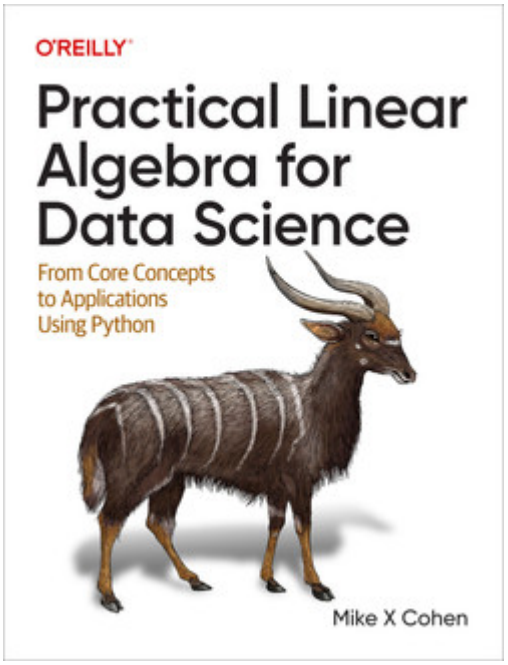
Matrizes - Parte 1



Uma matriz é um vetor levado ao próximo nível.

Matrizes são objetos matemáticos altamente versáteis. Elas podem guardar conjuntos de equações, transformações geométricas, as posições de partículas ao longo do tempo, registros financeiros e inúmeras outras coisas.

Em ciência de dados, algumas vezes as matrizes são chamadas de tabelas de dados, cujas linhas representam **observações** e as colunas as **variáveis**.

Bibliotecas

In [2]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.linalg import toeplitz
```

Criando e Visualizando Matrizes no NumPy

Dependendo do contexto, matrizes podem ser explicadas como um conjunto de vetores em colunas empilhadas uma ao lado da outra, ou linhas em camadas sobrepostas uma acima da outra, ou uma coleção ordenada de elementos individuais da matriz.

In [3]:
```
# Criando uma matriz 3x3 com valores inteiros
matriz = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print(matriz)
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

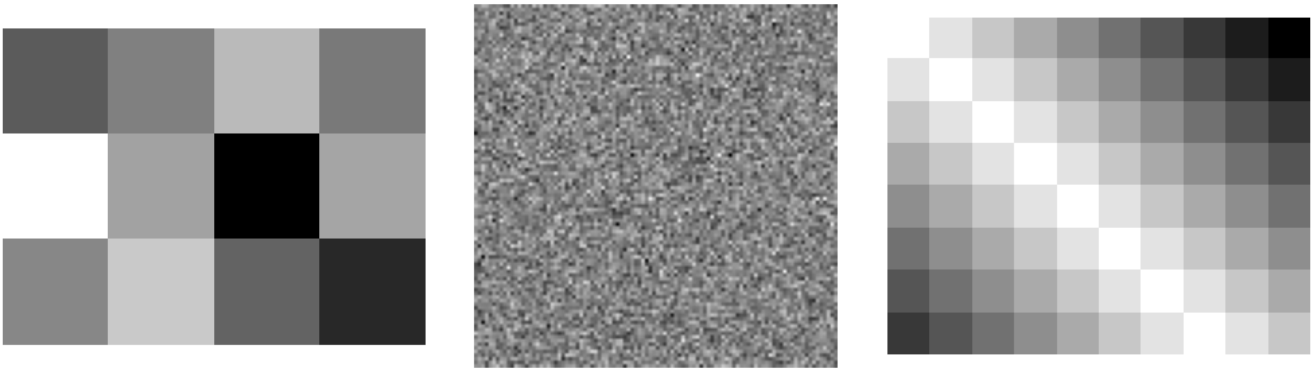
Visualizando, Indexando e Cortando Matrizes

$$\begin{bmatrix} 1 & 2 \\ \pi & 4 \\ 6 & 7 \end{bmatrix}, \begin{bmatrix} -6 & \frac{1}{3} \\ e^{4,3} & -1,4 \\ \frac{6}{5} & 0 \end{bmatrix}$$

Matrizes pequenas podem ser visualizadas inteiramente. Porém, na prática, muitas vezes elas serão muito maiores.

Portanto, elas podem ser representadas como imagens, onde os valores numéricos da matriz são substituídos por cores na imagem.

In [4]:
```
# Cria algumas matrizes
A = np.random.randn(3,4)
B = np.random.randn(100,100)
C = -toeplitz(np.arange(8),np.arange(10))

# E mostra elas como imagens
fig,axs = plt.subplots(1,3,figsize=(10,3))

axs[0].imshow(A,cmap='gray')
axs[1].imshow(B,cmap='gray')
axs[2].imshow(C,cmap='gray')

for i in range(3): axs[i].axis('off')
plt.tight_layout()
plt.show()
```



Matrizes são representadas por letras maiusculas em negrito, como **A** e **M**. O tamanho da matriz é indicado usando uma convenção de (linhas, colunas).

$$\boldsymbol{M} = \begin{bmatrix} 1 & 3 & 5 & 7 \\ 0 & 2 & 4 & 6 \\ 1 & 4 & 7 & 8 \end{bmatrix}$$

A matriz M é uma matriz de tamanho $(3, 4)$ porque contém 3 linhas e 4 colunas.

