Análisis del conjunto de datos con Pandas

Cuando tenemos un conjunto de datos, el objetivo de las Ciencias de Datos es procesar este conjunto para generar nueva información. Ejemplo de tal procesamiento puede ser resumir la información agregando y agrupando los datos. Pandas ofrece una conjunto de operaciones que nos permite hacer el procesamiento de datos de forma fácil y eficiente.

Inicialización del entorno.

Lo primero será importar el paquete Pandas con alias pd. Posteriormente visualizamos la versión usada de Pandas ya que es un dato importante para consultar la documentación. En el momento de editar este notebook la versión de pandas es: 1.4.3

Además para facilitar los ejercicios también importamos el paquete Numpy con el alias np.

In [1]:
```
import pandas as pd
import numpy as np
np.set_printoptions(floatmode='fixed', precision=3)
print('Pandas versión: ', pd.__version__)
```

```
Pandas versión:  1.4.3
```

Obteniendo un resumen general.

Pandas ofrece unas conjunto de funciones que permiten obtener una primera visión resumida de un conjunto de datos.

Ejercicio 01: Dado un dataset `df1` se quiere mostrar las primeras filas.

El resultado debe ser algo parecido a lo siguiente:

	a	b	c	d
A	1.764052	0.400157	0.978738	2.240893
B	1.867558	-0.977278	0.950088	-0.151357
C	-0.103219	0.410599	0.144044	1.454274
D	0.761038	0.121675	0.443863	0.333674
E	1.494079	-0.205158	0.313068	-0.854096

In [2]:
```
np.random.seed(0)
df1 = pd.DataFrame(np.random.randn(40).reshape(10,4),
                   index=list('ABCDEFGHIJ'),
                   columns=list('abcd'))
#Pon tu código aquí.
#Sugerencia usa el método head() para mostrar las primeras filas.

#
```

Ejercicio 02: Dado un dataset `df1` se quiere mostrar las últimas filas.

El resultado debe ser algo parecido a lo siguiente:

```
          a         b         c         d
F -2.552990  0.653619  0.864436 -0.742165
G  2.269755 -1.454366  0.045759 -0.187184
H  1.532779  1.469359  0.154947  0.378163
I -0.887786 -1.980796 -0.347912  0.156349
J  1.230291  1.202380 -0.387327 -0.302303
```

In [3]:
```
np.random.seed(0)
df1 = pd.DataFrame(np.random.randn(40).reshape(10,4),
                   index=list('ABCDEFGHIJ'),
                   columns=list('abcd'))
#Pon tu código aquí.
#Sugerencia usa el método tail() para mostrar las primeras filas.

#
```

Ejercicio 03: Dado un dataset `df1` se quiere mostrar unas estadísticas generales por columnas.

El resultado debe ser algo parecido a lo siguiente:

```
              a          b          c          d
count  10.000000  10.000000  10.000000  10.000000
mean    0.737556  -0.035981   0.315970   0.232625
std     1.501947   1.125385   0.496926   0.960543
min    -2.552990  -1.980796  -0.387327  -0.854096
25%     0.112845  -0.784248   0.070330  -0.273523
50%     1.362185   0.260916   0.234008   0.002496
75%     1.706234   0.592864   0.759293   0.367040
max     2.269755   1.469359   0.978738   2.240893
```

In [4]:
```
np.random.seed(0)
df1 = pd.DataFrame(np.random.randn(40).reshape(10,4),
                   index=list('ABCDEFGHIJ'),
                   columns=list('abcd'))
#Pon tu código aquí.
#Sugerencia usa el método describe() para mostrar estadísticas
#por columnas.

#
```

Agregación de datos.

Pandas ofrece una series de funciones para obtener datos agregados a partir de un objeto Series o un objeto Dataframe.

Ejercicio 04: Data una serie queremos mostrar su valor medio, y su varianza.

