

*Instituto Nacional de
Telecomunicações - INATEL*

AULA 2

SUMARIO

- 1 - Conceitos fundamentais de python
- 2 - Manipulação de dados com numpy, pandas e matplotlib
- 3 - EDA simples

1 - INTRODUÇÃO AO PYTHON

Para começarmos a construção de uma rede neural, precisamos primeiro, conhecer um pouco da linguagem que vamos utilizar ao longo do curso. E essa linguagem é o python



INATEL

INSTALAÇÃO DO PYTHON

[Link de download](#)

CONFIRAM A VERSÃO QUE VOCES INSTALARARAM
DO PYTHON. VAMOS UTILIZAR A 3.11



PYTHON

Curiosidades da linguagem:

- Fortemente tipada
- Tipagem dinâmica
- Tudo é objetivo
- Não é necessário declarar

LISTAS

Listas são estruturas de dados usadas para armazenar varios valores dentro de uma única variável. Elas são : ordenadas, mutaveis e permitem diferentes tipos de dados.

Representamos uma lista por :

`dados = [10, 20,30]`



```
dados = [10,20 ,30]
print(type(dados))
```

... <class 'list'>

LISTAS

Em DS as listas são bastante utilizadas para :

- *Organizar dados brutos*
- *Organizar informações*
- *Manipular valores antes da análise*
- *Servir de base para bibliotecas como np e pandas*



OPERAÇÕES COM LISTAS

Temos diversas operações com listas. Na iamgem ao lado temos os exemplos mais usados e cada um com sua importancia.

- *Append : adiciona item no final*
- *Insert: Insere na posição X*
- *Pop : Remove e retorna o numero*
- *Extend : Estende a lista*
- *Remove : Remove o item*
- *Sort : Ordena a lista*

Criação	Acesso
<ul style="list-style-type: none">• <code>lista1 = [1, 2, 3, 4]</code>• <code>lista2 = ['a', 'b', 'c']</code>• <code>lista3 = list(range(5))</code>	<ul style="list-style-type: none">• <code>lista[0] → 1</code>• <code>lista[2] → 3</code>• <code>lista[-1] → 'd'</code>
Operações Básicas	
<p> append: <code>lista.append(5)</code> Adiciona ao final.</p>	<p> extend: <code>lista.extend([6, 7])</code> Estende a lista.</p>
<p> insert: <code>lista.insert(1, 'x')</code> Insere na posição.</p>	<p> remove: <code>lista.remove(3)</code> Remove um elemento.</p>
<p> pop: <code>lista.pop()</code> Remove e retorna.</p>	<p> sort: <code>lista.sort()</code> Ordena a lista.</p>

OPERAÇÕES COM LISTAS

```
#Calculo de media
notas = [7.5, 8.0, 6.5, 9.0]

media = sum(notas) / len(notas)

print("Média:", media)
```

```
#filtrando valores >= 7
notas = [5.0, 7.5, 8.0, 6.0, 9.0]

aprovados = []

for nota in notas:
    if nota >= 7:
        aprovados.append(nota)

print("Aprovados:", aprovados)
```

Aprovados: [7.5, 8.0, 9.0]

```
#Encontrando o maior e o menor valor de uma lista
temperaturas = [22.5, 23.1, 21.8, 24.0]

maior = max(temperaturas)
menor = min(temperaturas)

print("Maior:", maior)
print("Menor:", menor)
```

Maior: 24.0
Menor: 21.8

DICIONÁRIOS

```
▶ aluno = {  
    "nome": "Pedro",  
    "idade": 23,  
    "nota": 8.5  
}  
type(aluno)
```

```
... dict
```

Dicionários são estruturas que armazenam informações chave → valor. Eles são : Ordenados, mutáveis, ideais para representar dados estruturados.

DICIONÁRIOS

Em DS, eles são bastante importantes. Utilizamos eles para representar registros, armazenar métricas, contar frequências, trabalhar com dados padrao json, organizar parâmetros de modelos.

