Algoritmos_de_Estatistica

April 3, 2022

1 Demonstração de algoritmos de estatística em Python

[1]:
```
# Bibliotecas utilizadas

import statistics
import math
from scipy import stats
import pandas as pd
from collections import Counter
```

1.0.1 Média Aritmética

Para dados não agrupados = $[x_1, x_2, x_3, ...]$ ##

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i$$

[2]:
```
def media_aritmetica(dados):
    somatorio = 0
    n = len(dados)
    media = 0
    for x in dados:
        somatorio = somatorio + x
    if n > 0:
        media = somatorio / n
    return media
```

1.0.2 Média Aritmética Ponderada

Para dados agrupados = $[[w_1, x_1], [w_2, x_2], [w_3, x_3], ...]$ ##

$$\mu_p = \frac{1}{\sum_{i=1}^{n} w_i}\sum_{i=1}^{n} w_i x_i$$

[3]:
```
def media_aritmetica_ponderada(dados):
    somatorio = 0
```

```
    pesos = 0
    media = 0
    for i in dados:
        w = i[0]
        x = i[1]
        somatorio = somatorio + (w * x)
        pesos = pesos + w
    if pesos > 0:
        media = somatorio / pesos
    return media
```

1.0.3 Média Geométrica

Para dados não agrupados $= [x_1, x_2, x_3, ...]$ ##

$$\mu_g = \sqrt[n]{\prod_{i=1}^{n} x_i}$$

[4]:
```
def media_geometrica(dados):
    produtorio = 1
    n = len(dados)
    media = 0
    for x in dados:
        produtorio = produtorio * x
    if n > 0:
        media = pow(produtorio, 1 / n)
    return media
```

1.0.4 Média Geométrica Ponderada

Para dados agrupados $= [[w_1, x_1], [w_2, x_2], [w_3, x_3], ...]$ ##

$$\mu_{gp} = \sqrt[\Sigma_{i=1}^{n} w_i]{\prod_{i=1}^{n} {x_i}^{w_i}}$$

[5]:
```
def media_geometrica_ponderada(dados):
    produtorio = 1
    pesos = 0
    media = 0
    for i in dados:
        w = i[0]
        x = i[1]
        produtorio = produtorio * pow(x, w);
        pesos = pesos + w
    if pesos > 0:
```

```
        media = pow(produtorio, 1 / pesos)
    return media
```

1.0.5 Média Harmônica

Para dados não agrupados = $[x_1, x_2, x_3, ...]$ ##

$$\mu_h = \frac{n}{\sum_{i=1}^{n} \frac{1}{x_i}}$$

[6]:
```
def media_harmonica(dados):
    somatorio = 0
    n = len(dados)
    media = 0
    for x in dados:
        somatorio = somatorio + (1 / x)
    if somatorio != 0:
        media = n / somatorio
    return media
```

1.0.6 Média Harmônica Ponderada

Para dados agrupados = $[[w_1, x_1], [w_2, x_2], [w_3, x_3], ...]$ ##

$$\mu_{hp} = \frac{\sum_{i=1}^{n} w_i}{\sum_{i=1}^{n} \frac{w_i}{x_i}}$$

[7]:
```
def media_harmonica_ponderada(dados):
    somatorio = 0
    pesos = 0
    media = 0
    for i in dados:
        w = i[0]
        x = i[1]
        somatorio = somatorio + (w / x)
        pesos = pesos + w
    if somatorio != 0:
        media = pesos / somatorio
    return media
```

1.0.7 Média Quadrática

Para dados não agrupados = $[x_1, x_2, x_3, ...]$ ##

$$\mu_q = \sqrt{\frac{1}{n} \sum_{i=1}^{n} {x_i}^2}$$

[8]:
```
def media_quadratica(dados):
    somatorio = 0
    n = len(dados)
    media = 0
    for x in dados:
        somatorio = somatorio + pow(x, 2)
    if n > 0:
        media = math.sqrt(somatorio / n)
    return media
```

