Estatística_III

May 7, 2025

1 Aula Estatística III - Tipos de Estatística

[1]:
```
# Criamos um vetor

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

freq = np.array([5,7,5,9,7,7,6,9,9,9,10,12,12,7])
```

[2]:
```
freq
```

[2]:
```
array([ 5,  7,  5,  9,  7,  7,  6,  9,  9,  9, 10, 12, 12,  7])
```

1.1 Frequencia Absoluta: f_i

Chamamos de frequência absoluta o número de vezes que um mesmo dado apareceu dentro de um conjunto.

[3]:
```
# valores únicos e suas contagens
valores, contagens = np.unique(freq, return_counts=True)
```

[4]:
```
# mensagem de versão descontinuada
pd.value_counts(freq, sort=False).sort_index()
```

```
/var/folders/yq/pq2twqh913s0mr445r3_jz7h0000gn/T/ipykernel_64514/3918607239.py:2
: FutureWarning: pandas.value_counts is deprecated and will be removed in a
future version. Use pd.Series(obj).value_counts() instead.
  pd.value_counts(freq, sort=False).sort_index()
```

[4]:
```
5     2
6     1
7     4
9     4
10    1
12    2
Name: count, dtype: int64
```

[5]:
```
pd.Series(freq).value_counts().sort_index()
```

[5]:
```
5     2
6     1
7     4
9     4
10    1
12    2
Name: count, dtype: int64
```

1.2 Frequencia Relativa: n e $f_r\%$

A frequência relativa de um valor em um conjunto de dados é a proporção de vezes que ele aparece em relação ao total de observações

[6]:
```
# valor / qtd e (valor / qtd)*100
contagens / len(freq)
```

[6]:
```
array([0.14285714, 0.07142857, 0.28571429, 0.28571429, 0.07142857,
       0.14285714])
```

[7]:
```
2/14
```

[7]:
```
0.14285714285714285
```

1.3 Frequencia Acumulada

A frequência acumulada de um valor x_i é a soma progressiva das frequências absolutas até um determinado valor. Ela mostra quantos elementos existem até aquele ponto na distribuição.

$fa(x_i) = \sum_{j=1}^{i} f(x_j)$

Onde: - $fa(x_i)$ é a frequência acumulada até o valor x_i - $f(x_j)$ é a frequência absoluta do valor x_j

[8]:
```
np.cumsum(contagens)
```

[8]:
```
array([ 2,  3,  7, 11, 12, 14])
```

1.4 Criando a tabela de frequencia

[9]:
```
# Criando DataFrame

# Frequência absoluta
valores, contagens = np.unique(freq, return_counts=True)

# Frequência relativa
frequencia_relativa = contagens / len(freq)

# Frequência acumulada
```

```
frequencia_acumulada = np.cumsum(contagens)

tabela_frequencia = pd.DataFrame({
    'Valor': valores,
    'Frequência (f)': contagens,
    'Frequência Relativa (fr)': np.round(frequencia_relativa, 3),
    'Frequência Acumulada (fa)': frequencia_acumulada
})

# Exibir a tabela
print(tabela_frequencia)
```

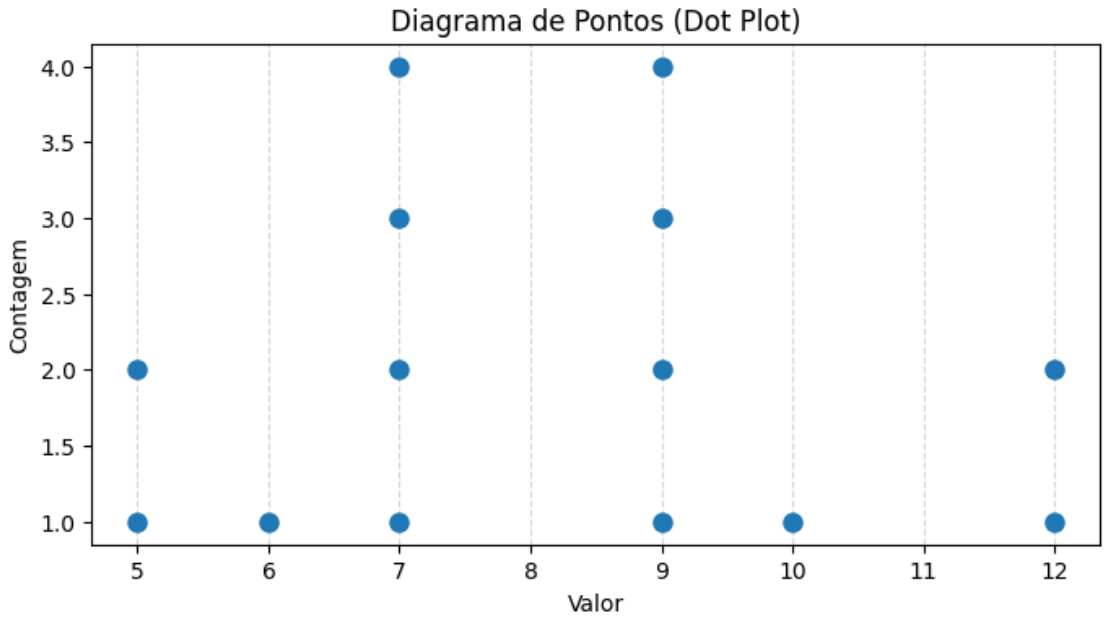
	Valor	Frequência (f)	Frequência Relativa (fr)	Frequência Acumulada (fa)
0	5	2	0.143	2
1	6	1	0.071	3
2	7	4	0.286	7
3	9	4	0.286	11
4	10	1	0.071	12
5	12	2	0.143	14