Você pode se referir a certo elemento da matriz pela posição do seu índice, por exemplo, o elemento na linha 2 e coluna 3 é representado por **M**(2, 3). Assim, **M**(2, 3) = 4.

No Python a indexação começa em 0, então o elemento na linha 2 e coluna 3 é representado por **M**[1, 2]. Assim, **M**[1, 2] = 4.

Extrair um subconjunto de linhas ou colunas de uma matriz é chamado de "fatiar" a matriz. Você pode fatiar uma matriz usando dois pontos (:) para indicar o intervalo de linhas ou colunas que você deseja extrair.

In [5]:
```
# Criando uma matriz 4x4
matriz = np.array([[10,  20,  30,  40 ],
                   [50,  60,  70,  80 ],
                   [90,  100, 110, 120],
                   [130, 140, 150, 160]])

# Fatiando a matriz: extraindo as linhas 1, 2 e as primeiras 3 colunas
fatiada = matriz[1:3, :3]

print('Matriz original:')
print(matriz)
print('')
print('Fatia da matriz:')
print(fatiada)
```

```
Matriz original:
[[ 10  20  30  40]
 [ 50  60  70  80]
 [ 90 100 110 120]
 [130 140 150 160]]

Fatia da matriz:
[[ 50  60  70]
 [ 90 100 110]]
```

Matrizes Especiais

As matrizes podem ser descritas usando uma quantidade relativamente pequena de características que criam "familias" ou categorias de matrizes. Essas categorias são importantes de conhecer porque elas aparecem em certas operações ou possuem certas propriedades específicas.

Matriz de números aleatórios

Essa é uma matriz que contém números aleatórios de alguma distribuição, normalmente a Gaussiana (conhecida tamém como Normal). Elas são úteis para explorar a algebra linear.

In [6]:
```
Mrows = 4
Ncols = 6

matriz = np.random.randn(Mrows, Ncols)
print('Matriz:')
print(matriz)
```

```
Matriz:
[[ 0.95421899 -0.71822086  0.38461992 -1.97729058  0.46520551  0.72566208]
 [-0.3530108   0.21146224  1.38237949  0.32868897  0.83858079  0.2674361 ]
 [ 0.25601023 -3.01484778  0.27666465 -0.292508    0.6331631  -0.89345783]
 [ 0.64813392 -0.27705688 -0.01039789  0.14551602 -0.22211495 -1.37931426]]
```

Quadrada X Não-Quadrada (Retangular)

Uma matriz quadrada possui o mesmo número de linhas e colunas ($\mathbb{R}^{N\times N}$). Já uma matriz não-quadrada, ou retângular, tem um número diferente de linhas e colunas. Por exemplo, uma matriz 4×6 possui 4 linhas e 6 colunas.

Você pode criar matrizes quadradas ou não-quadradas ajustando os valores de `Mrows` e

`Ncols` no código anterior.

In [7]:
```
# Criando uma matriz quadrada 3x3
matriz_quadrada = np.random.randint(1, 10, size=(3, 3))
print("Matriz Quadrada:")
print(matriz_quadrada)

# Criando uma matriz retangular 3x6
matriz_retangular = np.random.randint(1, 10, size=(3, 6))
print("\nMatriz Retangular:")
print(matriz_retangular)
```

```
Matriz Quadrada:
[[5 2 8]
 [2 4 5]
 [1 6 9]]

Matriz Retangular:
[[8 4 3 8 7 9]
 [5 9 4 6 2 7]
 [3 1 6 7 9 4]]
```

Matrizes retangulares são chamadas de altas (tall) se tiverem mais linhas que colunas, ou largas (wide) se for ao contrário.

In [8]:
```
# Matriz retangular alta (6 linhas e 3 colunas)
matriz_alta = np.random.randint(1, 10, size=(6, 3))
print("Matriz Retangular Alta:")
print(matriz_alta)

# Matriz retangular larga (3 linhas e 6 colunas)
matriz_larga = np.random.randint(1, 10, size=(3, 6))
print("\nMatriz Retangular Larga:")
print(matriz_larga)
```

```
Matriz Retangular Alta:
[[7 6 6]
 [1 8 3]
 [9 4 8]
 [7 1 8]
 [1 6 1]
 [6 7 9]]

Matriz Retangular Larga:
[[6 1 9 4 4 8]
 [1 8 3 5 4 2]
 [4 7 1 9 3 2]]
```

Diagonal

A diagonal de uma matriz consiste nos elementos que começam no canto superior esquerdo e descem até o canto inferior direito. Uma matriz diagonal é aquela que possui zeros em todos os elementos fora da sua diagonal principal. Os elementos na diagonal principal podem conter zeros, mas são os únicos que podem ter valores diferentes de zero.