1.0.8 Média Quadrática Ponderada

Para dados agrupados = $[[w_1, x_1], [w_2, x_2], [w_3, x_3], ...]$ ##

$$\mu_{qp} = \sqrt{\frac{1}{\sum_{i=1}^{n} w_i} \sum_{i=1}^{n} w_i {x_i}^2}$$

[9]:
```
def media_quadratica_ponderada(dados):
    somatorio = 0
    pesos = 0
    media = 0
    for i in dados:
        w = i[0]
        x = i[1]
        somatorio = somatorio + (w * pow(x, 2))
        pesos = pesos + w
    if pesos > 0:
        media = math.sqrt(somatorio / pesos)
    return media
```

1.0.9 Média Cúbica

Para dados não agrupados = $[x_1, x_2, x_3, ...]$ ##

$$\mu_c = \sqrt[3]{\frac{1}{n} \sum_{i=1}^{n} {x_i}^3}$$

[10]:
```
def media_cubica(dados):
    somatorio = 0
    n = len(dados)
    media = 0
    for x in dados:
        somatorio = somatorio + pow(x, 3)
    if n > 0:
        media = pow((somatorio / n), 1/3)
```

```
    return media
```

1.0.10 Média Cúbica Ponderada

Para dados agrupados = $[[w_1, x_1], [w_2, x_2], [w_3, x_3], ...]$ ##

$$\mu_{cp} = \sqrt[3]{\frac{1}{\sum_{i=1}^{n} w_i} \sum_{i=1}^{n} w_i {x_i}^3}$$

[11]:
```
def media_cubica_ponderada(dados):
    somatorio = 0
    pesos = 0
    media = 0
    for i in dados:
        w = i[0]
        x = i[1]
        somatorio = somatorio + (w * pow(x, 3))
        pesos = pesos + w
    if pesos > 0:
        media = pow((somatorio / pesos), 1/3)
    return media
```

1.0.11 Média Desarmônica

Para dados não agrupados = $[x_1, x_2, x_3, ...]$ ##

$$\mu_d = \frac{2}{\frac{1}{\frac{\sum_{i=1}^{n} x_i}{n}} + \frac{1}{\frac{\left(\frac{\sum_{i=1}^{n} x_i}{n}\right)^2}{\frac{n}{\sum_{i=1}^{n} \frac{1}{x_i}}}}}$$

[12]:
```
def media_desarmonica(dados):
    sx = 0
    sxi = 0
    n = len(dados)
    for x in dados:
        sx = sx + x
        sxi = sxi + (1 / x)
    media = 2 / ((1 / (sx / n)) + (1 / (pow(sx / n, 2) / (n / sxi))))
    return media
```

1.0.12 Média Desarmônica Ponderada

Para dados agrupados $= [[w_1, x_1], [w_2, x_2], [w_3, x_3], ...]$ ##

$$\mu_{dp} = \frac{2}{\frac{1}{\frac{\sum_{i=1}^{n} w_i x_i}{\sum_{i=1}^{n} w_i}} + \frac{1}{\frac{\left(\frac{\sum_{i=1}^{n} w_i x_i}{\sum_{i=1}^{n} w_i}\right)^2}{\frac{\sum_{i=1}^{n} w_i}{\sum_{i=1}^{n} \frac{w_i}{x_i}}}}}$$

[13]:
```
def media_desarmonica_ponderada(dados):
    spwx = 0
    sw = 0
    sdwx = 0
    for i in dados:
        w = i[0]
        x = i[1]
        spwx = spwx + (w * x)
        sw = sw + w
        sdwx = sdwx + (w / x)
    media = 2 / ((1 / (spwx / sw)) + (1 / (pow(spwx / sw, 2) / (sw / sdwx))))
    return media
```

1.0.13 Mediana

Para dados não agrupados $= [x_1, x_2, x_3, ...]$

[14]:
```
def mediana(dados):
    dados.sort()
    n = len(dados)
    i = round(n / 2) - 1
    if (n % 2) != 0:
        mediana = dados[i]
    else:
        mediana = media_aritmetica([dados[i], dados[i + 1]]);
    return mediana
```

1.0.14 Moda

Para dados não agrupados $= [x_1, x_2, x_3, ...]$

[15]:
```
def moda(dados):
    moda = []
    contagem = Counter(dados).most_common()
    frequencias = [list(i) for i in contagem]
    if frequencias[0][1] > 1:
        for i in frequencias:
            if i[1] == frequencias[0][1]:
                moda.append(i[0])
```

```
            else:
                break
    return moda
```