1.5 Diagrama de Pontos

[10]:
```
# Gerando coordenadas para o gráfico de pontos
x = []
y = []

for valor, contagem in zip(valores, contagens):
    x.extend([valor] * contagem)
    y.extend(list(range(1, contagem + 1)))

# Plotando
plt.figure(figsize=(8, 4))
plt.plot(x, y, 'o', markersize=8)
plt.xlabel("Valor")
plt.ylabel("Contagem")
plt.title("Diagrama de Pontos (Dot Plot)")
plt.grid(True, axis='x', linestyle='--', alpha=0.5)
plt.show()
```



1.6 Questions...

Qual a amplitude dos nossos dados?

Quantos alunos tem mais que 9 anos?

Qual idade possui maior frequência?

[11]:
```
amplitude = np.max(freq) - np.min(freq)
print("Amplitude:", amplitude)
```

```
Amplitude: 7
```

[12]:
```
# significa peak to peak, ou seja, máximo - mínimo.

amplitude = np.ptp(freq)  # resultado: 12 - 5 = 7
print("Amplitude:", amplitude)
```

```
Amplitude: 7
```

[13]:
```
# > 9 anos

alunos_mais_que_9 = freq[freq > 9]
print('Alunos com mais de 9 anos: ',alunos_mais_que_9)
print(len(alunos_mais_que_9))
```

```
Alunos com mais de 9 anos:  [10 12 12]
3
```

[14]:
```
freq[freq > 9]
```

[14]:
```
array([10, 12, 12])
```

1.7 Idade com maior frequência (Moda?)

[15]:
```
contagens
```

[15]:
```
array([2, 1, 4, 4, 1, 2])
```

[16]:
```
# Determinar a frequência máxima
frequencia_max = np.max(contagens)

# Identificar todas as modas
modas = valores[contagens == frequencia_max]

print(f"As modas são: {modas}, cada uma com {frequencia_max} ocorrências.")
```

```
As modas são: [7 9], cada uma com 4 ocorrências.
```