El resultado debe ser algo parecido a lo siguiente:

```
Serie:
 0    0.636962
```

```
1    0.269787
2    0.040974
3    0.016528
4    0.813270
5    0.912756
6    0.606636
7    0.729497
8    0.543625
9    0.935072
dtype: float64

Media    : 0.5505105129032412
Varianza : 0.1127145331212907
```

In [5]:

```
gen=np.random.default_rng(0)
ser = pd.Series(gen.random(10))
print('Serie: \n', ser)
mean = 0.0
var = 0.0
#Pon tu código aquí.
#Sugerencia: utiliza los métodos mean() y var() de la serie.

#
print("\nMedia    : {}".format(mean))
print("Varianza : {}".format(var))
```

```
Serie:
 0    0.636962
1    0.269787
2    0.040974
3    0.016528
4    0.813270
5    0.912756
6    0.606636
7    0.729497
8    0.543625
9    0.935072
dtype: float64

Media    : 0.0
Varianza : 0.0
```

Ejercicio 05: Dado un dataframe queremos mostrar los valores medio y varianza para cada columna.

El resultado debe ser algo parecido a lo siguiente:

```
Dataframe:
          a         b         c
A  0.636962  0.269787  0.040974
B  0.016528  0.813270  0.912756
C  0.606636  0.729497  0.543625
D  0.935072  0.815854  0.002739

Medias
 a    0.548799
b    0.657102
c    0.375023
dtype: float64

Varianzas
```

```
 a    0.147879
b    0.068282
c    0.189256
dtype: float64
```

In [6]:
```
gen=np.random.default_rng(0)
df = pd.DataFrame(gen.random(12).reshape(4,3),
                  index=list('ABCD'),
                    columns=list('abc'))
print('Dataframe: \n', df)
mean = 0.0
var = 0.0
#Pon tu código aquí.
#Sugerencia: utiliza los métodos mean() y var() del dataframe.

#
print("\nMedias\n", mean)
print("\nVarianzas\n",var)
```

```
Dataframe:
          a         b         c
A  0.636962  0.269787  0.040974
B  0.016528  0.813270  0.912756
C  0.606636  0.729497  0.543625
D  0.935072  0.815854  0.002739

Medias
 0.0

Varianzas
 0.0
```

Ejercicio 06: Data un dataframe queremos normalizarlo en media y varianza, esto es, asumiendo que cada fila del dataframe es una muestra, y las columnas son características medidas, queremos transformar el dataframe de forma que:

$$\text{df2.iloc}[i,j] = \frac{\text{df1.iloc}[i,j] - \mu_j}{\sigma_j}$$

donde μ_j y σ_j son la media y desviación de la columna j.

El resultado debe ser algo parecido a lo siguiente:

```
Dataframe:
          a         b         c
A  0.636962  0.269787  0.040974
B  0.016528  0.813270  0.912756
C  0.606636  0.729497  0.543625
D  0.935072  0.815854  0.002739

Dataframe normalizado:
          a         b         c
A  0.229261 -1.482220 -0.767867
B -1.384139  0.597643  1.236066
C  0.150400  0.277048  0.387559
D  1.004479  0.607529 -0.855757
```

In [7]:
```
gen=np.random.default_rng(0)
df1 = pd.DataFrame(gen.random(12).reshape(4,3),
```

```
                  index=list('ABCD'),
                    columns=list('abc'))
print('Dataframe: \n', df)
df2 = np.zeros_like(df1)
mean = 0.0
stdev = 0.0
#Pon tu código aquí.
#Sugerencia: utiliza los métodos mean() y var(). Recuerda
#   dividir con sigma = sqrt(var). Aplica funciones universales
#   para realizar el proceso del dataframe de forma vectorizada.
#   No se debe utilizar bucles.

#
print("\nDataframe normalizado:\n", df2)
```

```
Dataframe:
          a         b         c
A  0.636962  0.269787  0.040974
B  0.016528  0.813270  0.912756
C  0.606636  0.729497  0.543625
D  0.935072  0.815854  0.002739

Dataframe normalizado:
 [[0.000 0.000 0.000]
 [0.000 0.000 0.000]
 [0.000 0.000 0.000]
 [0.000 0.000 0.000]]
```

Las funciones de agregación filtran los valores `NaN` por defecto.