1.0.15 Desvio Absoluto Médio

Para dados não agrupados $= [x_1, x_2, x_3, ...]$ ##

$$D_{am} = \frac{1}{n}\sum_{i=1}^{n}|x_i - \mu|$$

[16]:
```
def desvio_medio(dados):
    somatorio = 0
    n = len(dados)
    media = media_aritmetica(dados)
    desvio = 0
    for x in dados:
        somatorio = somatorio + abs(x - media)
    if n > 0:
        desvio = somatorio / n
    return desvio
```

1.0.16 Desvio Absoluto Mediano

Para dados não agrupados $= [x_1, x_2, x_3, ...]$ ##

$$D_{am} = Md(|x_i - \tilde{x}|)$$

[17]:
```
def desvio_mediano(dados):
    md = mediana(dados)
    desvios = []
    for x in dados:
        desvios.append(abs(x - md))
    desvio = mediana(desvios)
    return desvio
```

1.0.17 Variância Populacional

Para dados não agrupados $= [x_1, x_2, x_3, ...]$ ##

$$\sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2$$

[18]:
```
def variancia_populacional(dados):
    somatorio = 0
    n = len(dados)
```

```
    media = media_aritmetica(dados)
    variancia = 0
    for i in dados:
        somatorio = somatorio + pow(i - media, 2)
    if n > 0:
        variancia = somatorio / n
    return variancia
```

1.0.18 Desvio Padrão Populacional

Para dados não agrupados $= [x_1, x_2, x_3, ...]$ ##

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2}$$

[19]:
```
def desvio_padrao_populacional(dados):
    variancia = variancia_populacional(dados)
    desvio = math.sqrt(variancia)
    return desvio
```

1.0.19 Variância Amostral

Para dados não agrupados $= [x_1, x_2, x_3, ...]$ ##

$$s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$$

[20]:
```
def variancia_amostral(dados):
    somatorio = 0
    n = len(dados)
    media = media_aritmetica(dados)
    variancia = 0
    for i in dados:
        somatorio = somatorio + pow(i - media, 2)
    if n > 1:
        variancia = somatorio / (n - 1)
    return variancia
```

1.0.20 Desvio Padrão Amostral

Para dados não agrupados $= [x_1, x_2, x_3, ...]$ ##

$$s = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

[21]:
```
def desvio_padrao_amostral(dados):
    variancia = variancia_amostral(dados)
    desvio = math.sqrt(variancia)
    return desvio
```

1.0.21 Variância Populacional

Para dados agrupados = $[[w_1, x_1], [w_2, x_2], [w_3, x_3], ...]$ ##

$$\sigma^2 = \frac{1}{\sum_{i=1}^{n} w_i} \sum_{i=1}^{n} \left((x_i - \mu)^2 w_i\right)$$

[22]:
```
def variancia_populacional_agrupado(dados):
    somatorio = 0
    pesos = 0
    media = media_aritmetica_ponderada(dados)
    variancia = 0
    for i in dados:
        w = i[0]
        x = i[1]
        somatorio = somatorio + (pow(x - media, 2) * w)
        pesos = pesos + w
    if pesos > 0:
        variancia = somatorio / pesos
    return variancia
```

1.0.22 Desvio Padrão Populacional

Para dados agrupados = $[[w_1, x_1], [w_2, x_2], [w_3, x_3], ...]$ ##

$$\sigma = \sqrt{\frac{1}{\sum_{i=1}^{n} w_i} \sum_{i=1}^{n} \left((x_i - \mu)^2 w_i\right)}$$

[23]:
```
def desvio_padrao_populacional_agrupado(dados):
    variancia = variancia_populacional_agrupado(dados)
    desvio = math.sqrt(variancia)
    return desvio
```

1.0.23 Variância Amostral

Para dados agrupados = $[[w_1, x_1], [w_2, x_2], [w_3, x_3], ...]$ ##

$$s^2 = \frac{1}{\sum_{i=1}^{n} w_i - 1} \sum_{i=1}^{n} \left((x_i - \bar{x})^2 w_i\right)$$