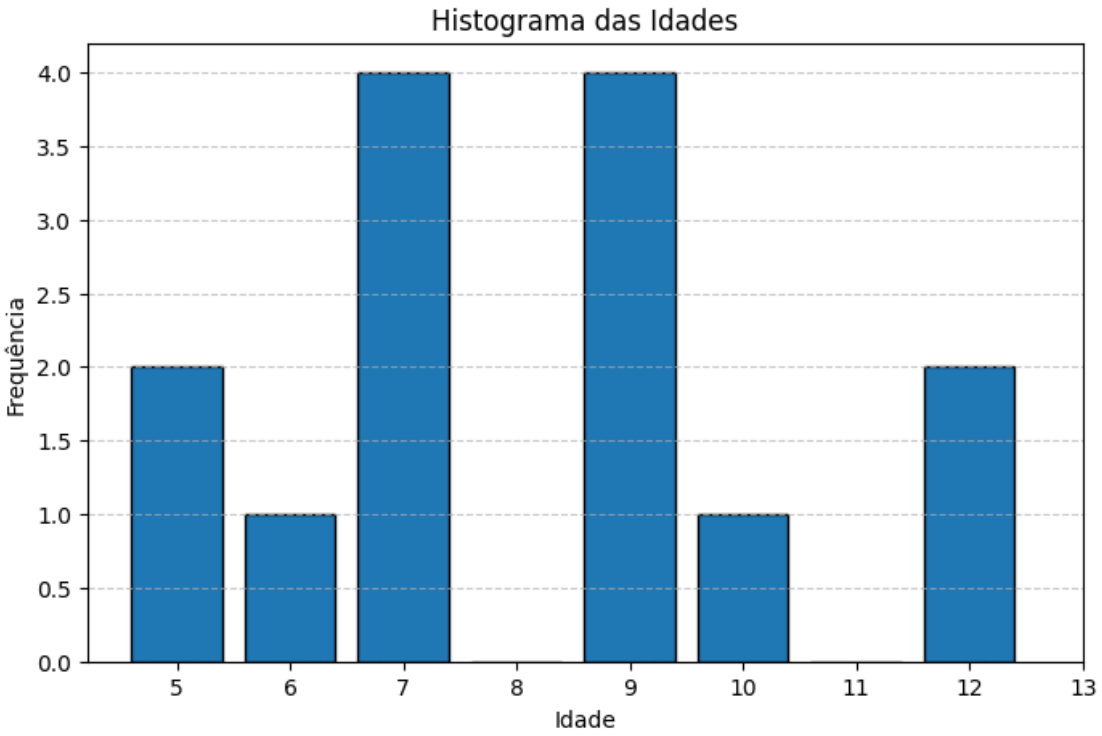
[17]:
```
np.max(contagens)
```

[17]:
```
np.int64(4)
```

1.8 Representação em um Histograma

Esse tipo de gráfico é amplamente utilizado para enterndermos a distribuição das amostras em um espaço de características, auxiliando na etapa de tomada de decisões.

[18]:
```
# Criando o histograma
plt.figure(figsize=(8, 5))
plt.hist(freq, bins=range(5, 14), edgecolor='black', align='left', rwidth=0.8)
plt.title("Histograma das Idades")
plt.xlabel("Idade")
plt.ylabel("Frequência")
plt.xticks(range(5, 14))  # Garantir que todas as idades apareçam no eixo x
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```



1.9 Vamos analisar esse outro caso:

Imagine que em um determinado dia da semana, o dono de uma pizzaria resolve coletar a idades de todas as pessoas que chegam em seu restaurante.

Dado o seguinte vetor de idades abaixo, quais seriam os métodos adotados para construção da análise para o proprietário?

[19]:
```
idades = np.array([1,3,27,32,5,63,26,25,18,16,4,45,29,19,22,51,58,9,42,6])
len(idades)
```

[19]: 20

[20]:
```
# Existe repetições?

pd.Series(idades).value_counts().sort_index()
```

[20]:
```
1     1
3     1
4     1
5     1
6     1
9     1
16    1
```

```
18    1
19    1
22    1
25    1
26    1
27    1
29    1
32    1
42    1
45    1
51    1
58    1
63    1
Name: count, dtype: int64
```

1.10 Criação de Classes (Intervalos)

Para construir histogramas ou distribuir dados em classes, algumas regras empíricas podem ser utilizadas para definir o número ideal de intervalos (k):

1.10.1 Regra de Sturges

$k = 1 + \log_2(n)$

Onde:

- (k): número de classes
- (n): número de dados no conjunto

Observação: Arredonde (k) para o número inteiro mais próximo.

1.10.2 Regra da Raiz Quadrada

$k = \sqrt{n}$

Simples de aplicar e útil em análises exploratórias. Também deve-se arredondar o resultado.

1.10.3 Amplitude do Intervalo (h)

Uma vez determinado o número de classes, a amplitude de cada intervalo pode ser calculada por:

$h = \frac{\text{Valor máximo} - \text{Valor mínimo}}{k}$

Dicas práticas:

- Prefira amplitudes com números arredondados (como 5, 10, 20).
- Adapte os limites inferiores e superiores para que todos os dados estejam contidos nos intervalos.
- As classes devem ser mutuamente exclusivas e exaustivas.

1.11 Conhecendo os dados (amplitude e elementos)

[21]:
```
n = len(idades)                          # Total de dados
min_idade = np.min(idades)              # Mínimo
max_idade = np.max(idades)              # Máximo
amplitude_total = max_idade - min_idade
```

[22]:
```
np.ptp(idades)
```

[22]: np.int64(62)

[23]:
```
# A função np.ceil() significa "ceiling" (teto, em inglês).
# Ela arredonda um número para cima, para o menor inteiro que é maior ou igual␣
 ↪ao número dado.

