Aula 8B - Estatística Descritiva

Gustavo Oliveira e Andréa Rocha

Departamento de Computação Científica / UFPB

Julho de 2020

1 Estatística Descritiva com apoio do *Python*

1.1 As bibliotecas *numpy* e *pandas*

- Vamos apresentar vários métodos estatísticos desenvolvidos para *Series* e *DataFrames*.

[64]:
```
import numpy as np
import pandas as pd
```

1.2 Distribuição de Frequência

- Uma distribuição de frequência é uma tabela que contém um resumo dos dados obtido em uma amostra.
- A distribuição é organizada em formato de tabela, e cada entrada da tabela contém a frequência dos dados em um determinado intervalo, ou em um grupo.
- Abaixo vemos um exemplo simplificado de tabela de distribuição de frequência:

Alturas em metros	Número dos Alunos
1,50 ⊢ 1,60	5
1,60 ⊢ 1,70	15
1,70 ⊢ 1,80	17
1,80 ⊢ 1,90	3
Total	40

1.2.1 Construção de uma distribuição de frequência

- Para ilustrar como se constrói uma distribuição de frequência, nós vamos considerar um exemplo específico.
- Assim, suponha que uma pesquisa foi feita, e o seguinte conjunto de dados foi obtido:
  - **Dados Brutos**: 24-23-22-28-35-21-23-33-34-24-21-25-36-26-22-30-32-25-26-33-34-21-31-25-31-26-25-35-33-31.

[65]:
```
dados =␣
↪[24,23,22,28,35,21,23,33,34,24,21,25,36,26,22,30,32,25,26,33,34,21,31,25,31,26,25,35,33,31]
```

Rol de dados

- A primeira coisa que fazemos é ordenar os dados do menor para o maior, formando o *rol de dados*:
  - **Rol de dados**: 21-21-21-22-22-23-23-24-25-25-25-25-26-26-26-28-30-31-31-31-32-33-33-33-34-34-34-35-35-36.

[66]:
```
np.sort(dados)
```

[66]:
```
array([21, 21, 21, 22, 22, 23, 23, 24, 24, 25, 25, 25, 25, 26, 26, 26, 28,
       30, 31, 31, 31, 32, 33, 33, 33, 34, 34, 35, 35, 36])
```

Amplitude Total

- Em seguida, calculamos a *amplitude total*, ou seja, o maior valor obtido na amostra subtraído do menor valor obtido na amostra:
  - **Amplitude Total**: R = 36-21 = 15.

[67]:
```
R = np.max(dados) - np.min(dados); R
```

[67]:
```
15
```

Tamanho Amostral

- Vamos calcular, agora, o tamanho amostral, ou seja, o número de observações obtidas na amostra.
  - **Tamanho Amostral**: n = 30.

[68]:
```
n = len(dados); n
```

[68]:
```
30
```

- Para Series e DataFrames o método **count()** retorna a total de valores.

[69]:
```
n = pd.Series(dados).count(); n
```

[69]:
```
30
```

Número de Classes

- Queremos, agora, dividir a amostra em uma quantidade de grupos que formarão os intervalos. Cada grupo é chamado de *classe*, assim, queremos definir o *número de classes* a ser considerado na tabela de *distribuição de frequência*:
  - **Número de Classes**: K.
    - K=5 para $n \leq 25$ e $K \approx \sqrt{n}$, para $n > 25$.
    - Fórmula de Sturges $K \approx 1 + 3,22 \log n$.

- Logo, pela primeira regra temos $K = \sqrt{30} \approx 5,48 \approx 6$, e pela segunda regra $K \approx 1 + 3,22 \log 30 \approx 5,75 \approx 6$. Desta forma, em ambos os casos temos $K = 6$, que será o valor considerado.