A função `np.diag()` do NumPy possui dois comportamentos distintos, dependendo da entrada fornecida:

1. Se a entrada for uma matriz, a função retornará um vetor contendo os elementos da

diagonal principal.

2. Se a entrada for um vetor, a função retornará uma matriz com os elementos do vetor posicionados na diagonal principal, enquanto os demais elementos serão preenchidos com zeros.

In [9]:
```
#Criar uma matriz diagonal a partir de um vetor
vetor = np.array([1, 2, 3, 4])
matriz_diagonal = np.diag(vetor)
print("Matriz Diagonal criada a partir do vetor:")
print(matriz_diagonal)

#Extrair a diagonal principal de uma matriz
diagonal_extraida = np.diag(matriz_quadrada)
print("\nDiagonal principal extraída da matriz_quadrada:")
print(diagonal_extraida)
```

```
Matriz Diagonal criada a partir do vetor:
[[1 0 0 0]
 [0 2 0 0]
 [0 0 3 0]
 [0 0 0 4]]

Diagonal principal extraída da matriz_quadrada:
[5 4 9]
```

Triângular

Uma matriz triangular possui apenas zeros acima ou abaixo da diagonal principal. A matriz é chamada de triangular superior se os elementos diferentes de zero estão acima da diagonal, ou triangular inferior se estiverem abaixo.

O NumPy tem funções dedicadas para extrair o triângulo superior (`np.triu()`) ou inferior (`np.tril()`) de uma matriz.

In [10]:
```
# Criando uma matriz triangular superior a partir de uma matriz existente
triangular_superior = np.triu(matriz_quadrada)
print("Matriz Triangular Superior:")
print(triangular_superior)

# Criando uma matriz triangular inferior a partir de uma matriz existente
triangular_inferior = np.tril(matriz_quadrada)
print("\nMatriz Triangular Inferior:")
print(triangular_inferior)
```

```
Matriz Triangular Superior:
[[5 2 8]
 [0 4 5]
 [0 0 9]]

Matriz Triangular Inferior:
[[5 0 0]
 [2 4 0]
 [1 6 9]]
```

Identidade

Equivalente ao número 1, no sentido de que qualquer matriz ou vetor multiplicado pela matriz idêntidade resultar na mesma matriz ou vetor. A matriz identidade é uma matriz diagonal quadrada com todos os elementos diagonais tendo o valor de 1. Ela é indicada

pela letra $\bold{I}$ e pode ter um número subscrito informando seu tamanho ($\bold{I}_5$ é uma matriz idêntidade 5×5).

Você pode criar uma matriz identidade no Python usando o `np.eye()`.

In [11]:
```
# Criando uma matriz identidade 4x4
matriz_identidade = np.eye(4)
print("Matriz Identidade:")
print(matriz_identidade)
```

```
Matriz Identidade:
[[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]]
```

In [12]:
```
# Multiplicando a matriz identidade pela matriz_quadrada
matriz_identidade = np.eye(matriz_quadrada.shape[0])
resultado = np.dot(matriz_identidade, matriz_quadrada)

print('Matriz Quadrada:')
print(matriz_quadrada)
print('\nMatriz Identidade:')
print(matriz_identidade)
print('\nResultado da multiplicação da matriz identidade pela matriz_quadrada:')
print(resultado)
```

```
Matriz Quadrada:
[[5 2 8]
 [2 4 5]
 [1 6 9]]

Matriz Identidade:
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]

Resultado da multiplicação da matriz identidade pela matriz_quadrada:
[[5. 2. 8.]
 [2. 4. 5.]
 [1. 6. 9.]]
```

Zeros

A matriz zero é comparavel ao vetor zero: uma matriz com apenas zeros. Ela é indicada pelo zero em negrito: $\bold{0}$.