np.ceil(4.2)
np.floor(4.2)
np.round(4.2)
```

[23]: np.float64(4.0)

[24]:
```
# Regra de Sturges

k_sturges = int(np.ceil(1 + np.log2(n)))
h_sturges = int(np.ceil(amplitude_total / k_sturges))
print(f" O número de classes: {k_sturges} e valor da amplitude: {h_sturges}")
```

```
 O número de classes: 6 e valor da amplitude: 11
```

[ ]:

[25]:
```
np.ceil(4.3)
```

[25]: np.float64(5.0)

[26]:
```
# Regra da Raiz Quadrada

k_raiz = int(np.ceil(np.sqrt(n)))
h_raiz = int(np.ceil(amplitude_total / k_raiz))
print(f" O número de classes: {k_raiz} e valor da amplitude: {h_raiz}")
```

```
 O número de classes: 5 e valor da amplitude: 13
```

[27]:
```
# Regra de Sturges

bins_sturges = np.arange(min_idade, max_idade + h_sturges, h_sturges)
labels_sturges = [f'{int(bins_sturges[i])}-{int(bins_sturges[i+1])-1}' for i in␣
 ↪range(len(bins_sturges)-1)]
```

```
faixas_sturges = pd.cut(idades, bins=bins_sturges, labels=labels_sturges,␣
 ↪right=False)
distribuicao_sturges = faixas_sturges.value_counts().sort_index().
 ↪reset_index(name='Frequência')
distribuicao_sturges.rename(columns={'index': 'Faixa Etária'}, inplace=True)

# Regra da Raiz Quadrada

bins_raiz = np.arange(min_idade, max_idade + h_raiz, h_raiz)
labels_raiz = [f'{int(bins_raiz[i])}-{int(bins_raiz[i+1])-1}' for i in␣
 ↪range(len(bins_raiz)-1)]
faixas_raiz = pd.cut(idades, bins=bins_raiz, labels=labels_raiz, right=False)
distribuicao_raiz = faixas_raiz.value_counts().sort_index().
 ↪reset_index(name='Frequência')
distribuicao_raiz.rename(columns={'index': 'Faixa Etária'}, inplace=True)

# Exibindo os resultados
print("Distribuição - Regra de Sturges:")
print(distribuicao_sturges)

print("\nDistribuição - Regra da Raiz Quadrada:")
print(distribuicao_raiz)
```

```
Distribuição - Regra de Sturges:
  Faixa Etária  Frequência
0         1-11           6
1        12-22           4
2        23-33           5
3        34-44           1
4        45-55           2
5        56-66           2

Distribuição - Regra da Raiz Quadrada:
  Faixa Etária  Frequência
0         1-13           6
1        14-26           6
2        27-39           3
3        40-52           3
4        53-65           2
```

[ ]:

1.12 Agrupando manualmente

[28]:
```
# Intervalo desejado de 10 em 10
intervalo = 10

# Valor mínimo e máximo
min_idade = np.min(idades)
max_idade = np.max(idades)

# Criando os bins automaticamente com base no intervalo
bins = np.arange(start=(min_idade // intervalo) * intervalo, # garante que o␣
 ↪primeiro intervalo comece em múltiplo de 10
                 stop=((max_idade // intervalo) + 1) * intervalo + intervalo,␣
 ↪#garante que o último intervalo seja grande o suficiente para incluir o␣
 ↪maior valor
                 step=intervalo)

# Criando rótulos para as faixas
labels = [f"{bins[i]}-{bins[i+1]-1}" for i in range(len(bins) - 1)]

# Classificando as idades
faixas = pd.cut(idades, bins=bins, labels=labels, right=True,␣
 ↪include_lowest=True) # Função que segmenta os dados em intervalos definidos.