Número de classes padrão:

[70]:
```
if n<25:
    K = 5
else:
    K = np.ceil(np.sqrt(n))
K
```

[70]: 6.0

Número de classes fórmula de Sturges:

[71]:
```
KFS = np.ceil( 1 + 3.22*np.log10(n)); KFS
```

[71]: 6.0

Amplitude das Classes

- O próximo passo é saber o comprimento de cada intervalo a ser considerado, ou seja, calcular a amplitude de cada classe. Queremos que todas as classes tenham a mesma amplitude e portanto, temos:
  - **Amplitude das Classes**: $h = \frac{R}{K} = \frac{15}{6} = 2,5 \approx 3$.

[72]:
```
h = np.ceil(R/K); h
```

[72]: 3.0

Limites das Classes

- Vamos agora definir os *limites das classes*. Para tanto, começamos com o menor valor obtido da amostra, ou equivalentemente, o primeiro valor do *rol de dados*, e vamos somando a amplitude para definir cada limite de intervalo:

Classes
$21 \vdash 24$
$24 \vdash 27$
$27 \vdash 30$
$30 \vdash 33$
$33 \vdash 36$
$36 \vdash 39$

[73]:
```
bins = [np.min(dados) + i*h.astype('int') for i in range(K.astype('int')+1)];␣
↪bins
```

[73]:
```
[21, 24, 27, 30, 33, 36, 39]
```

Frequência dos Dados

- Agora, calculamos as frequências dos dados em cada intervalo e, chamada também de *frequência absoluta*. E finalmente montamos a tabela de *Distribuição de Frequência*.

Classes	Frequência
21 ⊢ 24	7
24 ⊢ 27	9
27 ⊢ 30	1
30 ⊢ 33	5
33 ⊢ 36	7
36 ⊢ 39	1

No *pandas*, a função **cut** cria classes a partir dos dados e o método **value_counts()** cria uma tabela de frequências. Combinando os dois obtemos uma *Distribuição de Frequência*.

[74]:
```
pd.cut(dados, bins=bins, right=False).value_counts()
```

[74]:
```
[21, 24)    7
[24, 27)    9
[27, 30)    1
[30, 33)    5
[33, 36)    7
[36, 39)    1
dtype: int64
```

[75]:
```
def n_classes(dados: pd.Series, tipo='Padrão'):
    n_obs = len(dados)
    if tipo=='Padrão':
        return 5 if n_obs<25 else np.ceil(np.sqrt(n_obs)).astype(int)
    if tipo=='Sturges':
        return (1 + np.ceil(np.log2(n_obs))).astype(int)

def amplitude_classes(dados: pd.Series, tipo='Padrão', arredondar=True):
    amplitude = np.ceil((dados.max() - dados.min())/n_classes(dados, tipo=tipo))␣
↪\
    if arredondar else (dados.max() - dados.min())/n_classes(dados, tipo=tipo)
    return amplitude

def construir_tabela(dados, tipo='Padrão', direita=False, arredondar=True):
    dados_series = pd.Series(dados)
```

```
    n_class = n_classes(dados_series, tipo=tipo)
    amp_class = amplitude_classes(dados_series, tipo=tipo, arredondar=arredondar)
    bins = [dados_series.min() + i*amp_class for i in range(n_class+1)]
    return pd.cut(dados_series, bins=bins, right=direita).
 ↪value_counts(sort=False).rename('Frequência')

def formatar_intervalos(intervalos, prec, direita=False):
    fechado = 'right' if direita else 'left'
    return [pd.Interval(left=np.round(intervalo.left,prec),
                        right=np.round(intervalo.right,prec), closed=fechado)␣
 ↪for intervalo in intervalos]
```

[76]:
```
def dist_freq(dados, tipo='Padrão', prec=2, direita=False, arredondar=True,␣
 ↪exibir_total=True):
    df_dist_freq = pd.DataFrame(construir_tabela(dados, tipo=tipo,␣
 ↪direita=direita, arredondar=arredondar))
    df_dist_freq.index = formatar_intervalos(df_dist_freq.index.array, prec,␣
 ↪direita=direita)
    df_dist_freq.index = df_dist_freq.index.rename('Classes')
    if exibir_total:
        total_dist = pd.DataFrame({'Frequência':df_dist_freq['Frequência'].
 ↪sum()}, index=['Total'])
        total_dist.index = total_dist.index.rename('Classes')
        df_dist_freq = pd.concat([df_dist_freq, total_dist])
    return  df_dist_freq.query('Frequência>0')
```