[24]:
```python
def variancia_amostral_agrupado(dados):
    somatorio = 0
    pesos = 0
    media = media_aritmetica_ponderada(dados)
    variancia = 0
    for i in dados:
        w = i[0]
        x = i[1]
        somatorio = somatorio + (pow(x - media, 2) * w)
        pesos = pesos + w
    if pesos > 1:
        variancia = somatorio / (pesos - 1)
    return variancia
```

1.0.24 Desvio Padrão Amostral

Para dados agrupados = $[[w_1, x_1], [w_2, x_2], [w_3, x_3], ...]$ ##

$$s = \sqrt{\frac{1}{\sum_{i=1}^{n} w_i - 1} \sum_{i=1}^{n} \left((x_i - \bar{x})^2 w_i\right)}$$

[25]:
```python
def desvio_padrao_amostral_agrupado(dados):
    variancia = variancia_amostral_agrupado(dados)
    desvio = math.sqrt(variancia)
    return desvio
```

1.0.25 Coeficiente de Variação

Para dados não agrupados = $[x_1, x_2, x_3, ...]$ ##

$$CV = \frac{\sigma}{\mu} \times 100$$

[26]:
```python
def coeficiente_variacao(dados):
    desvio = desvio_padrao_populacional(dados)
    media = media_aritmetica(dados)
    cv = desvio / media * 100
    return cv / 100
```

1.0.26 Coeficiente de Variação

Para dados agrupados = $[[w_1, x_1], [w_2, x_2], [w_3, x_3], ...]$ ##

$$CV = \frac{\sigma}{\mu} \times 100$$

[27]:
```python
def coeficiente_variacao_agrupado(dados):
    desvio = desvio_padrao_populacional_agrupado(dados)
    media = media_aritmetica_ponderada(dados)
    cv = desvio / media * 100
    return cv / 100
```

1.0.27 Covariância Populacional

Para dados correlacionados = $[[x_1, y_1], [x_2, y_2], [x_3, y_3], ...]$ ##

$$\sigma_{xy} = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu_x)(y_i - \mu_y)$$

[28]:
```python
def covariancia_populacional(dados):
    somatorio = 0
    n = len(dados)
    covariancia = 0
    dadosx = []
    dadosy = []
    for i in dados:
        x = i[0]
        y = i[1]
        dadosx.append(x)
        dadosy.append(y)
    mediax = media_aritmetica(dadosx)
    mediay = media_aritmetica(dadosy)
    for i in dados:
        x = i[0]
        y = i[1]
        somatorio = somatorio + ((x - mediax) * (y - mediay))
    if n > 0:
        covariancia = somatorio / n
    return covariancia
```

1.0.28 Covariância Amostral

Para dados correlacionados = $[[x_1, y_1], [x_2, y_2], [x_3, y_3], ...]$ ##

$$s_{xy} = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})$$

[29]:
```python
def covariancia_amostral(dados):
    somatorio = 0
    n = len(dados)
    covariancia = 0
    dadosx = []
```

```
    dadosy = []
    for i in dados:
        x = i[0]
        y = i[1]
        dadosx.append(x)
        dadosy.append(y)
    mediax = media_aritmetica(dadosx)
    mediay = media_aritmetica(dadosy)
    for i in dados:
        x = i[0]
        y = i[1]
        somatorio = somatorio + ((x - mediax) * (y - mediay))
    if n > 1:
        covariancia = somatorio / (n - 1)
    return covariancia
```

1.0.29 Coeficiente de Correlação Populacional de Pearson

Para dados correlacionados = $[[x_1, y_1], [x_2, y_2], [x_3, y_3], ...]$ ##

$$\rho_{xy} = \frac{\sigma_{xy}}{\sigma_x \sigma_y}$$

[30]:
```
def coeficiente_correlacao_populacional_pearson(dados):
    dadosx = []
    dadosy = []
    for i in dados:
        x = i[0]
        y = i[1]
        dadosx.append(x)
        dadosy.append(y)
    covariancia = covariancia_populacional(dados)
    desviox = desvio_padrao_populacional(dadosx)
    desvioy = desvio_padrao_populacional(dadosy)
    coeficiente = covariancia / (desviox * desvioy)
    return coeficiente
```