# Distribuição por faixa
distribuicao = faixas.value_counts().sort_index()

# Resultado
print(distribuicao)
```

```
0-9      6
10-19    3
20-29    5
30-39    1
40-49    2
50-59    2
60-69    1
Name: count, dtype: int64
```

[29]:
```
bins = np.arange(0, 80, 10)
bins
```

[29]:
```
array([ 0, 10, 20, 30, 40, 50, 60, 70])
```

[30]:
```
# Agora precisamos separar os vetores

faixas
```

[30]:
```
['0-9', '0-9', '20-29', '30-39', '0-9', …, '50-59', '50-59', '0-9', '40-49',
'0-9']
Length: 20
Categories (7, object): ['0-9' < '10-19' < '20-29' < '30-39' < '40-49' < '50-59'
< '60-69']
```

[31]:
```
# Separa faixas e frequências

faixas = distribuicao.index.astype(str).tolist()
valores = distribuicao.values
```

[ ]:
```
```

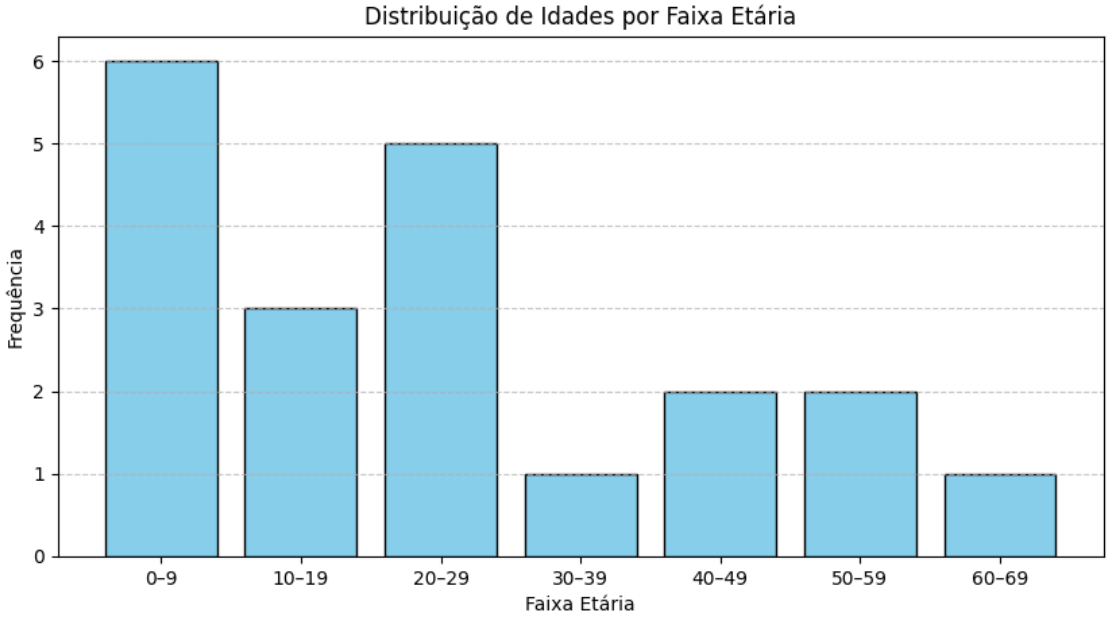
[32]:
```
valores
```

[32]:
```
array([6, 3, 5, 1, 2, 2, 1])
```

[ ]:
```
```

[33]:
```
# Plotando gráfico de barras

plt.figure(figsize=(10, 5))
plt.bar(faixas, valores, color='skyblue', edgecolor='black')
plt.xlabel("Faixa Etária")
plt.ylabel("Frequência")
plt.title("Distribuição de Idades por Faixa Etária")
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```