[77]:
```
dist_freq(dados)
```

[77]:

Classes	Frequência
[21.0, 24.0)	7
[24.0, 27.0)	9
[27.0, 30.0)	1
[30.0, 33.0)	5
[33.0, 36.0)	7
[36.0, 39.0)	1
Total	30

[78]:
```
z=np.random.normal(0,1,200)
dist_freq(z)
```

[78]:

Classes	Frequência
[-2.76, -1.76)	5
[-1.76, -0.76)	40
[-0.76, 0.24)	65
[0.24, 1.24)	73

```
[1.24, 2.24)            16
[2.24, 3.24)             1
Total                  200
```

[79]:
```
np.min(z)
```

[79]:
```
-2.7604132215733026
```

[80]:
```
np.max(z)
```

[80]:
```
2.6582617944970792
```

1.3 Medidas de Posição

- As medidas de posição são valores que representam a tendência de concentração dos dados observados.
- As mais importantes são as *medidas de tendência central*.
- As três medidas de tendência central mais utilizadas são: *Média*, *Moda* e *Mediana*.

1.3.1 Média

- É um valor que representa uma característica do conjunto de dados. Essa característica é tal que a soma dos dados é preservada. A média é obtida a partir de todos os elementos da distribuição e do tamanho da amostra.
- *Notação*: representamos a média de um conjunto de dados por $\overline{X}$.
- Calculamos a média aritmética pela fórmula:

$$\overline{X} = \sum_{i=1}^{n} \frac{X_i}{n}.$$

- Para Series e DataFrames o método **mean()** retorna a média dos valores.

[81]:
```
pd.Series(dados).mean()
```

[81]:
```
27.833333333333332
```

Média para dados agrupados em intervalos

- No caso em que temos os dados agrupados em intervalos, utilizamos a frequência e o ponto médio de cada classes para calcular a média pela fórmula:

$$\overline{X} = \sum_{i=1}^{K} \frac{F_i \cdot pm_i}{n},$$

onde K é o número de classes, F_i é a frequência da i-ésima classe e pm_i é o ponto médio da i-ésima classe.

[82]:
```
dist_freq_pm = dist_freq(dados, exibir_total=False)
intervalos = dist_freq_pm.index.array
dist_freq_pm['Ponto Médio'] = [(intervalo.left+intervalo.right)/2 for intervalo␣
 ↪in intervalos]
dist_freq_pm
```

[82]:

Classes	Frequência	Ponto Médio
[21.0, 24.0)	7	22.5
[24.0, 27.0)	9	25.5
[27.0, 30.0)	1	28.5
[30.0, 33.0)	5	31.5
[33.0, 36.0)	7	34.5
[36.0, 39.0)	1	37.5

[83]:
```
media = (dist_freq_pm['Frequência']*dist_freq_pm['Ponto Médio']).sum()/
 ↪dist_freq_pm['Frequência'].sum()
media
```

[83]: 28.4

[84]:
```
def media_dist_freq(d_freq):
    if(type(d_freq.index.array).__name__ != 'IntervalArray'):
        d_freq = d_freq.head(-1).copy()
    intervalos = d_freq.index.array
    d_freq['Ponto Médio'] = [(intervalo.left+intervalo.right)/2 for intervalo in␣
 ↪intervalos]
    return (d_freq['Frequência']*d_freq['Ponto Médio']).sum()/
 ↪d_freq['Frequência'].sum()
```

[85]:
```
media_dist_freq(dist_freq(dados))
```

[85]: 28.4

[86]:
```
media_dist_freq(dist_freq(z))
```

[86]: 0.030000000000000027

1.3.2 Moda

- Definimos a moda de um conjunto de dados como o valor mais frequente deste conjunto.
- *Notação*: representamos a moda de um conjunto de dados por *Mo*.
- *Exemplo*:
  - 1, 2, 4, 5 e 8 - não existe valor mais frequente - não existe moda (Amodal).
  - 2, 2, 3, 7 e 8 - *Mo* = 2 (Unimodal).
  - 1, 1, 10, 5, 5, 8, 7, 2 - *Mo* = 1 e 5 (Bimodal).