Ejercicio 07: Comprobar que los valores NAN son filtrados de forma automática al aplicar la funciones de agregación.

El resultado debe ser algo parecido a lo siguiente:

```
Serie:
 0    1.0
1    2.0
2    NaN
3    3.0
dtype: float64

Valor mínimo: 1.0
Valor máximo: 3.0
```

In [8]:

```
s1=pd.Series([1, 2, None, 3])
print('Serie:\n',s1)
min_v = 0;
max_v = 0;
#Pon tu código aquí.
#Sugerencia: utiliza los métodos min()/max().

#
print('\nValor mínimo: {}'.format(min_v))
print('Valor máximo: {}'.format(max_v))
```

```
Serie:
 0    1.0
1    2.0
2    NaN
3    3.0
dtype: float64
```

```
Valor mínimo: 0
Valor máximo: 0
```

Si trabajamos con un objeto DataFrame podemos indicar sobre qué eje (filas o columnas) operarar usando el argumento `axis`.

Ejercicio 08: Dado un dataframe queremos saber el rango de valores en cada fila.

El resultado debe ser algo parecido a lo siguiente:

```
Dataframe:
    a   b   c
A  0   1   2
B  3   4   5
C  6   7   8
D  9  10  11

Valores mínimos por fila:
 A    0
B    3
C    6
D    9
dtype: int64

Valores máximos por fila:
 A     2
B     5
C     8
D    11
dtype: int64
```

In [9]:
```
df1 = pd.DataFrame(np.arange(12).reshape((4,3)),
                   index=list('ABCD'),
                     columns=list('abc'))
print('Dataframe: \n', df1)
min_v = 0.0
max_v = 0.0
#Pon tu código aquí.
#Sugerencia: utiliza los métodos min()/max() pero indicando el
#valor apropiado del argumento 'axis'.

#
print("\nValores mínimos por fila: \n", min_v)
print("\nValores máximos por fila: \n", max_v)
```

```
Dataframe:
    a   b   c
A  0   1   2
B  3   4   5
C  6   7   8
D  9  10  11

Valores mínimos por fila:
 0.0

Valores máximos por fila:
 0.0
```

Ejercicio 08: Tenemos un dataframe donde las filas son vectores sobre un espacio de características y queremos obtener una versión normalizada del dataframe donde los vectores (las filas) tengan norma L2 =

1.0, esto es,

$$\text{df2.iloc}[i,j] = \frac{\text{df1.iloc}[i,j]}{N_i}$$

donde N_i sería la suma de los valores de la fila i al cuadrado y se calcula de la forma

$$N_i = \sqrt{\sum_j \left(\text{df1.iloc}[i,j]\right)^2}$$

El resultado debe ser algo parecido a lo siguiente:

```
Dataframe:
    a   b   c
A  0   1   2
B  3   4   5
C  6   7   8
D  9  10  11

Dataframe normalizado:
           a         b         c
A  0.000000  0.447214  0.894427
B  0.424264  0.565685  0.707107
C  0.491539  0.573462  0.655386
D  0.517892  0.575435  0.632979
```

In [10]:
```
df1 = pd.DataFrame(np.arange(12).reshape((4,3)),
                   index=list('ABCD'),
                     columns=list('abc'))
print('Dataframe: \n', df1)
#Pon tu código aquí.
#Sugerencia: utiliza los métodos mul() y sum() indicando el
#   valor apropiado del argumento 'axis'.
#Recuerda que necesitamos usar la ufunc np.sqrt.
#Procura no utilizar bucles.

#
print('\nDataframe normalizado:\n', df2)
```

```
Dataframe:
    a   b   c
A  0   1   2
B  3   4   5
C  6   7   8
D  9  10  11

Dataframe normalizado:
 [[0.000 0.000 0.000]
 [0.000 0.000 0.000]
 [0.000 0.000 0.000]
 [0.000 0.000 0.000]]
```

Agrupación de datos.

En ocasiones, al trabajar con un objeto DataFrame queremos obtener datos agregados pero agrupando las filas según los valores de una columna denominada "clave" de la agrupación.