1.0.30 Coeficiente de Correlação Amostral de Pearson

Para dados correlacionados = $[[x_1, y_1], [x_2, y_2], [x_3, y_3], ...]$ ##

$$r_{xy} = \frac{s_{xy}}{s_x s_y}$$

[31]:
```
def coeficiente_correlacao_amostral_pearson(dados):
    dadosx = []
    dadosy = []
```

```
    for i in dados:
        x = i[0]
        y = i[1]
        dadosx.append(x)
        dadosy.append(y)
    covariancia = covariancia_amostral(dados)
    desviox = desvio_padrao_amostral(dadosx)
    desvioy = desvio_padrao_amostral(dadosy)
    coeficiente = covariancia / (desviox * desvioy)
    return coeficiente
```

1.0.31 Somatório dos Quadrados

Para dados não agrupados = $[x_1, x_2, x_3, ...]$ ##

$$SS_x = \sum_{i=1}^{n} {x_i}^2 - \frac{(\sum_{i=1}^{n} x_i)^2}{n}$$

[32]:
```
def somatorio_quadrados(dados):
    sq = 0
    s = 0
    somatorio = 0
    n = len(dados)
    for i in dados:
        sq = sq + pow(i, 2)
        s = s + i
    if n > 0:
        somatorio = sq - (pow(s, 2) / n)
    return somatorio
```

1.0.32 Somatório dos Produtos XY

Para dados correlacionados = $[[x_1, y_1], [x_2, y_2], [x_3, y_3], ...]$ ##

$$SS_{xy} = \sum_{i=1}^{n} x_i y_i - \frac{(\sum_{i=1}^{n} x_i)(\sum_{i=1}^{n} y_i)}{n}$$

[33]:
```
def somatorio_produtos(dados):
    sp = 0
    sx = 0
    sy = 0
    somatorio = 0
    n = len(dados)
    for i in dados:
        x = i[0]
        y = i[1]
```

```
        sp = sp + (x * y)
        sx = sx + x
        sy = sy + y
    if n > 0:
        somatorio = sp - ((sx * sy) / n)
    return somatorio
```

1.0.33 Coeficiente de Correlação de Pearson

Para dados correlacionados = $[[x_1, y_1], [x_2, y_2], [x_3, y_3], ...]$ ##

$$r = \frac{SS_{xy}}{\sqrt{SS_x \times SS_y}}$$

[34]:
```
def coeficiente_correlacao_pearson(dados):
    dadosx = []
    dadosy = []
    for i in dados:
        x = i[0]
        y = i[1]
        dadosx.append(x)
        dadosy.append(y)
    sp = somatorio_produtos(dados)
    sqx = somatorio_quadrados(dadosx)
    sqy = somatorio_quadrados(dadosy)
    coeficiente = sp / math.sqrt(sqx * sqy)
    return coeficiente
```

1.0.34 Z-score Populacional

Para dados não agrupados = $[x_1, x_2, x_3, ...]$ ##

$$z = \frac{x - \mu}{\sigma}$$

[35]:
```
def escore_z_populacional(x, dados):
    media = media_aritmetica(dados)
    desvio = desvio_padrao_populacional(dados)
    escore = (x - media) / desvio
    return escore
```

1.0.35 Z-score Amostral

Para dados não agrupados = $[x_1, x_2, x_3, ...]$ ##

$$z = \frac{x - \bar{x}}{s}$$

[36]:
```python
def escore_z_amostral(x, dados):
    media = media_aritmetica(dados)
    desvio = desvio_padrao_amostral(dados)
    escore = (x - media) / desvio
    return escore
```

1.0.36 Três Desvios

Para dados não agrupados = $[x_1, x_2, x_3, ...]$

[37]:
```python
def tres_desvios(dados, pop = True):
    media = media_aritmetica(dados)
    tresdesvios = {}
    if pop:
        desvio = desvio_padrao_populacional(dados)
    else:
        desvio = desvio_padrao_amostral(dados)
    tresdesvios['-3'] = media - (3 * desvio)
    tresdesvios['-2'] = media - (2 * desvio)
    tresdesvios['-1'] = media - desvio
    tresdesvios['+1'] = media + desvio
    tresdesvios['+2'] = media + (2 * desvio)
    tresdesvios['+3'] = media + (3 * desvio)
    return tresdesvios
```

