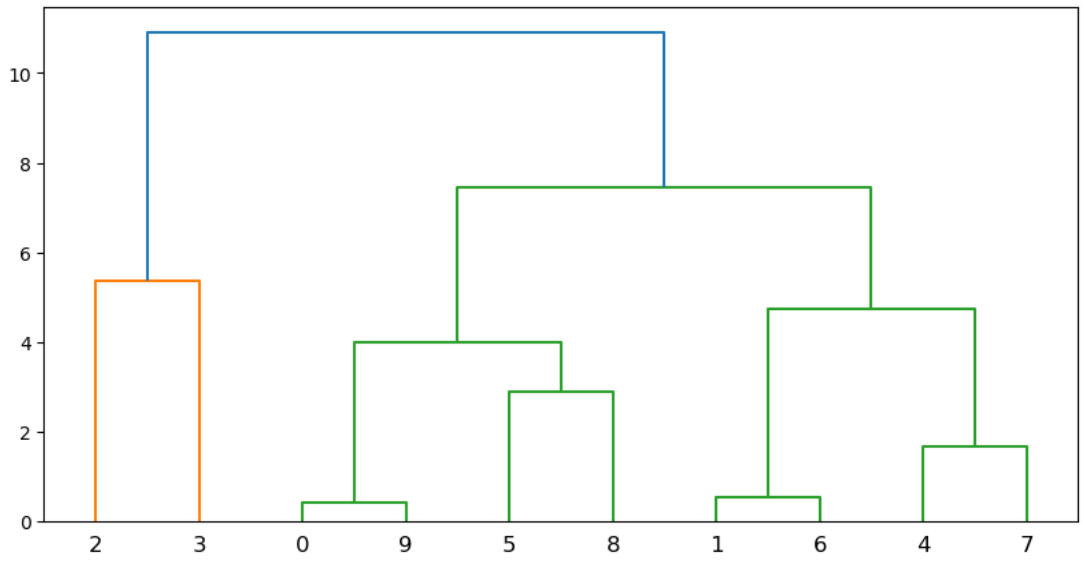
```
# Compute the linkage matrix using Ward's method
Z = linkage(matrix_euclidean_norm, 'ward')

# Plot the dendrogram
plt.figure(figsize=(10, 5))
dendrogram(Z)
plt.show()
```



- Datos Escalados

```
sc = StandardScaler()
datos_norm = sc.fit_transform(numeric_movies)
matrix_euclidean_norm = distance_matrix(datos_norm, datos_norm, p=2)
```

```
# Compute the linkage matrix using Ward's method
Z = linkage(matrix_euclidean_norm, 'ward')

# Plot the dendrogram
plt.figure(figsize=(10, 5))
dendrogram(Z)
plt.show()
```



1. **¿Qué diferencias puede encontrar en los resultados previos?**

   Notemos que a diferencia de los datos originales y los datos normalizados, las alturas del dendrograma cambia dependiendo también del tipo de normalización. En este caso la altura mínima la tiene la Normalización de los datos (cada uno con norma 1), posteriormente la estandarización (media cero y varianza uno). Al final en todos cambian los clústers.

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

   En el caso de que queramos implementar algún modelo de Machine Learning la normalización puede ayudar a mejorar la precisión del modelo y reducir la influencia de las variables con rangos de valores más grandes. Así mismo el escalar nuestros datos resulta importante si lo que haremos será comparar columnas que tienen datos en diferente magnitud.

   En estos casos el Normalizer lo que hace es normalizar nuestros datos llevándolos a una norma de uno (esto es útil en agrupamiento o análisis de similitud). Esto con la finalidad que que aquellos valores más grandes no dominen a los más pequeños. Por otra parte StandardScaler lo que hace es pasar nuestros datos a media cero y desviación estándar 1, esto para tener nuestros datos distribuidos normalmente (esto es útil en estadística o en modelos que asumen normalidad).

3. **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 uno para elegir una determinada distancia?

   Basándonos en las características de nuestros datos podemos elegir una cierta métrica dentro de nuestro método "*linkage*", esto en general nos dice que tipo de distancia tomará para medir esta misma entre dos puntos (el parámetro *metric*, únicamente es usado cuando pasamos nuestros datos aún como una matrix con nuestros vectores (no de distancias)).

   - Si notamos que nuestros datos tienen valores atípicos, una métrica como las distancias Euclidianas o Manhattan no servirán de mucho (esto porque la media no es un valor representativo de la columna). En este caso es mejor Mahalanobis, etc.

Conclusión

En esta práctica se utilizaron conceptos de filtrado de datos y cálculo de distancias para resolver una serie de preguntas utilizando el dataset "titanic3.csv". También se construyó una función para calcular una matriz de distancias utilizando diferentes tipos de distancia para el dataset "movies.csv".

Se aprendió la importancia de la normalización de datos para algunos algoritmos de cálculo de distancias, así como la selección de diferentes tipos de distancia según las características particulares de los datos.

En general, la práctica permitió reforzar los conceptos de manipulación y análisis de datos en Python, así como el uso de librerías especializadas como pandas y scipy.spatial.