Por ejemplo supongamos un cojunto de datos sobre personas con una característica (columna) que indidica el "género". Podemos obtener el dato agregado "media" para la columna "altura" pero separando por un lado las filas con género "Masculino" del género "Femenino".

Esta operación se puede visualizar como tres pasos:

1. Dividimos el dataframe en tantos grupos como valores distintos tiene la columna "clave".
2. Para cada grupo se aplica la función de agregación indicada.
3. Se fusiona en un nuevo DataFrame las agregaciones para cada grupo.

Pandas realiza este proceso usando un objeto `GroupBy`.

Ejercicio 09: Dado el dataframe `df` queremos obtener la medias agrupando según la columna `'Clave'`.

El resultado debe ser algo parecido a lo siguiente:

```
Dataframe:
   Clave  V1  V2
0     A   0   7
1     B   7   2
2     C   6   5
3     A   5   8
4     B   8   3
5     A   5   6

Medias:
              V1   V2
Clave
A      3.333333  7.0
B      7.500000  2.5
C      6.000000  5.0
```

In [11]:
```
df = pd.DataFrame({'Clave':['A', 'B', 'C', 'A', 'B', 'A'],
                   'V1':[ 0, 7, 6, 5, 8, 5],
                   'V2':[ 7, 2, 5, 8, 3, 6]})
print('Dataframe:\n', df)
medias = 0
#Pon tu código aquí.
#Sugerencia: usa el método groupby() indicando la columna 'Clave'
#para obtener el objeto GroupBy y depués aplicale el método mean()

#
print('\nMedias:\n', medias)
```

```
Dataframe:
   Clave  V1  V2
0     A   0   7
1     B   7   2
2     C   6   5
3     A   5   8
4     B   8   3
5     A   5   6

Medias:
 0
```

El objeto GroupBy tiene un comportamiento similar a un DataFrame donde ahora el índice de las filas son distintos valores que aparecen en la columna usada como campo clave de agrupamiento.

Ejercicio 10: Dado un dataframe `df` queremos agrupar usando la columna `'Clave'` y obtener las medias del agrupamiento sólo de la columna con la etiqueta `'V1'`.

El resultado debe ser algo parecido a lo siguiente:

```
Dataframe:
   Clave  V1  V2
0     A   0   7
1     B   7   2
2     C   6   5
3     A   5   8
4     B   8   3
5     A   5   6

Medias para la columna "V1":
 Clave
A    3.333333
B    7.500000
C    6.000000
Name: V1, dtype: float64
```

In [12]:
```
df = pd.DataFrame({'Clave':['A', 'B', 'C', 'A', 'B', 'A'],
                   'V1':[ 0, 7, 6, 5, 8, 5],
                   'V2':[ 7, 2, 5, 8, 3, 6]})
print('Dataframe:\n', df)
media = 0
#Pon tu código aquí.
#Sugerencia: usa el método groupby() e indexa el objeto devuelto
#  como si fuera un dataframe para obtener la serie correspondiente
#  a la etiqueta 'V1'.

#
print('\nMedias para la columna "V1":\n', media)
```

```
Dataframe:
   Clave  V1  V2
0     A   0   7
1     B   7   2
2     C   6   5
3     A   5   8
4     B   8   3
5     A   5   6

Medias para la columna "V1":
 0
```

Otro uso interesante del objeto GroupBy es para filtar un DataFrame aplicando algún criterio a los datos agregados por grupo. Para ello utilizaremos el método `filter()`. A este método le damos un objeto *callable* que recibe como parámetro el objeto GroubBy, realiza alguna operación y devuelve una máscara (array de valores lógicos 'True' o 'False'). Se recomienda revisar la sección "Usando un objeto *callable* para indexar" del cuaderno "Indexación y selección de datos".

Ejercicio 11: Dado un dataframe `df`, se quiere filtar los datos que pertenezcan a grupos con un valor medio para la columna 'V2' menor que 5.0. Utiliza una expresión lambda para expresar el filtro.