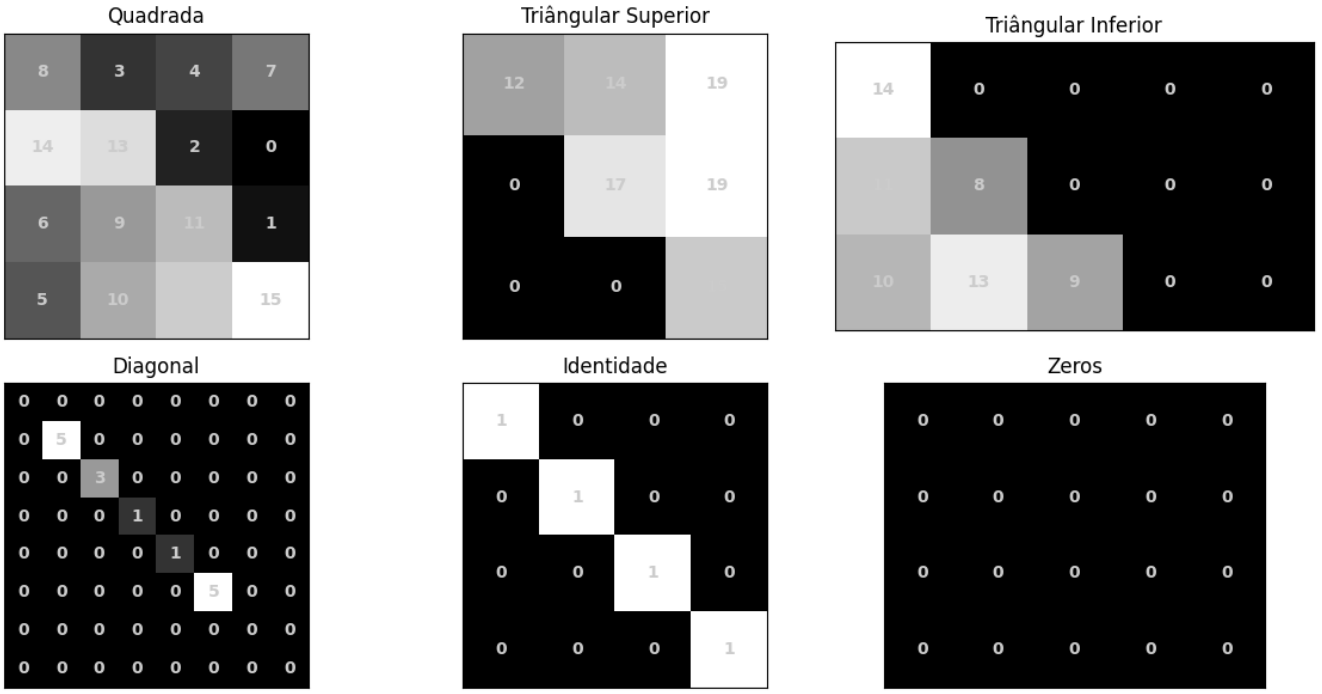
Pode ser criada pela função `np.zeros()`.

In [13]:
```
# Criando uma matriz zero 3x3
matriz_zero = np.zeros((3, 3))
print("Matriz Zero:")
print(matriz_zero)
```

```
Matriz Zero:
[[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]
```

In [14]:
```
## Criando imagens das matrizes

# Quadrada
```

```
M1 = np.random.permutation(16).reshape(4,4)

# Triangular superior
M2 = np.triu(np.random.randint(10,20,(3,3)))

# Triangular inferior
M3 = np.tril(np.random.randint(8,16,(3,5)))

# Diagonal
M4 = np.diag( np.random.randint(0,6,size=8) )

# Identidade
M5 = np.eye(4,dtype=int)

# Zeros
M6 = np.zeros((4,5),dtype=int)

matrices  = [ M1,M2,M3,M4,M5,M6 ]
matLabels = [ 'Quadrada','Triângular Superior','Triângular Inferior','Diagonal','Id


_,axs = plt.subplots(2,3,figsize=(12,6))
axs = axs.flatten()

for mi,M in enumerate(matrices):
  axs[mi].imshow(M,cmap='gray',origin='upper',
                 vmin=np.min(M),vmax=np.max(M))
  axs[mi].set(xticks=[],yticks=[])
  axs[mi].set_title(matLabels[mi])

  # Labels de texto
  for (j,i),num in np.ndenumerate(M):
    axs[mi].text(i,j,num,color=[.8,.8,.8],ha='center',va='center',fontweight='bold'

plt.tight_layout()
plt.show()
```



Adição, Multiplicação Escalar e Multiplicação Hadamard

Adição e Subtração

Você pode somar duas matrizes somando seus elementos correspondentes.

$$\begin{bmatrix} 2 & 3 & 4 \\ 1 & 2 & 4 \end{bmatrix} + \begin{bmatrix} 0 & 3 & 1 \\ -1 & -4 & 2 \end{bmatrix} = \begin{bmatrix} 2 & 6 & 5 \\ 0 & -2 & 6 \end{bmatrix}$$

A soma só pode ser realizada entre matrizes de mesmo tamanho, ou seja, que possuem o mesmo número de linhas e colunas.

In [15]:
```
# Definindo as matrizes
matriz1 = np.array([[2, 3, 4],
                                        [1, 2, 4]])

matriz2 = np.array([[0,  3, 1],
                                        [-1, -4, 2]])

# Somando as matrizes
resultado = matriz1 + matriz2

print("Matriz 1:")
print(matriz1)
print("\nMatriz 2:")
print(matriz2)
print("\nResultado da soma:")
print(resultado)
```

```
Matriz 1:
[[2 3 4]
 [1 2 4]]

Matriz 2:
[[ 0  3  1]
 [-1 -4  2]]

Resultado da soma:
[[ 2  6  5]
 [ 0 -2  6]]
```

"Deslocando" uma Matriz

Assim como acontece com vetores, não é formalmente possível somar um escalar diretamente a uma matriz, como em $\lambda + \bold{A}$.