- Para Series e DataFrames o método **mode()** retorna a moda dos valores.

[87]:
```
pd.Series(dados).mode()
```

[87]:
```
0    25
dtype: int64
```

[88]:
```
pd.Series(z).mode()
```

[88]:
```
0     -2.760413
1     -2.240176
2     -2.233798
3     -2.031991
4     -2.004331
          ...
195    1.795184
196    2.057122
197    2.070173
198    2.194724
199    2.658262
Length: 200, dtype: float64
```

Moda em dados agrupados em intervalos

- Neste caso, utiliza-se a fórmula de Czuber identificando a *classe modal*, isto é, a classe com a maior frequencia.

$$\text{Mo} = l_{\text{Mo}} + \left[\frac{h(F_{\text{Mo}} - F_{\text{ant}})}{2F_{\text{Mo}} - (F_{\text{ant}} + F_{\text{Pos}})} \right],$$

onde:

h é a amplitude intervalar,

F_{Mo} é a frequência da *classe modal*,

l_{Mo} é o limite inferior da *classe modal*,

F_{ant} é a frequência da classe anterior à *classe modal*,

F_{Pos} é a frequência da classe posterior à classe modal.

[89]:
```
def encontra_indices_modais(d_freq):
    if(type(d_freq.index.array).__name__ != 'IntervalArray'):
        d_freq = d_freq.head(-1).copy()
    d_temp = d_freq.reset_index()['Frequência']
    return ((d_temp[d_temp == d_temp.max()]).index).to_numpy()

def encontra_freq_anterior(d_freq):
    idx_modal = encontra_indices_modais(d_freq).astype('float')
    idx_anterior = idx_modal-1
```

```
    idx_anterior[idx_anterior<0] = np.nan
    freq_anterior = d_freq['Frequência'].iloc[idx_anterior[~np.
 ↪isnan(idx_anterior)]].to_numpy()
    if(np.isnan(idx_anterior[0])):
        freq_anterior = np.insert(freq_anterior,0,0)
    return freq_anterior

def encontra_freq_posterior(d_freq):
    idx_modal = encontra_indices_modais(d_freq).astype('float')
    n_classes = d_freq.shape[0]
    idx_posterior = idx_modal+1
    idx_posterior[idx_posterior >= n_classes] = np.nan
    freq_posterior = d_freq['Frequência'].iloc[idx_posterior[~np.
 ↪isnan(idx_posterior)]].to_numpy()
    if(np.isnan(idx_posterior[-1])):
        freq_posterior = np.append(freq_posterior,0)
    return freq_posterior

def moda_dist_freq(d_freq):
    if(type(d_freq.index.array).__name__ != 'IntervalArray'):
        d_freq = d_freq.head(-1).copy()
        d_freq.index = d_freq.index.array.astype(pd.arrays.IntervalArray)
    idx_modal = encontra_indices_modais(d_freq)
    h = d_freq.index.array[0].right-d_freq.index.array[0].left
    lMo = d_freq.index.array[idx_modal].left.array
    FMo = d_freq.iloc[idx_modal]['Frequência'].array
    FPos = encontra_freq_posterior(d_freq)
    FAnt = encontra_freq_anterior(d_freq)
    return lMo + (h*(FMo-FAnt))/(2*FMo - (FAnt+FPos))
```

[90]:
```
moda_dist_freq(dist_freq(z))
```

[90]:
```
<PandasArray>
[0.3630769230769231]
Length: 1, dtype: float64
```

[91]:
```
moda_dist_freq(dist_freq(dados))
```

[91]:
```
<PandasArray>
[24.6]
Length: 1, dtype: float64
```

1.3.3 Mediana

- Definimos a mediana de um conjunto de dados como o valor que divide um o *rol de dados* em duas partes com a mesma quantidade de dados.
- Notação: representamos a mediana de um conjunto de dados por Md.