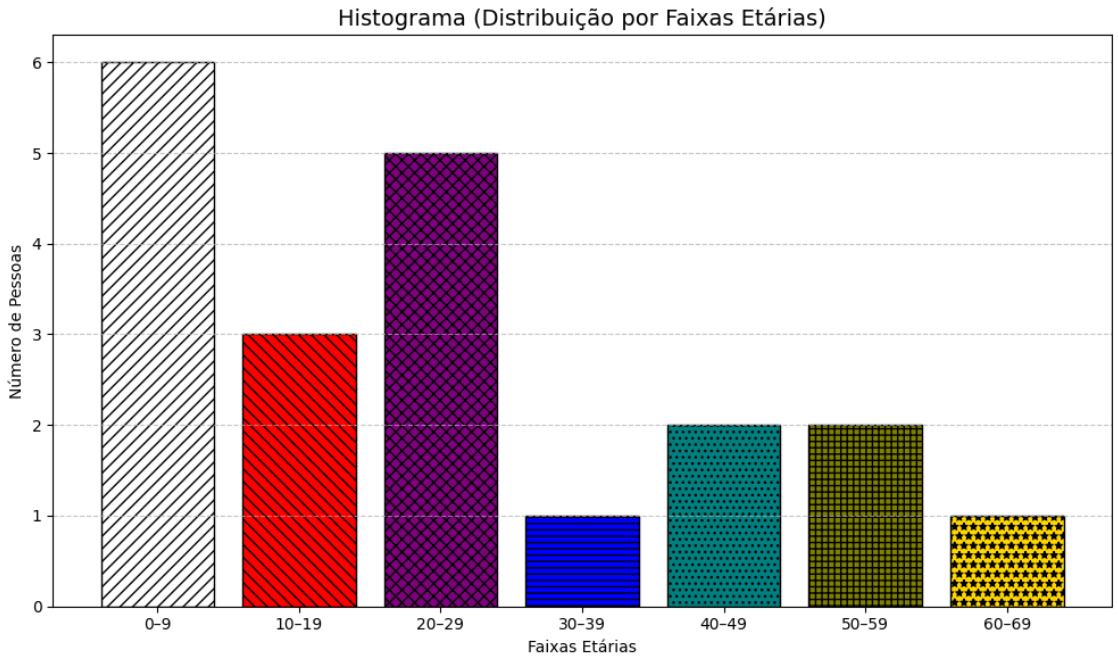
[34]:
```
import matplotlib.pyplot as plt

# Plotando o histograma estilizado
plt.figure(figsize=(10, 6))
bars = plt.bar(faixas, valores, edgecolor='black')

# Adicionando hachuras e cores variadas como na imagem
patterns = ['///', '\\\\\\', 'xxx', '---', '...', '+++', '**']
colors = ['white', 'red', 'purple', 'blue', 'teal', 'olive', 'gold']

for bar, pattern, color in zip(bars, patterns, colors):
    bar.set_hatch(pattern)
    bar.set_facecolor(color)

plt.title("Histograma (Distribuição por Faixas Etárias)", fontsize=14)
plt.xlabel("Faixas Etárias")
plt.ylabel("Número de Pessoas")
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```



1.13 Olhando para o Histograma, quais conclusões poderíamos tirar agora?

2 Exercícios

2.0.1 Os dados a seguir representam o numero de cliente, por dia, no restaurante Bom Prato desde sua inalguração:

96, 279, 255, 254, 75, 211, 271 e 291. **Utilizando esses dados, crie o histograma e apresente a interpretação dos resultados**

3 Estatística Descritiva

`1 - Média`

- A **média aritmética** é a medida de tendência central mais conhecida. Ela representa um valor "típico" ou "central" de um conjunto de dados numéricos.
- É calculada somando todos os valores e dividindo pelo número de observações.

  ATENÇÃO: A média é sensível a valores extremos (outliers).

Fórmula::

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

`2 - Mediana`

- A **mediana** é o valor central de um conjunto ordenado de dados. Ela é resistente a outliers, o que a torna útil quando há valores extremos.

Fórmula:

$$\text{Mediana} = x_{(\frac{n+1}{2})}$$

`3 - Moda`

- Valor que mais se repete na amostra.

Fórmula:

$$\text{Moda} = \text{valor mais frequente em } \{x_1, x_2, \dots, x_n\}$$

`4 - Desvio Padrão`

- Medida que expressa o grau de dispersão de um conjunto de dados. Ou seja, o **desvio padrão** indica o quanto um conjunto de dados é uniforme.
- Quanto mais próximo de 0 for o desvio padrão, mais homogêneo são os dados.

Fórmula:

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

`5 - Quartis`

- Q_1: Valor que serpara os **25%** menores dados.
- Q_2: mediana.

- Q_3: Valor que serpara os **75%** menores dados.

  6 - Outliers com Intervalo Interquartil (IQR)

- IQR: Q_3 - Q_1
- **Limite Inferior:** (Q_1 - 1.5) * IQR
- **Limite Superior:** (Q_3 + 1.5) * IQR

3.1 Exemplos:

[35]:
```
# Vamos utilizar o seguinte conjunto de dados para todos os exemplos:

import numpy as np
from scipy import stats

dados = [10, 12, 23, 23, 16, 23, 21, 16]
```

3.2 Boxplot

- O boxplot (ou diagrama de caixa) é uma representação gráfica da distribuição estatística de um conjunto de dados, focando especialmente em medidas de posição e dispersão.

3.3 Boxplot Violino

- É útil quando se deseja visualizar distribuições assimétricas ou multimodais, além da posição central.

[ ]:
```

```