Entretanto, existe uma forma de "somar" um escalar a uma matriz quadrada, procedimento conhecido como "deslocamento" (shifting) da matriz. Para isso, adiciona-se o valor constante apenas aos elementos da diagonal principal, o que pode ser feito somando o produto do escalar pela matriz identidade:

$$\bold{A} + \lambda I$$

$$\begin{bmatrix} 4 & 5 & 1 \\ 0 & 1 & 11 \\ 4 & 9 & 7 \end{bmatrix} + 6. \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 10 & 5 & 1 \\ 0 & 7 & 11 \\ 4 & 9 & 13 \end{bmatrix}$$

In [20]:
```
matriz = np.array([[4, 5, 1 ],
                   [0, 1, 11],
                   [4, 9, 7]])
s = 6
```

```
# Shifting a matriz
print('Matriz deslocada:')
print(matriz + s*np.eye(len(matriz)))
```

```
Matriz deslocada:
[[10.  5.  1.]
 [ 0.  7. 11.]
 [ 4.  9. 13.]]
```

Na prática, desloca-se uma quantidade relativamente pequena para preservar o máximo de informação em uma matriz, enquanto se aproveita os benefícios do deslocamento, como o aumento da estabilidade numérica da matriz.

Deslocar uma matriz tem duas aplicações principais:

1. Serve como mecanismo para encontrar os autovalores de uma matriz;
2. Atua como mecanismo de regularização de matrizes ao ajustar modelos aos dados.

Multiplicações Escalares e Hadamard

Uma multiplicação de matriz-escalar multiplica cada elemento na matriz pelo mesmo escalar.

$$\gamma \begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} \gamma a & \gamma b \\ \gamma c & \gamma d \end{bmatrix}$$

A multiplicação de Hadamard trabablha multiplicando duas matrizes de forma elementar

$$\begin{bmatrix} 2 & 3 \\ 4 & 5 \end{bmatrix} . \begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} 2a & 3b \\ 4c & 5d \end{bmatrix}$$

In [34]:
```
print("Matriz quadrada:")
print(matriz_quadrada)

print("\nMatriz triangular superior:")
print(triangular_superior)

# Multiplicação escalar
escalar = 3
print(f'\nEscalar: {escalar}')
resultado_escalar = escalar * matriz_quadrada

print("\nResultado da multiplicação escalar:")
print(resultado_escalar)

# Multiplicação de Hadamard
resultado_hadamard = matriz_quadrada * triangular_superior
print("\nResultado da multiplicação de Hadamard:")
print('Matriz quadrada X Matriz triangular superior')
print(resultado_hadamard)
```

```
Matriz quadrada:
[[5 2 8]
 [2 4 5]
 [1 6 9]]

Matriz triangular superior:
[[5 2 8]
 [0 4 5]
 [0 0 9]]

Escalar: 3

Resultado da multiplicação escalar:
[[15  6 24]
 [ 6 12 15]
 [ 3 18 27]]

Resultado da multiplicação de Hadamard:
Matriz quadrada X Matriz triangular superior
[[25  4 64]
 [ 0 16 25]
 [ 0  0 81]]
```

A multiplicação de Hadamard tem algumas aplicações na álgebra linear, como, por exemplo, no cálculo da matriz inversa. Entretanto, é mais utilizada em aplicações como uma forma conveniente de armazenar diversas operações individuais.

Multiplicação Padrão de Matrizes

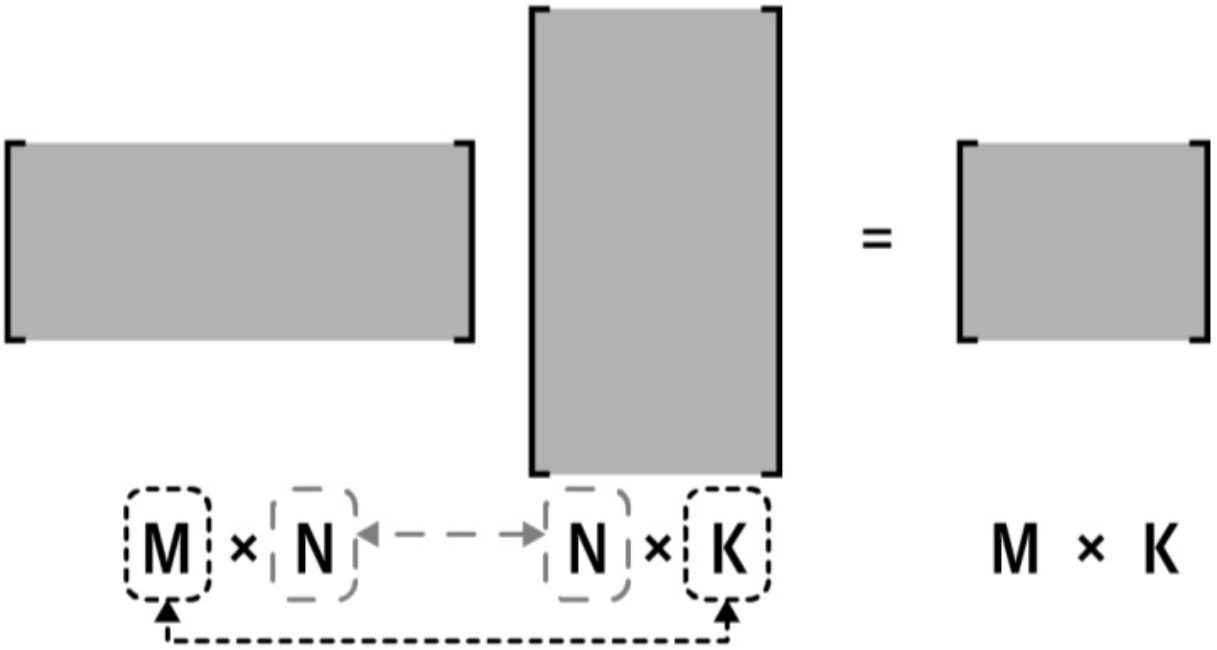
A multiplicação padrão de matrizes não opera elemento a elemento, mas sim por linhas e colunas. Nesse processo, cada elemento da matriz resultante é obtido calculando o produto escalar entre uma linha da primeira matriz e uma coluna da segunda matriz.