- O *elemento mediano*, E_{Md}, aponta o local no *rol de dados* onde a mediana está localizada. A mediana será o valor assumido na posição E_{Md}.
  - Se o tamanho amostral n é ímpar, temos que $E_{\text{Md}} = \frac{(n+1)}{2}$.
  - Caso tamanho amostral seja par, teremos dois valores possíveis para o elemento mediano: $(\frac{n}{2})$ e $\frac{n}{2} + 1$. Neste caso a mediana será a média dos valores assumidos nestas posições.
- Exemplos:
  - 1, 2, 4, 5, 8. Como n é ímpar, temos $E_{\text{Md}} = 3$, e $Md = 4$.
  - 2, 2, 3, 7, 8, 10. Aqui n é par, assim $E_{\text{Md},1} = \frac{6}{2} = 3$ e $E_{\text{Md},2} = \frac{6}{2} + 1 = 4$. Daí $Md = \frac{3+7}{2} = 5$.
- Para Series e DataFrames o método **median()** retorna a mediana dos valores.

[92]:
```
pd.Series(dados).median()
```

[92]:
```
26.0
```

[93]:
```
pd.Series(z).median()
```

[93]:
```
0.13149383237393403
```

Mediana em dados agrupados em intervalos

- Neste caso, utilizamos $E_{\text{Md}} = \frac{n}{2}$ independentemente de n ser par ou ímpar.
- A *classe mediana* é a primeira classe tal que $F_{\text{ac}} \geq E_{\text{Md}}$.
- Definimos a *mediana* pela fórmula

$$\text{Md} = l_{\text{Md}} + h \cdot \left[\frac{E_{\text{Md}} - F_{\text{ac,ant}}}{F_{\text{Md}}} \right],$$

onde,

l_{Md} é o limite inferior da *classe mediana*,

h é a amplitude do intervalo,

$F_{\text{ac,ant}}$ é a frequência acumulada da classe anterior à *classe mediana*,

F_{Md} é a frequência da *classe mediana*.

- Para Series e DataFrames o método **cumsum()** retorna a soma acumulada dos valores.

[94]:
```
d_freq_temp = dist_freq(dados, exibir_total=False)
d_freq_temp['Freq Acumulada'] = d_freq_temp['Frequência'].cumsum()
d_freq_temp
```

[94]:
```
             Frequência  Freq Acumulada
Classes
```

```
[21.0, 24.0)          7              7
[24.0, 27.0)          9             16
[27.0, 30.0)          1             17
[30.0, 33.0)          5             22
[33.0, 36.0)          7             29
[36.0, 39.0)          1             30
```

[95]:
```
def mediana_dist_freq(d_freq):
    if(type(d_freq.index.array).__name__ != 'IntervalArray'):
        d_freq = d_freq.head(-1).copy()
        d_freq.index = d_freq.index.array.astype(pd.arrays.IntervalArray)
    n_obs = d_freq['Frequência'].sum()
    h = d_freq.index.array[0].right-d_freq.index.array[0].left
    d_freq['Freq Acumulada'] = d_freq['Frequência'].cumsum()
    lMd = (d_freq[d_freq['Freq Acumulada'] >= n_obs/2].iloc[0]).name.left
    EMd = n_obs/2
    FMd = d_freq[d_freq['Freq Acumulada'] >= n_obs/2].iloc[0]['Frequência']
    if (d_freq['Freq Acumulada'] < n_obs/2).any():
        FAcAnt = d_freq[d_freq['Freq Acumulada'] < n_obs/2].iloc[-1]['Freq␣
↪Acumulada']
    else:
        FAcAnt = 0
    return lMd + h*(EMd-FAcAnt)/FMd
```