1.0.37 Amplitude

Para dados não agrupados = $[x_1, x_2, x_3, ...]$

ou agrupados = $[[w_1, x_1], [w_2, x_2], [w_3, x_3], ...]$

[38]:
```python
def amplitude(dados):
    if isinstance(dados[0], list):
        dados = desagrupar_dados(dados)
    dados.sort()
    n = len(dados)
    amplitude = dados[n-1] - dados[0]
    return amplitude
```

1.0.38 Assimetria

Para dados não agrupados = $[x_1, x_2, x_3, ...]$ ##

$$A = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{x_i - \bar{x}}{s}\right)^3$$

[39]:
```python
def assimetria(dados):
    somatorio = 0
```

```
    n = len(dados)
    media = media_aritmetica(dados)
    desvio = desvio_padrao_amostral(dados)
    for x in dados:
        somatorio = somatorio + pow((x - media) / desvio, 3)
    assimetria = somatorio / n
    return assimetria
```

1.0.39 Curtose

Para dados não agrupados = $[x_1, x_2, x_3, ...]$ ##

$$K = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{x_i - \bar{x}}{s}\right)^4 - 3$$

[40]:
```
def curtose(dados):
    somatorio = 0
    n = len(dados)
    media = media_aritmetica(dados)
    desvio = desvio_padrao_amostral(dados)
    for x in dados:
        somatorio = somatorio + pow((x - media) / desvio, 4)
    curtose = (somatorio / n) - 3
    return curtose
```

1.0.40 Quartis

Para dados não agrupados = $[x_1, x_2, x_3, ...]$ ##

$$i = \frac{j(n+1)}{4}$$

##

$$Q_j = x_i + \left(\frac{j(n+1)}{4} - i\right)(x_{i+1} - x_i)$$

para j = 1, 2 e 3

[41]:
```
def quartis(dados):
    quartis = []
    dados.sort()
    n = len(dados)
    for j in range(3):
        k = ((j + 1) * (n + 1)) / 4
        i = math.floor(k)
        q = dados[i-1] + ((k - i) * (dados[i] - dados[i-1]))
        quartis.append(q)
    return quartis
```

1.0.41 Desagrupar dados

Para dados agrupados $= [[w_1, x_1], [w_2, x_2], [w_3, x_3], ...]$

[42]:
```
def desagrupar_dados(dados):
    dadosdesagrupados = []
    for i in dados:
        w = i[0]
        x = i[1]
        for j in range(w):
            dadosdesagrupados.append(x)
    return dadosdesagrupados
```

1.0.42 Agrupar dados

Para dados não agrupados $= [x_1, x_2, x_3, ...]$

[43]:
```
def agrupar_dados(dados):
    dadosagrupados = []
    dados.sort()
    n = len(dados)
    i = 0
    while i < n:
        x = dados[i]
        w = 1
        j = i + 1
        while j < n:
            if x == dados[j]:
                w = w + 1
                j = j + 1
            else:
                break
        dadosagrupados.append([w, x])
        i = j
    return dadosagrupados
```

2 Demonstração dos cálculos estatísticos

(*) funções nativas

[44]:
```
# dados para exemplo

dados_nao_agrupados = [1, 2, 2, 3, 3, 5, 6, 6, 6, 6, 7, 8, 8, 8, 9]

dados_agrupados = [[1, 1], [2, 2], [2, 3], [1, 5], [4, 6], [1, 7], [3, 8], [1,␣
↪9]]
```

```
dados_correlacionados_agrupados = [[1, 9], [2, 10], [3, 11], [4, 12], [5, 13],␣
 ↪[6, 14], [7, 15], [8, 16]]

dados_correlacionados_a = [1, 2, 3, 4, 5, 6, 7, 8]
dados_correlacionados_b = [9, 10, 11, 12, 13, 14, 15, 16]

x = 2

df_nao_agrupado = pd.DataFrame (dados_nao_agrupados, columns = ['x'])

df_agrupado = pd.DataFrame (dados_agrupados, columns = ['w', 'x'])
```