El resultado debe ser algo parecido a lo siguiente:

```
Dataframe:
   Clave  V1  V2
```

```
   0     A   0   7
   1     B   7   2
   2     C   6   5
   3     A   5   8
   4     B   8   3
   5     A   5   6

   Medias según columna 'Clave':
                V1   V2
   Clave
   A      3.333333  7.0
   B      7.500000  2.5
   C      6.000000  5.0

   Dataframe filtrado:
      Clave  V1  V2
   0     A   0   7
   2     C   6   5
   3     A   5   8
   5     A   5   6
```

In [13]:

```python
df1 = pd.DataFrame({'Clave':['A', 'B', 'C', 'A', 'B', 'A'],
                    'V1':[ 0, 7, 6, 5, 8, 5],
                    'V2':[ 7, 2, 5, 8, 3, 6]})
print('Dataframe:\n', df1)
print('\nMedias según columna \'Clave\':\n',
      df1.groupby('Clave').mean())
df2 = np.zeros_like(df1)
#Pon tu código aquí.
#Sugerencia: utiliza el método groupby para agrupar por el campo 'Clave'
#   y aplica el método filter() al objeto GroupBy devuelto.
#Sugerencia: usa una expresión lambda para expresar el filtro:
#   la media para la columan 'V2' es mayor o igual que 5.0.

#
print('\nDataframe filtrado:\n', df2)
```

```
Dataframe:
   Clave  V1  V2
0     A   0   7
1     B   7   2
2     C   6   5
3     A   5   8
4     B   8   3
5     A   5   6

Medias según columna 'Clave':
             V1   V2
Clave
A      3.333333  7.0
B      7.500000  2.5
C      6.000000  5.0

Dataframe filtrado:
 [[0 0 0]
 [0 0 0]
 [0 0 0]
 [0 0 0]
 [0 0 0]
 [0 0 0]]
```

Como podemos ver el grupo 'B' tiene un valor medio para la columna 'V2' menor a 5.0 y por lo tanto los

datos (filas) del grupo B han sido filtrados del dataframe. Observa que el filtro mantedrán los datos para los cuales la expresión lógica se evalue a `True`.

Agrupando por más de un criterio.

También podemos especificar un conjunto de claves y obtendremos agrupaciones con más de un nivel. Para ello usaremos una lista para indicar las columnas con los criterios a usar.

Ejercicio 12: Sea el dataframe `df` queremos obtener una agrupación usando como criterios las colunmasn 'C1' y 'C2' y obtener las medias por cada sub-grupo.

El resultado debe ser algo parecido a lo siguiente:

```
   Dataframe:
      C1  C2         V
   0  A   2  0.110656
   1  B   2  0.999423
   2  C   1  0.543209
   3  A   1  0.893821
   4  B   2  0.955848
   5  A   2  0.814363

   Medias por subgrupo:
                  V
   C1 C2
   A  1   0.893821
      2   0.462509
   B  2   0.977636
   C  1   0.543209
```

In [14]:
```
df = pd.DataFrame({'C1':['A', 'B', 'C', 'A', 'B', 'A'],
                   'C2':[ 2, 2, 1, 1, 2, 2],
                   'V':np.random.rand(6)})
print('Dataframe:\n', df)
medias = 0
#Pon tu código aquí.
#Sugerencia: usa la lista ['C1', 'C2'] para agrupar.

#
print('\nMedias por subgrupo:\n', medias)
```

```
Dataframe:
   C1  C2         V
0  A   2  0.315428
1  B   2  0.363711
2  C   1  0.570197
3  A   1  0.438602
4  B   2  0.988374
5  A   2  0.102045

Medias por subgrupo:
 0
```

Discretizar variables continuas.

Pandas ofrece la función `cut()` para agrupar una variable contínua por intervalos ("*bins*").

Ejercicio 13: Dada la serie `s` con una característica continua, queremos obtener una agrupación (discretizarla) usando 3 intervalos. Mostrar los intervalos.