[96]:
```
mediana_dist_freq(dist_freq(z))
```

[96]:
```
0.08615384615384591
```

[97]:
```
mediana_dist_freq(dist_freq(dados))
```

[97]:
```
26.666666666666668
```

1.4 Medidas de Dispersão

- As medidas de dispersão medem o grau de variabilidade dos elementos de uma distribuição;
- O valor zero indica ausência de dispersão;
- A dispersão aumenta à medida que aumenta o valor da medida de dispersão.
- As principais Medidas de Dispersão: *Amplitude*, *Desvio Médio*, *Variância*, *Desvio Padrão*.
- Motivação para as medidas de dispersão

Alunos			Notas			Média
Antônio	5	5	5	5	5	5
João	6	4	5	4	6	5
José	10	5	5	5	0	5

Alunos			Notas			Média
Pedro	10	10	5	0	0	5

- Observa-se que:
  - As notas de Antônio não variaram;
  - As notas de João variaram menos do que as notas de José;
  - As notas de Pedro variaram mais do que as notas de todos os outros alunos.

1.4.1 Amplitude

- A amplitude nos fornece uma idéia do campo de variação dos elementos. Mais precisamente, ela fornece a maior variação possível dos dados.
- A amplitude é dada pela fórmula:

$$R = X_{\max} - X_{\min}.$$

  onde, $X_{\max}$ é o máximo dos valores nos dados e $X_{\min}$ é o mínimo dos valores nos dados.
- Para Series e DataFrames os métodos **max()** e **min()** retornam respectivamente o máximo e mínimo dos valores.

[98]:
```
R = pd.Series(dados).max()-pd.Series(dados).min(); R
```

[98]:
```
15
```

1.4.2 Desvio Médio

- Desejando-se medir a dispersão dos dados em relação a média, parece interessante a análise dos desvios em torno da média. Isto é, análise dos desvios:

$$d_i = (X_i - \overline{X}).$$

- Mas a soma de todos os desvios é igual a zero. Isto é:

$$\sum_{i=1}^{n} d_i = \sum_{i=1}^{n}(X_i - \overline{X}) = \sum_{i=1}^{n} X_i - \sum_{i=1}^{n} \overline{X} = \sum_{i=1}^{n} X_i - n\overline{X} =$$

$$= \sum_{i=1}^{n} X_i - n\frac{\sum_{i=1}^{n} X_i}{n} = \sum_{i=1}^{n} X_i - \sum_{i=1}^{n} X_i = 0.$$

- Logo, será preciso encontrar uma maneira de se trabalhar com os desvios sem que a soma dê zero. Dessa forma, define-se o *desvio médio*.
- Notação: representamos o *desvio médio* de um conjunto de dados por DM.
- Portanto, definimos o *desvio médio* pela fórmula:

$$DM = \sum_{i=1}^{n} \frac{|d_i|}{n} = \sum_{i=1}^{n} \frac{|X_i - \overline{X}|}{n}.$$

- Para Series e DataFrames o método **mad()** retorna a *desvio médio* dos valores.

[99]:
```
pd.Series(dados).mad()
```

[99]:
```
4.422222222222222
```

[100]:
```
pd.Series(z).mad()
```

[100]:
```
0.7585853154341842
```

Desvio médio em dados agrupados em intervalos

- No caso em que temos os dados agrupados em intervalos, utilizamos a frequência e o ponto médio de cada classes para calcular a *desvio médio* pela fórmula:

$$DM = \sum_{i=1}^{K} \frac{|d_i| \cdot F_i}{n} = \sum_{i=1}^{K} \frac{|pm_i - \overline{X}| \cdot F_i}{n}.$$

onde K é o número de classes, F_i é a frequência da i-ésima classe e pm_i é o ponto médio da i-ésima classe.