[45]:
```
# execuções das funções
# (*) funções nativas

print(f'Média Aritmética: {media_aritmetica(dados_nao_agrupados)}')
print(f'Média Aritmética (*): {statistics.mean(dados_nao_agrupados)}')
print(f"Média Aritmética (*): {df_nao_agrupado['x'].mean()}\n")

print(f'Média Aritmética Ponderada:␣
 ↪{media_aritmetica_ponderada(dados_agrupados)}')

print(f'Média Geométrica: {media_geometrica(dados_nao_agrupados)}')

print(f'Média Geométrica Ponderada:␣
 ↪{media_geometrica_ponderada(dados_agrupados)}')

print(f'Média Harmônica: {media_harmonica(dados_nao_agrupados)}')

print(f'Média Harmômica Ponderada:␣
 ↪{media_harmonica_ponderada(dados_agrupados)}')

print(f'Média Quadrática: {media_quadratica(dados_nao_agrupados)}')

print(f'Média Quadrática Ponderada:␣
 ↪{media_quadratica_ponderada(dados_agrupados)}')

print(f'Média Cúbica: {media_cubica(dados_nao_agrupados)}')

print(f'Média Cúbica Ponderada: {media_cubica_ponderada(dados_agrupados)}')

print(f'Média Desarmônica: {media_desarmonica(dados_nao_agrupados)}')

print(f'Média Desarmônica Ponderada:␣
 ↪{media_desarmonica_ponderada(dados_agrupados)}\n')
```

```
print(f'Mediana: {mediana(dados_nao_agrupados)}')
print(f'Mediana (*): {statistics.median(dados_nao_agrupados)}')
print(f"Mediana (*): {df_nao_agrupado['x'].median()}\n")

print(f'Moda: {moda(dados_nao_agrupados)}')
print(f'Moda (*): {statistics.mode(dados_nao_agrupados)}')
print(f"Moda (*): {df_nao_agrupado['x'].mode()}\n")

print(f'Desvio Absoluto Médio: {desvio_medio(dados_nao_agrupados)}')
print(f'Desvio Absoluto Mediano: {desvio_mediano(dados_nao_agrupados)}\n')

print(f'Variância Populacional: {variancia_populacional(dados_nao_agrupados)}')
print(f'Variância Populacional (*): {statistics.
 ↪pvariance(dados_nao_agrupados)}\n')

print(f'Desvio Padrão Populacional:␣
 ↪{desvio_padrao_populacional(dados_nao_agrupados)}')
print(f'Desvio Padrão Populacional (*): {statistics.
 ↪pstdev(dados_nao_agrupados)}\n')

print(f'Variância Amostral: {variancia_amostral(dados_nao_agrupados)}')
print(f'Variância Amostral (*): {statistics.variance(dados_nao_agrupados)}')
print(f"Variância Amostral (*): {df_nao_agrupado['x'].var()}\n")

print(f'Desvio Padrão Amostral: {desvio_padrao_amostral(dados_nao_agrupados)}')
print(f'Desvio Padrão Amostral (*): {statistics.stdev(dados_nao_agrupados)}')
print(f"Desvio Padrão Amostral (*): {df_nao_agrupado['x'].std()}\n")

print(f'Variância Populacional (agrupado):␣
 ↪{variancia_populacional_agrupado(dados_agrupados)}')

print(f'Desvio Padrão Populacional (agrupado):␣
 ↪{desvio_padrao_populacional_agrupado(dados_agrupados)}')

print(f'Variância Amostral (agrupado):␣
 ↪{variancia_amostral_agrupado(dados_agrupados)}')

print(f'Desvio Padrão Amostral (agrupado):␣
 ↪{desvio_padrao_amostral_agrupado(dados_agrupados)}\n')

print(f'Coeficiente de Variação: {coeficiente_variacao(dados_nao_agrupados)}')

print(f'Coeficiente de Variação (agrupado):␣
 ↪{coeficiente_variacao_agrupado(dados_agrupados)}\n')
```

```
print(f'Covariância Populacional:␣
 ↪{covariancia_populacional(dados_correlacionados_agrupados)}')

print(f'Covariância Amostral:␣
 ↪{covariancia_amostral(dados_correlacionados_agrupados)}\n')

print(f'Coeficiente de Correlação Populacional de Pearson:␣
 ↪{coeficiente_correlacao_populacional_pearson(dados_correlacionados_agrupados)}')

print(f'Coeficiente de Correlação Amostral de Pearson:␣