DICIONÁRIOS

Vamos saber como acessar valores dentro de um dicionário:

```
▶ aluno = {  
    "nome": "Ana",  
    "idade": 21,  
    "nota": 9.0  
  
}  
  
print(aluno["nome"])  
print(aluno["nota"])  
  
... Ana  
9.0
```

Iterando dentro de um dicionário :

```
▶ produto = {  
    "nome": "Notebook",  
    "preco": 3500,  
    "estoque": 10  
  
}  
  
for chave, valor in produto.items():  
    print(chave, ":", valor)  
  
... nome : Notebook  
    preco : 3500  
    estoque : 10
```

DICIONÁRIOS

Outras aplicações:

CONTAGEM DE FREQUENCIAS (QUANTAS VEZES CADA NÚMERO APARECE):

```
numeros = [1, 2, 2, 3, 1, 2, 4]
contagem = {}

for numero in numeros:
    if numero in contagem:
        contagem[numero] += 1
    else:
        contagem[numero] = 1

print(contagem)
```

```
{1: 2, 2: 3, 3: 1, 4: 1}
```

ATUALIZAÇÃO DE VALORES

```
produto = {
    "nome": "Mouse",
    "preco": 100,
    "estoque": 20
}

produto["estoque"] -= 1 # vendeu 1 unidade

print(produto)

{'nome': 'Mouse', 'preco': 100, 'estoque': 19}
```

DICIONÁRIOS

Comandos
básicos :



Dicionários em Python

Comandos Básicos

{ } Criação:

```
dicionário = {'nome': 'Ana', 'idade': 25, 'nota': 8.5}
```

Cria um dicionário

key Acessar:

```
dicionário['nome']
```

Acessa valores

```
dicionário.get('idade')
```

+ Adicionar/Atualizar:

```
dicionário['email'] = "ana@email.com"
```

```
dicionário['idade'] = 26
```

Adiciona ou atualiza valores

trash Remover:

```
del dicionário['idade']
```

```
idade = dicionário.pop('idade')
```

Remove valores

refresh Iterar:

```
for chave, valor in dicionário.items():
```

```
print(chave, ':', valor)
```

magnifying glass Contar Frequência:

```
from collections import Counter
```

```
contagem = Counter(valores)
```

```
print(contagem)
```

ESTRUTURAS DE REPETIÇÃO

Estruturas de repetição (loops) permitem executar um bloco de código várias vezes.

Em python temos esses dois principais tipos:

- *while : Usando quando repetimos enquanto uma condição é verdadeira*
- *for : Usada quando sabemos quantas vezes queremos repetir*

Loops são fundamentais para : Processar dados, percorrer listas, aplicar transformações

LOOP FOR

For é usado para percorrer elementos de uma sequencia (string, lista, range e etc)

```
numeros = [1, 2, 3, 4]  
  
for numero in numeros:  
    print(numero)
```

```
... 1  
2  
3  
4
```

APLICAÇÕES COM FOR

Calculando uma soma com FOR:

```
valores = [10, 20, 30, 40]  
  
soma = 0  
  
for valor in valores:  
    soma += valor  
  
print("Soma:", soma)
```

```
Soma: 100
```



Loops **for** em Python

INATEL

APLICAÇÕES COM FOR

Sintaxe

```
for item in sequencia:  
    # bloco de código
```

Executa uma vez para **cada elemento** na sequência

Exemplo Simples:

```
numeros = [1, 2, 3, 4]  
for numero in numeros:  
    print(numero)
```

Saída:
1 2 3 4

Imprime números de 1 a 4

Uso Prático em Data Science:

```
valores = [10, 20, 30, 40]  
soma = 0  
for valor in valores:  
    soma += valor  
media = soma / len(valores)  
print("Média:", media)
```

Saída:
Número: 0
Número: 1
Número: 2
Número: 3
Número: 4

Calculando a média:

Util para processar e analisar dados

Iterando com Range:



- Podemos iterar listas, strings, tuplas e **ranges**
- "range" é muito útil para gerar seqüência de números
- Essencial para manipulação de dados em Python



WHILE

O while é executado enquanto uma determinada condição for verdadeira

```
contador = 0  
  
while contador < 5:  
    print(contador)  
    contador += 1
```

```
... 0  
1  
2  
3  
4
```

APLICAÇÕES COM WHILE

Calculando média com while:

```
[  ] os
  ⏎ notas = [7.5, 8.0, 6.5, 9.0]

  i = 0
  soma = 0

  while i < len(notas):
    soma += notas[i]
    i += 1

  media = soma / len(notas)

  print("Média:", media)
...
  ... Média: 7.75
```

Validação de entrada:

```
[  ] .
  ⏎ numero = int(input("Digite um número maior que 10: "))

  while numero <= 10:
    print("Valor inválido!")
    numero = int(input("Digite novamente: "))

  print("Número aceito!")

...
  ... Digite um número maior que 10: 11
  Número aceito!
```



Sintaxe

while condição:

bloco de código

→ Executa enquanto a **condição** for verdadeira



Exemplo Simples:

```
x = 0
while x < 3:
    print(x)
    x += 1
```

Imprime números de 0 a 2

Saída:

0 1 2



Simulação de Crescimento:

```
valor = 100
meta = 200
while valor < meta:
    valor *= 1.10 # crescimento de 10%
    print("Valor atual: ", round(valor, 2))
```

Usamos while para simular aumento de valor.

Saída:

Valor atual: 110.0
Valor atual: 121.0
Valor atual: 133.1
...
Valor atual: 214.36

Uso de while

Usamos **while** para simular aumento de valor.

APLICAÇÕES COM WHILE



LIST COMPREHENSION

Forma compacta e elegante de criar e transformar listas.

Ela nos permite :

- Criar listas rapidamente
- Aplicar transformações
- Filtrar valores
- Escrever menos códigos

Temos uma forma geral para definir:

[expressão for item in sequência]

EXEMPLOS LIST COMPREHENSION

Nesse exemplo vamos comparar uma aplicação básica na forma tradicional e na forma comprehensão

Forma tradicional

```
numeros = [1, 2, 3, 4]
dobrados = []
for numero in numeros:
    dobrados.append(numero * 2)
print(dobrados)
• [2, 4, 6, 8]
```

Compreensão

```
numeros = [1, 2, 3, 4]
doblados = [numero * 2 for numero in numeros]
print(dobrados)
[2, 4, 6, 8]
```

APLICAÇÃO LIST COMPREHENSION

Exemplo de filtragem de valores. Vamos criar uma lista apenas com numeros maiores que 3

```
numeros = [2, 6, 8, 3, 10, 1]  
  
maiores = [numero for numero in numeros if numero > 5]  
  
print(maiores)  
  
[6, 8, 10]
```

2 - BIBLIOTECAS IMPORTANTES

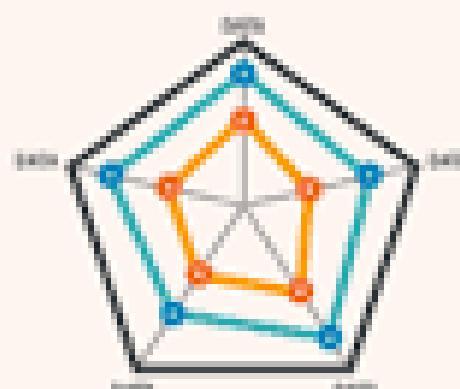
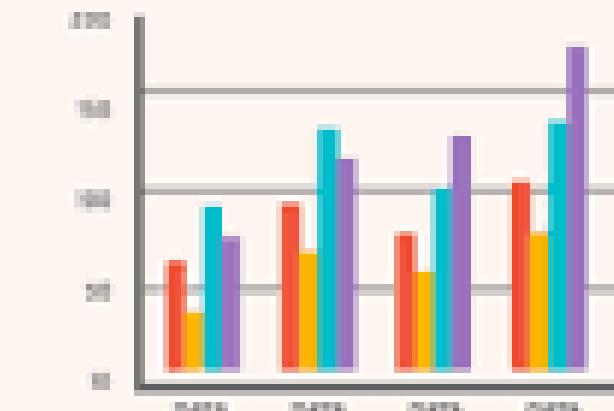
Para darmos continuidade vamos estudar três bibliotecas que são fundamentais. São elas : numpy(np), pandas(pd) e matplotlib.