Regras para Validação de uma Multiplicação de Matrizes

Você já sabe que o tamanho das matrizes é representado por $\bold{M} \times \bold{N}$. Duas matrizes que serão multiplicadas podem ter tamanhos diferentes. Vamos nos referir ao tamanho da segunda matriz como $\bold{N} \times \bold{K}$.

A multiplicação de matrizes só é válida quando as dimensões "interiores" ($\bold{N}$) são iguais, ou seja, o número de colunas da primeira matriz deve ser igual ao número de linhas da segunda matriz. O tamanho da matriz resultante será definido pelas dimensões "exteriores", ou seja, $\bold{M} \times \bold{K}$.

Exemplo:
Se a matriz $\bold{A}$ tem tamanho 3×2 e a matriz $\bold{B}$ tem tamanho 2×4, a multiplicação $\bold{A} \cdot \bold{B}$ é válida, e a matriz resultante terá tamanho 3×4.

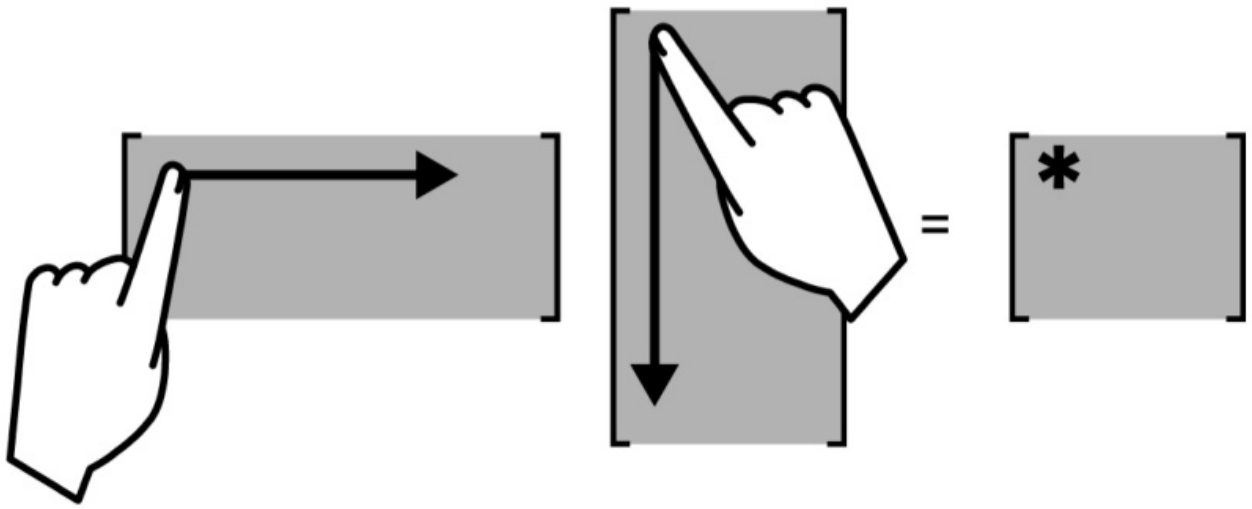


A multiplicação de matrizes não obedece à lei de comutação: $\boldsymbol{A} \cdot \boldsymbol{B}$ pode ser válida enquanto $\boldsymbol{B} \cdot \boldsymbol{A}$ é inválida. Mesmo quando ambas as multiplicações são válidas (por exemplo, se ambas as matrizes forem quadradas), elas podem produzir resultados diferentes:
$\boldsymbol{C} = \boldsymbol{A} \cdot \boldsymbol{B}$ e $\boldsymbol{D} = \boldsymbol{B} \cdot \boldsymbol{A}$, então, em geral,
$\boldsymbol{C} \neq \boldsymbol{D}$.