El resultado debe ser algo parecido a lo siguiente:

```
Serie:
 0    0.636962
1    0.269787
2    0.040974
3    0.016528
4    0.813270
5    0.912756
6    0.606636
7    0.729497
8    0.543625
9    0.935072
dtype: float64

Valores discretizados:
 0    2
1    0
2    0
3    0
4    2
5    2
6    1
7    2
8    1
9    2
dtype: int64

Intervalos:
 [0.016 0.323 0.629 0.935]
```

In [15]:

```
gen = np.random.default_rng(0)
s = pd.Series(gen.random(10))
print('Serie:\n', s)
s_discreta = 0
intervalos = 0
#Pon tu código aquí.
#Sugerencia: usa la función cut(). Recuerda que esta función
#   devuelve dos valores discritazados y los intervalos
#   creados. Utiliza los argumentos labels y retbins de forma
#   adecuada.

#
print('\nValores discretizados:\n', s_discreta)
print('\nIntervalos:\n', intervalos)
```

```
Serie:
 0    0.636962
1    0.269787
2    0.040974
3    0.016528
4    0.813270
5    0.912756
6    0.606636
7    0.729497
8    0.543625
9    0.935072
dtype: float64
```

```
Valores discretizados:
 0

Intervalos:
 0
```

Con estos intervalos podemos ahora agrupar nuestros datos y obtener valores agregados por por grupos.

Ejercicio 14: dado el dataframe `df` se requiere calcular medias agrupando la característica `'A'` en tres intervalos.

El resultado debe ser algo parecido a lo siguiente:

```
Dataframe:
          A         B         C
0  0.636962  0.269787  0.040974
1  0.016528  0.813270  0.912756
2  0.606636  0.729497  0.543625
3  0.935072  0.815854  0.002739
4  0.857404  0.033586  0.729655
5  0.175656  0.863179  0.541461
6  0.299712  0.422687  0.028320
7  0.124283  0.670624  0.647190
8  0.615385  0.383678  0.997210
9  0.980835  0.685542  0.650459

Intervalos para columna 'A'
 0     (0.338, 0.659]
1    (0.0156, 0.338]
2     (0.338, 0.659]
3     (0.659, 0.981]
4     (0.659, 0.981]
5    (0.0156, 0.338]
6    (0.0156, 0.338]
7    (0.0156, 0.338]
8     (0.338, 0.659]
9     (0.659, 0.981]
Name: A, dtype: category
Categories (3, interval[float64]): [(0.0156, 0.338] %lt; (0.338, 0.659]
%lt; (0.659, 0.981]]

Medias:
                           A         B         C
A
(0.0156, 0.338]  0.154045  0.692440  0.532431
(0.338, 0.659]   0.619661  0.460987  0.527269
(0.659, 0.981]   0.924437  0.511660  0.460951
```

In [16]:

```
gen = np.random.default_rng(0)
df = pd.DataFrame(gen.random(30).reshape((10,3)),
                  columns=list('ABC'))
print('Dataframe: \n', df)
intervalos = 0
#Pon tu código aquí.
#Sugerencia: Usa cut() sobre la serie correspodiente a la
#   a la columna 'A'

#
```

```
print('\nIntervalos para columna \'A\'\n', intervalos)
medias = 0
#Pon tu código aquí.
#Sugerencia: Usa groupby() usando los intervalos calculados
#   para agrupar.

#
print('\nMedias:\n', medias)
```

```
Dataframe:
          A         B         C
0  0.636962  0.269787  0.040974
1  0.016528  0.813270  0.912756
2  0.606636  0.729497  0.543625
3  0.935072  0.815854  0.002739
4  0.857404  0.033586  0.729655
5  0.175656  0.863179  0.541461
6  0.299712  0.422687  0.028320
7  0.124283  0.670624  0.647190
8  0.615385  0.383678  0.997210
9  0.980835  0.685542  0.650459

Intervalos para columna 'A'
 0

Medias:
 0
```

Tablas Pivote.

Pandas ofrece un mecanismo flexible para generar agrupamientos más complejos conocido como tabla pivote.

