09_Análisis_de_datos_cuantitativos_agrupados_tarea_01

Alberto Simón

04/05/2025

Reglas de Agrupación de Datos Numéricos

Los procesos son muy similares, así que defino una función común

```
def generar_clases_y_marcas(data, k):

    # Precisión por defecto de 0.1
    min_val = np.min(data)
    if min_val.is_integer():
        precision = 1
    else:
        decimals = str(min_val).split('.')
        if len(decimals) > 1:
            precision = 10 ** (-len(decimals[1]))
        else:
            precision = 0.1  # Default precision

    # Amplitud tromada a intervalos iguales
    data_range = np.max(data) - np.min(data)
    A = data_range / k
    # Redondeo de la amplitud según la precisión
    A = np.ceil(A / precision) * precision
    # Salvo que sea exacto
    if A == data_range / k:
        A += precision

    # Los límites, con  un medio de la precisión
    L1 = np.min(data) - 0.5 * precision
    L = L1 + A * np.arange(k + 1)

    # Las marcas de clase como punto medio
    X = (L[:-1] + L[1:]) / 2

    return L, X
```

Se importan los datos y se transforman a numpy.

```
import numpy as np
import pandas as pd

crab = pd.read_csv("../data/datacrab.txt", delimiter=" ", decimal=".", encoding="utf-8")
weight = crab['weight'].to_numpy()
```

```
print(crab['weight'].describe())
```

```
## count     173.000000
## mean     2437.190751
## std       577.025214
## min      1200.000000
## 25%      2000.000000
## 50%      2350.000000
## 75%      2850.000000
## max      5200.000000
## Name: weight, dtype: float64
```

Pregunta 1

Da el algoritmo para reproducir el proceso de generación de clases y sus marcas donde el número de clases ha sido obtenido con la regla de la Scott en Python.

```
def scott(data):
    num_data = len(data)
    bin_width = 3.5 * np.std(data) / np.power(num_data, 1/3)
    num_bins = int(np.ceil((np.max(data) - np.min(data)) / bin_width))
    print(num_bins)
    return num_bins

print("Regla de Scott")
```

```
## Regla de Scott
```

```
K_scott = scott(weight)
```

```
## 12
```

```
L_scott, X_scott = generar_clases_y_marcas(weight, K_scott)
print("Divisiones:", L_scott)
```

```
## Divisiones: [1199.5 1533.5 1867.5 2201.5 2535.5 2869.5 3203.5 3537.5 3871.5 4205.5
##  4539.5 4873.5 5207.5]
```

```
print("Marcas de clase:", X_scott)
```

```
## Marcas de clase: [1366.5 1700.5 2034.5 2368.5 2702.5 3036.5 3370.5 3704.5 4038.5 4372.5
##  4706.5 5040.5]
```

Pregunta 2

Da el algoritmo para reproducir el proceso de generación de clases y sus marcas donde el número de clases ha sido obtenido con la regla de la raíz en Python.

```
def raiz(data):
   num_data = len(data)
   num_bins = int(np.ceil(np.sqrt(num_data)))
   print(num_bins)
   return num_bins

print("Regla de la raíz")
```

```
## Regla de la raíz
```

```
K_raiz = raiz(weight)
```

```
## 14
```

```
L_raiz, X_raiz = generar_clases_y_marcas(weight, K_raiz)
print("Divisiones:", L_raiz)
```

```
## Divisiones: [1199.5 1485.5 1771.5 2057.5 2343.5 2629.5 2915.5 3201.5 3487.5 3773.5
##  4059.5 4345.5 4631.5 4917.5 5203.5]
```

```
print("Marcas de clase:", X_raiz)
```

```
## Marcas de clase: [1342.5 1628.5 1914.5 2200.5 2486.5 2772.5 3058.5 3344.5 3630.5 3916.5
##  4202.5 4488.5 4774.5 5060.5]
```

Pregunta 3

Da el algoritmo para reproducir el proceso de generación de clases y sus marcas donde el número de clases ha sido obtenido con la regla de Sturges en Python.

```
def sturges(data):
   # para numero datos mayores de 30
   num_data = len(data)
   num_bins = 1 + int(np.log2(num_data))
   print(num_bins)
   return num_bins

print("Regla de Sturges")
```

```
## Regla de Sturges
```

```
K_sturges = sturges(weight)
```

```
## 8
```

```
L_sturges, X_sturges = generar_clases_y_marcas(weight, K_sturges)
print("Divisiones:", L_sturges)
```

```
## Divisiones: [1199.5 1700.5 2201.5 2702.5 3203.5 3704.5 4205.5 4706.5 5207.5]
```

```
print("Marcas de clase:", X_sturges)
```

```
## Marcas de clase: [1450. 1951. 2452. 2953. 3454. 3955. 4456. 4957.]
```

Pregunta 4

Da el algoritmo para reproducir el proceso de generación de clases y sus marcas donde el número de clases ha sido obtenido con la regla de la Freedman-Diaconis en Python.

```
def freedman_diaconis(data):
   num_data = len(data)
   irq = np.percentile(data, 75) - np.percentile(data, 25)
   bin_width = 2 * irq / np.power(num_data, 1/3)
   num_bins = int((np.max(data) -  np.min(data)) / bin_width)  + 1
   print(num_bins)
   return num_bins

print("Regla de Freedman-Diaconis")
```

```
## Regla de Freedman-Diaconis
K_fd = sturges(weight)
```

```
## 8
L_fd, X_fd = generar_clases_y_marcas(weight, K_fd)
print("Divisiones:", L_fd)
```

```
## Divisiones: [1199.5 1700.5 2201.5 2702.5 3203.5 3704.5 4205.5 4706.5 5207.5]
print("Marcas de clase:", X_fd)
```

```
## Marcas de clase: [1450. 1951. 2452. 2953. 3454. 3955. 4456. 4957.]
```