Multiplicação de Matriz

O motivo pelo qual a multiplicação de matrizes é válida somente se o número de colunas da matriz à esquerda corresponder ao número de linhas da matriz à direita é que o (i, j)-ésimo elemento na matriz resultante é o produto escalar entre a i-ésima linha da matriz à esquerda e a j-ésima coluna da matriz à direita.

$$\begin{bmatrix} 2 & 3 \\ 4 & 5 \end{bmatrix} \cdot \begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} (2a + 3c) & (2b + 3d) \\ (4a + 5c) & (4b + 5d) \end{bmatrix}$$



Lembre-se que o produto escalar é a soma dos produtos dos elementos correspondentes de dois vetores. O resultado da multiplicação de matrizes é uma nova matriz que guarda todas as relações lineares em pares entre as linhas da matriz à esquerda e as colunas da matriz à direita.

Essa propriedade é a base para o cálculo de matrizes de covariância e correlação, o modelo linear geral (usado em análises estatísticas, incluindo ANOVAs e regressões), decomposição de valores singulares e muitas outras aplicações.

In [40]:
```
# Exemplo 1: Multiplicação válida de matrizes
A = np.array([[1, 2], [3, 4], [5, 6]])  # Matriz 3x2
B = np.array([[7, 8, 9], [10, 11, 12]])  # Matriz 2x3

# Multiplicação padrão
C = np.dot(A, B)  # Ou A @ B
print("Matriz A (3x2):")
print(A)
print("\nMatriz B (2x3):")
print(B)
print("\nResultado da multiplicação A x B (3x3):")
print(C)
```

```
Matriz A (3x2):
[[1 2]
 [3 4]
 [5 6]]

Matriz B (2x3):
[[ 7  8  9]
 [10 11 12]]

Resultado da multiplicação A x B (3x3):
[[ 27  30  33]
 [ 61  68  75]
 [ 95 106 117]]
```

In [41]:
```
# Exemplo 2: Multiplicação de matrizes quadradas
E = np.array([[1, 2], [3, 4]])  # Matriz 2x2
F = np.array([[5, 6], [7, 8]])  # Matriz 2x2

# Multiplicação padrão
G = np.dot(E, F)  # Ou E @ F
H = np.dot(F, E)  # Ou E @ F

print("\nMatriz E (2x2):")
print(E)
print("\nMatriz F (2x2):")
print(F)
print("\nResultado da multiplicação E x F (2x2):")
print(G)
print("\nResultado da multiplicação F x E (2x2):")
print(H)
```

```
Matriz E (2x2):
[[1 2]
 [3 4]]

Matriz F (2x2):
[[5 6]
 [7 8]]

Resultado da multiplicação E x F (2x2):
[[19 22]
 [43 50]]

Resultado da multiplicação F x E (2x2):
[[23 34]
 [31 46]]
```

Multiplicação Matriz-Vetor

A multiplicação matriz-vetor tem muitas aplicações em ciência de dados, machine learning e computação gráfica. O básico:

- Uma matriz pode ser multiplicada à direita por um vetor coluna, mas não diretamente por um vetor linha. Da mesma forma, pode ser multiplicada à esquerda por um vetor linha, mas não diretamente por um vetor coluna. Ou seja, $\bold{A} \cdot v$ e $v^T \cdot \bold{A}$ são válidos, mas $\bold{A} \cdot v^T$ e $v \cdot \bold{A}$ não.

- O resultado de uma multiplicação de matriz-vetor é sempre um vetor, e sua orientação (coluna ou linha) é a mesma do vetor que multiplica.

Uma multiplicação matriz-vetor tem muitas aplicações:

1. **Na estatística**: Os dados preditos pelos modelos são resultados da multiplicação de uma matriz pelos coeficientes de regressão, que é escrito como $\bold{X}\beta$.
2. **No PCA (Análise de Componentes Principais)**: Um vetor de pesos de *feature-importance* é identificado maximizando a variância no conjunto de dados $\bold{Y}$, e isso é representado como $(\bold{Y}^T \cdot \bold{Y})v$.
3. **No processamento de sinais multivariados**: Um componente de dimensão reduzida é obtido aplicando-se um filtro espacial ao multicanal de dados de série temporal $\bold{S}$, representado como $w^T\bold{S}$.
4. **Na geometria e computação gráfica**: Um conjunto de coordenadas de imagem pode ser transformado usando uma matriz de transformação matemática, representada como $\bold{T}p$.