[101]:
```
def dm_dist_freq(d_freq):
    if(type(d_freq.index.array).__name__ != 'IntervalArray'):
        d_freq = d_freq.head(-1).copy()
    intervalos = d_freq.index.array
    d_freq['Ponto Médio'] = [(intervalo.left+intervalo.right)/2 for intervalo in␣
 ↪intervalos]
    return (d_freq['Frequência']*np.abs(d_freq['Ponto Médio'] -
                                        media_dist_freq(d_freq))).sum()/
 ↪(d_freq['Frequência'].sum())
```

[102]:
```
dm_dist_freq(dist_freq(dados))
```

[102]:
```
4.472413793103448
```

[103]:
```
dm_dist_freq(dist_freq(z))
```

[103]:
```
0.8176381909547737
```

Observações:

- A *amplitude* não mede bem a dispersão dos dados porque, usam-se apenas os valores extremos, ao invés de utilizar todos os elementos da distribuição.
- O *desvio médio* é mais vantajoso que a *amplitude*, visto que leva em consideração todos os valores da distribuição e é menos sensível a *outliers*.
- No entanto, *desvio médio* não é tão frequentemente empregado no ajuste de modelos, pois não apresenta propriedades matemáticas interessantes, porém é bastante utilizado na validação e comparação de modelos.

1.4.3 Variância

- A *variância* é a medida de dispersão mais utilizada. É o quociente entre a soma dos quadrados dos desvios e o número de elementos.
- Assim, temos a seguinte definição de *variância populacional* que a é dada pela fórmula:

$$\sigma^2 = \sum_{i=1}^{N} \frac{d_i^2}{N} = \sum_{i=1}^{N} \frac{(X_i - \overline{X})^2}{N}.$$

onde σ^2 indica a variância populacional e lê-se sigma ao quadrado ou sigma dois. Neste caso, $\overline{X}$ e N da formúla representam a média populacional e o tamanho populacional, respectivamente.

1.4.4 Variância Amostral

- Temos a seguinte definição de *variância amostral* que a é dada pela fórmula:

$$S^2 = \sum_{i=1}^{n} \frac{d_i^2}{n-1} = \sum_{i=1}^{n} \frac{(X_i - \overline{X})^2}{n-1}.$$

- Para Series e DataFrames o método **var()** retorna a *variância amostral* dos valores.

[104]:
```
pd.Series(dados).var()
```

[104]:
```
24.281609195402304
```

[105]:
```
pd.Series(z).var()
```

[105]:
```
0.8748136872613771
```

Variância amostral em dados agrupados em intervalos

- No caso em que temos os dados agrupados em intervalos, utilizamos a frequência e o ponto médio de cada classes para calcular a *variância* pela fórmula:

$$S^2 = \sum_{i=1}^{K} \frac{d_i^2 \cdot F_i}{n-1} = \sum_{i=1}^{K} \frac{(pm_i - \overline{X})^2 \cdot F_i}{n-1}.$$

onde K é o número de classes, F_i é a frequência da i-ésima classe e pm_i é o ponto médio da i-ésima classe.

[106]:
```
def var_dist_freq(d_freq):
    if(type(d_freq.index.array).__name__ != 'IntervalArray'):
        d_freq = d_freq.head(-1).copy()
    intervalos = d_freq.index.array
    d_freq['Ponto Médio'] = [(intervalo.left+intervalo.right)/2 for intervalo in␣
 ↪intervalos]
    return (d_freq['Frequência']*(d_freq['Ponto Médio'] -
```

```
                                    media_dist_freq(d_freq))**2).sum()/
↪(d_freq['Frequência'].sum()-1)
```

[107]:
```
var_dist_freq(dist_freq(dados))
```

[107]: 24.60875804666038

[108]:
```
var_dist_freq(dist_freq(z))
```

[108]: 0.9508396506407699

1.4.5 Desvio Padrão

- Temos também outra medida de dispersão, que é a raiz quadrada da variância, chamada de *desvio padrão*. Assim,

$$\sigma = \sqrt{\sigma^2} \quad \text{é o desvio desvio padrão populacional}$$

e

$$S = \sqrt{S^2} \quad \text{é o desvio desvio padrão amostral.}$$

- Para o cálculo do *desvio padrão* deve-se primeiramente determinar o valor da variância e, em seguida, extrair a raiz quadrada desse resultado.
- Para Series e DataFrames o método **std()** retorna a *variância amostral* dos valores.