 ↪{coeficiente_correlacao_amostral_pearson(dados_correlacionados_agrupados)}\n')

print(f'Somatório dos Quadrados: {somatorio_quadrados(dados_nao_agrupados)}')

print(f'Somatório dos Produtos:␣
 ↪{somatorio_produtos(dados_correlacionados_agrupados)}\n')

print(f'Coeficiente de Correlação de Pearson:␣
 ↪{coeficiente_correlacao_pearson(dados_correlacionados_agrupados)}\n')

print(f'Escore Z Populacional: {escore_z_populacional(x, dados_nao_agrupados)}')

print(f'Escore Z Amostral: {escore_z_amostral(x, dados_nao_agrupados)}\n')

print(f'Três Desvios: {tres_desvios(dados_nao_agrupados, True)}\n')

print(f'Amplitude (dados não agrupados): {amplitude(dados_nao_agrupados)}')
print(f'Amplitude (dados agrupados): {amplitude(dados_agrupados)}\n')

print(f'Assimetria: {assimetria(dados_nao_agrupados)}')

print(f'Curtose: {curtose(dados_nao_agrupados)}')

print(f'Quartis: {quartis(dados_nao_agrupados)}\n')

print(f'Desagrupar dados: {desagrupar_dados(dados_agrupados)}')

print(f'Agrupar dados: {agrupar_dados(dados_nao_agrupados)}')
```

```
Média Aritmética: 5.333333333333333
Média Aritmética (*): 5.333333333333333
Média Aritmética (*): 5.333333333333333

Média Aritmética Ponderada: 5.333333333333333
Média Geométrica: 4.554414300660055
Média Geométrica Ponderada: 4.554414300660055
Média Harmônica: 3.603775383735343
```

```
Média Harmômica Ponderada: 3.603775383735343
Média Quadrática: 5.876506898943736
Média Quadrática Ponderada: 5.876506898943736
Média Cúbica: 6.253349315923748
Média Cúbica Ponderada: 6.253349315923748
Média Desarmônica: 6.36546904484207
Média Desarmônica Ponderada: 6.36546904484207

Mediana: 6
Mediana (*): 6
Mediana (*): 6.0

Moda: [6]
Moda (*): 6
Moda (*): 0    6
dtype: int64

Desvio Absoluto Médio: 2.1333333333333337
Desvio Absoluto Mediano: 2

Variância Populacional: 6.088888888888889
Variância Populacional (*): 6.088888888888889

Desvio Padrão Populacional: 2.4675674031095665
Desvio Padrão Populacional (*): 2.4675674031095665

Variância Amostral: 6.523809523809524
Variância Amostral (*): 6.523809523809524
Variância Amostral (*): 6.523809523809524

Desvio Padrão Amostral: 2.554174920362645
Desvio Padrão Amostral (*): 2.554174920362645
Desvio Padrão Amostral (*): 2.554174920362645

Variância Populacional (agrupado): 6.088888888888889
Desvio Padrão Populacional (agrupado): 2.4675674031095665
Variância Amostral (agrupado): 6.523809523809524
Desvio Padrão Amostral (agrupado): 2.554174920362645

Coeficiente de Variação: 0.46266888808304374
Coeficiente de Variação (agrupado): 0.46266888808304374

Covariância Populacional: 5.25
Covariância Amostral: 6.0

Coeficiente de Correlação Populacional de Pearson: 1.0
Coeficiente de Correlação Amostral de Pearson: 1.0000000000000002
```

```
Somatório dos Quadrados: 91.33333333333331
Somatório dos Produtos: 42.0

Coeficiente de Correlação de Pearson: 1.0

Escore Z Populacional: -1.3508580673957478
Escore Z Amostral: -1.3050528790174099

Três Desvios: {'-3': -2.069368875995367, '-2': 0.3981985271142001, '-1':
2.8657659302237666, '+1': 7.8009007364429, '+2': 10.268468139552466, '+3':
12.736035542662034}

Amplitude (dados não agrupados): 8
Amplitude (dados agrupados): 8

Assimetria: -0.27561700705282527
Curtose: -1.4339353899159955
Quartis: [3.0, 6.0, 8.0]

Desagrupar dados: [1, 2, 2, 3, 3, 5, 6, 6, 6, 6, 7, 8, 8, 8, 9]
Agrupar dados: [[1, 1], [2, 2], [2, 3], [1, 5], [4, 6], [1, 7], [3, 8], [1, 9]]
```