Ejercicio 15: Tenemos un dataset con tres columnas 'X', 'Y', 'Z'. Queremos obtener una tabla que muestre los valores medios de 'Z' agrupando los valores conjuntos de las columnas 'X', 'Y' en cuatro grupos cada una. Utiliza el valor 0.0 para rellenar datos incompletos.

El resultado debe ser algo parecido a lo siguiente:

```
Intervalos para X:
 0      (0.499, 0.74]
1    (0.0156, 0.258]
2      (0.499, 0.74]
3      (0.74, 0.981]
4      (0.74, 0.981]
5    (0.0156, 0.258]
6     (0.258, 0.499]
7    (0.0156, 0.258]
8      (0.499, 0.74]
9      (0.74, 0.981]
Name: X, dtype: category
Categories (4, interval[float64]): [(0.0156, 0.258] < (0.258, 0.499] <
(0.499, 0.74] < (0.74, 0.981]]

Intervalos para Y:
 0     (0.241, 0.448]
1     (0.656, 0.863]
2     (0.656, 0.863]
3     (0.656, 0.863]
```

```
4    (0.0328, 0.241]
5     (0.656, 0.863]
6     (0.241, 0.448]
7     (0.656, 0.863]
8     (0.241, 0.448]
9     (0.656, 0.863]
Name: Y, dtype: category
Categories (4, interval[float64]): [(0.0328, 0.241] < (0.241, 0.448] <
(0.448, 0.656] < (0.656, 0.863]]

Agrupamiento:
 X                (0.0156, 0.258]  (0.258, 0.499]  (0.499, 0.74]  (0.74,
0.981]
Y
(0.0328, 0.241]         0.000000         0.00000       0.000000
0.729655
(0.241, 0.448]          0.000000         0.02832       0.519092
0.000000
(0.656, 0.863]          0.700469         0.00000       0.543625
0.326599
```

In [17]:

```
gen = np.random.default_rng(0)
df = pd.DataFrame(gen.random(30).reshape(10,3),
                  columns=list('XYZ'))
intervalos_x = 0
intervalos_y = 0
#Pon tu código aquí.
#Sugerencia: usa la función cut() para generar los intervalos
#para las columnas X e Y. Recuerda que queremos 4 intervalos
#para cada una.

#
print('\nIntervalos para X:\n', intervalos_x)
print('\nIntervalos para Y:\n', intervalos_y)

grp = 0
#Pon tu código aquí.
#Sugerencia: usa la función Pandas.pivot_table() para agrupar usando
#   la función mean los datos en los intervalos para los ejes 'X'
#   e 'Y'. Los valores a agrupar son la columna 'Z'.
#   Recuerda que los datos incompletos se rellenan con 0.0

#
print('\nAgrupamiento:\n', grp)
```

```
Intervalos para X:
 0

Intervalos para Y:
 0

Agrupamiento:
 0
```

Tablas de frecuencias cruzadas.

Otra función para agrupar es `crosstab()` que permite construir tablas de frecuencias cruzadas.

Ejercicio 16: Tenemos un dataset con tres columnas 'GT', 'PRED' que corresponde a un proceso de evaluación de un clasificador donde 'GT' ("*Ground Truth*) sería la etiqueta conocidad a priori y 'PRED'

corresponde a la etiqueta estimada por el clasificador ("*predicted*").

Se quiere generar la matriz de confusión del proceso.

El resultado debe ser algo parecido a lo siguiente:

```
Datos cruzados:
 PRED   A   B   C   D   E
GT
A      37  35  35  32  56
B      29  37  33  38  43
C      41  43  46  43  32
D      41  42  46  41  39
E      38  42  51  42  38
```

In [18]:
```
gen = np.random.default_rng(0)
labels = list('ABCDE')
gt = [gen.choice(labels) for i in range(1000)]
pred = [gen.choice(labels) for i in range(1000)]
data = pd.DataFrame({'GT':gt, 'PRED':pred})

m_c = None
#Pon tu código aquí:
#Sugerencia: utiliza crosstab() con las series 'GT' y 'PRED'.

#
print("Datos cruzados:\n", m_c)
```

```
Datos cruzados:
 None
```

In [ ]:
```

```