[109]:
```
pd.Series(dados).std()
```

[109]: 4.9276372832628725

[110]:
```
np.sqrt(pd.Series(dados).var())
```

[110]: 4.9276372832628725

[111]:
```
pd.Series(z).std()
```

[111]: 0.935314753043796

[112]:
```
np.sqrt(pd.Series(z).var())
```

[112]: 0.935314753043796

Desvio Padrão para dados agrupados em intervalos

[113]:
```
def dp_dist_freq(d_freq):
    return np.sqrt(var_dist_freq(d_freq))
```

[114]:
```
dp_dist_freq(dist_freq(dados))
```

[114]:
```
4.960721524804671
```

[115]:
```
dp_dist_freq(dist_freq(z))
```

[115]:
```
0.9751100710385315
```

1.5 Resumo Estatístico de uma *Serie* ou *DataFrame*

Para obtermos um resumo estatístico de uma *Serie* ou *DataFrame* do *pandas* utilizamos o método **describe**.

O método **describe** exclui observações faltantes por padrão.

Exemplos:

[116]:
```
pd.Series(dados).describe()
```

[116]:
```
count    30.000000
mean     27.833333
std       4.927637
min      21.000000
25%      24.000000
50%      26.000000
75%      32.750000
max      36.000000
dtype: float64
```

[117]:
```
pd.DataFrame(z).describe()
```

[117]:
```
                0
count  200.000000
mean     0.014842
std      0.935315
min     -2.760413
25%     -0.638306
50%      0.131494
75%      0.671522
max      2.658262
```

Observações

- Se as entradas da *Serie* não forem numéricas o método *describe* retornará uma tabela contendo as quantidades de valores únicos, qual o valor mais frequente e qual a quantidade de elementos do valor mais frequente.
- No caso de um *DataFrame* que contenha colunas numéricas e colunas não-numéricas, o método *describe* só irá considerar as colunas numéricas.

Exemplos:

[118]:
```
serie_ex1 = pd.Series(['a','b','c','d','e','f','g','h','i','j'])
serie_ex2 = pd.Series(range(10))
```

[119]:
```
serie_ex1.describe()
```

[119]:
```
count     10
unique    10
top        g
freq       1
dtype: object
```

[120]:
```
serie_ex2.describe()
```

[120]:
```
count    10.00000
mean      4.50000
std       3.02765
min       0.00000
25%       2.25000
50%       4.50000
75%       6.75000
max       9.00000
dtype: float64
```

Exemplo:

[121]:
```
df_exemplo = pd.concat([serie_ex1, serie_ex2], axis=1)
```

[122]:
```
df_exemplo
```

[122]:

	0	1
0	a	0
1	b	1
2	c	2
3	d	3
4	e	4
5	f	5
6	g	6
7	h	7
8	i	8
9	j	9

Exemplo:

[123]:
```
df_exemplo.describe()
```

[123]:
```
                1
count  10.00000
mean    4.50000
std     3.02765
min     0.00000
25%     2.25000
50%     4.50000
75%     6.75000
max     9.00000
```

Observação: Podemos controlar o que será considerado no describe utilizando os argumentos *include* ou *exclude*. No caso, devemos colocar como argumento uma lista contendo os tipos a serem incluídos ou excluídos. Existem vários tipos que podem ser considerados para serem incluídos ou excluídos. Para uma lista dos tipos disponíveis, por favor consultem a documentação da função **select_dtypes()**.

Exemplos:

[124]:
```
df_exemplo.describe(exclude='number')
```

[124]:
```
          0
count    10
unique   10
top       g
freq      1
```

[125]:
```
df_exemplo.describe(include='object')
```

[125]:
```
          0
count    10
unique   10
top       g
freq      1
```

Exemplo:

[126]:
```
df_exemplo.describe(include='all')
```

[126]:
```
           0         1
count     10  10.00000
unique    10       NaN
top        g       NaN
freq       1       NaN
mean     NaN   4.50000
std      NaN   3.02765
min      NaN   0.00000
25%      NaN   2.25000
50%      NaN   4.50000
```

```
75%      NaN   6.75000
max      NaN   9.00000
```