In [42]:
```
# Definindo uma matriz 3x3
A = np.array([[1, 2, 3],
              [4, 5, 6],
              [7, 8, 9]])

# Definindo um vetor coluna (3x1)
v_coluna = np.array([[1, 2, 3]]).T

# Multiplicação matriz-vetor (A * v_coluna)
resultado_coluna = np.dot(A, v_coluna)
print("Matriz A:")
print(A)
print("\nVetor coluna v:")
print(v_coluna)
print("\nResultado da multiplicação A * v_coluna:")
print(resultado_coluna)

# Definindo um vetor linha (1x3)
v_linha = np.array([1, 2, 3])

# Multiplicação vetor linha * matriz (v_linha * A)
resultado_linha = np.dot(v_linha, A)
print("\nVetor linha v:")
print(v_linha)
print("\nResultado da multiplicação v_linha * A:")
print(resultado_linha)
```

```
Matriz A:
[[1 2 3]
 [4 5 6]
 [7 8 9]]

Vetor coluna v:
[[1]
 [2]
 [3]]

Resultado da multiplicação A * v_coluna:
[[14]
 [32]
 [50]]

Vetor linha v:
[1 2 3]

Resultado da multiplicação v_linha * A:
[30 36 42]
```

Operações de Matriz: Transposta

O principio contua o mesmo: trocar as linhas pelas colunas e vice-versa.

$$a_{a,j}^T = a_{j,i}$$

$$\begin{bmatrix} 3 & 0 & 4 \\ 9 & 8 & 3 \end{bmatrix}^T = \begin{bmatrix} 3 & 9 \\ 0 & 8 \\ 4 & 3 \end{bmatrix}$$

In [44]:
```python
# Definindo a matriz
A = np.array([[3, 0, 4],
              [9, 8, 3]])

# Transpondo a matriz
A_transposta = A.T

print("Matriz A:")
print(A)
print("\nMatriz Transposta A^T:")
print(A_transposta)
```

```
Matriz A:
[[3 0 4]
 [9 8 3]]

Matriz Transposta A^T:
[[3 9]
 [0 8]
 [4 3]]
```

Operação de Matriz: LIVE EVIL (Ordem da operação)

LIVE EVIL é um palíndromo (uma palavra ou frase que é soletrada da mesma maneira de trás para frente) e uma forma de lembrar como a transposição afeta a ordem de multiplicar as matrizes.

Basicamente, a transposta do produto de matrizes é igual ao produto das transpostas das matrizes, mas na ordem inversa. Em termos matemáticos:

$$(LIVE)^T = E^T V^T I^T L^T$$

Essa propriedade é útil em álgebra linear e aparece frequentemente em manipulações de matrizes em ciência de dados e aprendizado de máquina.

In [45]:
```
# Definindo duas matrizes
L = np.array([[1, 2], [3, 4]])
I = np.array([[5, 6], [7, 8]])

# Produto das matrizes
produto = np.dot(L, I)

# Transposta do produto
transposta_produto = produto.T

# Produto das transpostas na ordem inversa
produto_transpostas = np.dot(I.T, L.T)

print("Matriz L:")
print(L)
print("\nMatriz I:")
print(I)
print("\nProduto L * I:")
print(produto)
print("\nTransposta do produto (L * I)^T:")
print(transposta_produto)
print("\nProduto das transpostas na ordem inversa I^T * L^T:")
print(produto_transpostas)
```

```
Matriz L:
[[1 2]
 [3 4]]

Matriz I:
[[5 6]
 [7 8]]

Produto L * I:
[[19 22]
 [43 50]]

Transposta do produto (L * I)^T:
[[19 43]
 [22 50]]

Produto das transpostas na ordem inversa I^T * L^T:
[[19 43]
 [22 50]]
```

Matrizes Simétricas

Matrizes simétricas têm muitas propriedades especiais que as tornam ótimas para trabalhar. Elas tendem a ser numericamente estáveis e, portanto, são convenientes para cálculos em algoritmos.

Uma matriz simétrica é aquela em que as linhas e as colunas correspondentes são iguais. Em outras palavras, a matriz permanece inalterada quando transposta, ou seja, $A^T = A$.

Exemplo de matriz simétrica:

$$\begin{bmatrix} a & e & f & g \\ e & b & h & i \\ f & h & c & j \\ g & i & j & d \end{bmatrix}$$

Observação: Uma matriz retangular não pode ser simétrica.

Criando Matrizes Simétricas átraves de Matrizes Assimétricas

Multiplicar qualquer matriz - mesmo uma retangular e assimétrica - por sua transposta sempre produzirá uma matriz quadrada e simétrica.

Em outras palavras, $A^T A$ é uma matriz quadrada e simétrica, assim como AA^T.

In [50]:
```
# Criando uma matriz retangular assimétrica
A = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

# Multiplicando a matriz pela sua transposta (A^T A)
AT_A = np.dot(A.T, A)

# Multiplicando a transposta da matriz pela matriz original (A A^T)
A_AT = np.dot(A, A.T)

print("Matriz A:")
print(A)
print("\nMatriz A^TA (quadrada e simétrica):")
print(AT_A)
print("\nMatriz AA^T (quadrada e simétrica):")
print(A_AT)
```

```
Matriz A:
[[1 2 3]
 [4 5 6]
 [7 8 9]]

Matriz A^TA (quadrada e simétrica):
[[ 66  78  90]
 [ 78  93 108]
 [ 90 108 126]]

Matriz AA^T (quadrada e simétrica):
[[ 14  32  50]
 [ 32  77 122]
 [ 50 122 194]]
```