NumPy



matplotlib



NUMPY

É a principal biblioteca de computação numérica. Ela permite :

- *Trabalhar com arrays (ndarrays)*
- *Realizar operações matemáticas rápidas*
- *Executar cálculos vetorizados*
- *Manipular matrizes e tensores*



NUMPY

Criação de um array

```
import numpy as np  
array = np.array([1, 2, 3, 4])  
print(array)
```

```
[1 2 3 4]
```

```
) matriz = np.array([[1, 2],  
                     [3, 4]])  
print(matriz)  
  
[[1 2]  
 [3 4]]
```

Diferente das listas, arrays são otimizados para cálculo matemático

NUMPY

Operações com numpy

Multiplicação

```
array = np.array([1, 2, 3, 4])  
dobrados = array * 2
```

```
print(dobrados)
```

```
[2 4 6 8]
```

Reshape

```
array = np.array([1, 2, 3, 4, 5, 6])  
matriz = array.reshape(2, 3)  
  
print(matriz)
```

```
[[1 2 3]  
 [4 5 6]]
```

NUMPY

Operações com numpy

```
array = np.array([10, 20, 30, 40])  
  
print("Soma:", np.sum(array))  
print("Média:", np.mean(array))  
print("Máximo:", np.max(array))  
print("Mínimo:", np.min(array))  
print("Desvio padrão:", np.std(array))
```

```
Soma: 100  
Média: 25.0  
Máximo: 40  
Mínimo: 10  
Desvio padrão: 11.180339887498949
```

O numpy é importante devido a esses fatores:

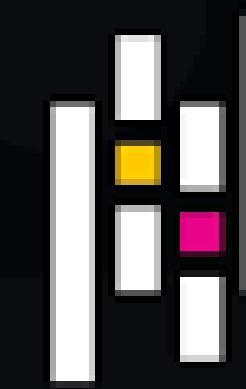
- *Extremamente rápido (implementado em C)*
- *Permite cálculos em larga escala*
- *Base de biblioteca científicas*
- *Evita uso de loops*

PANDAS

O pandas é a biblioteca responsável pela manipulação e análise de dados em py

Ela permite :

- *Trabalhar com tabelas*
- *Ler diferentes tipos de arquivos*
- *Filtrar e transformar dados*
- *Realizar análises estatísticas*
- *Organizar grandes volumes de informação*



pandas

PANDAS

O pandas possui duas estruturas principais:

- Serie : Coluna única
- DataFrame : Tabela completa(linhas e colunas)

```
▶ import pandas as pd  
dados = {  
    "Nome": ["Ana", "Pedro", "Carlos"],  
    "Idade": [21, 25, 30],  
    "Nota": [8.5, 7.0, 9.2]  
}  
  
df = pd.DataFrame(dados)  
print(df)
```

	Nome	Idade	Nota
0	Ana	21	8.5
1	Pedro	25	7.0
2	Carlos	30	9.2

PANDAS

Temos algumas importantes aplicações:

Leitura de um determinado arquivo

```
df = pd.read_csv("dados.csv")
```

Estatística básica

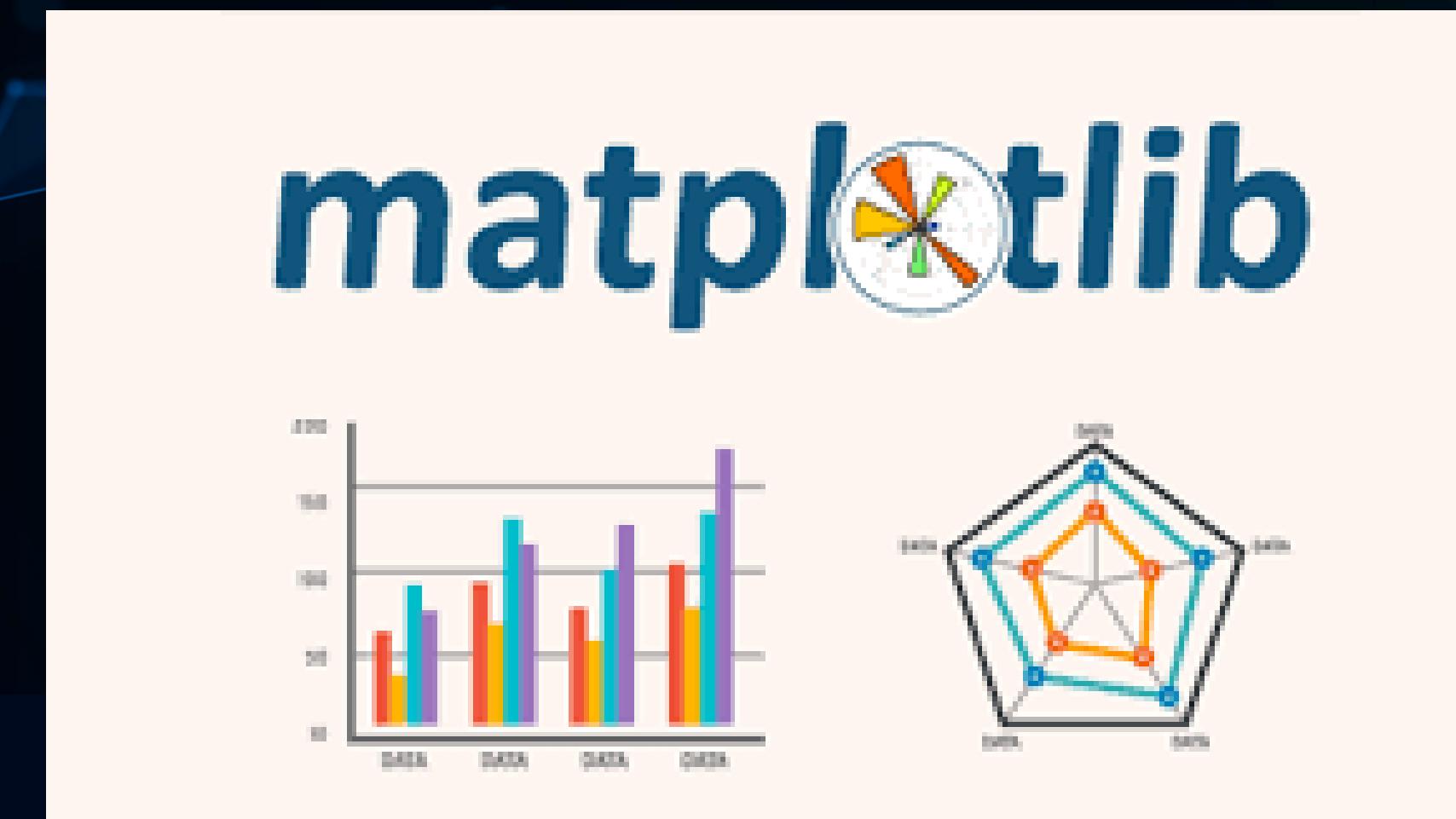
```
df.describe()
```

	Idade	Nota
count	3.000000	3.000000
mean	25.333333	8.233333
std	4.509250	1.123981
min	21.000000	7.000000
25%	23.000000	7.750000
50%	25.000000	8.500000
75%	27.500000	8.850000
max	30.000000	9.200000

MATPLOTLIB

Principal biblioteca de visualização de dados:

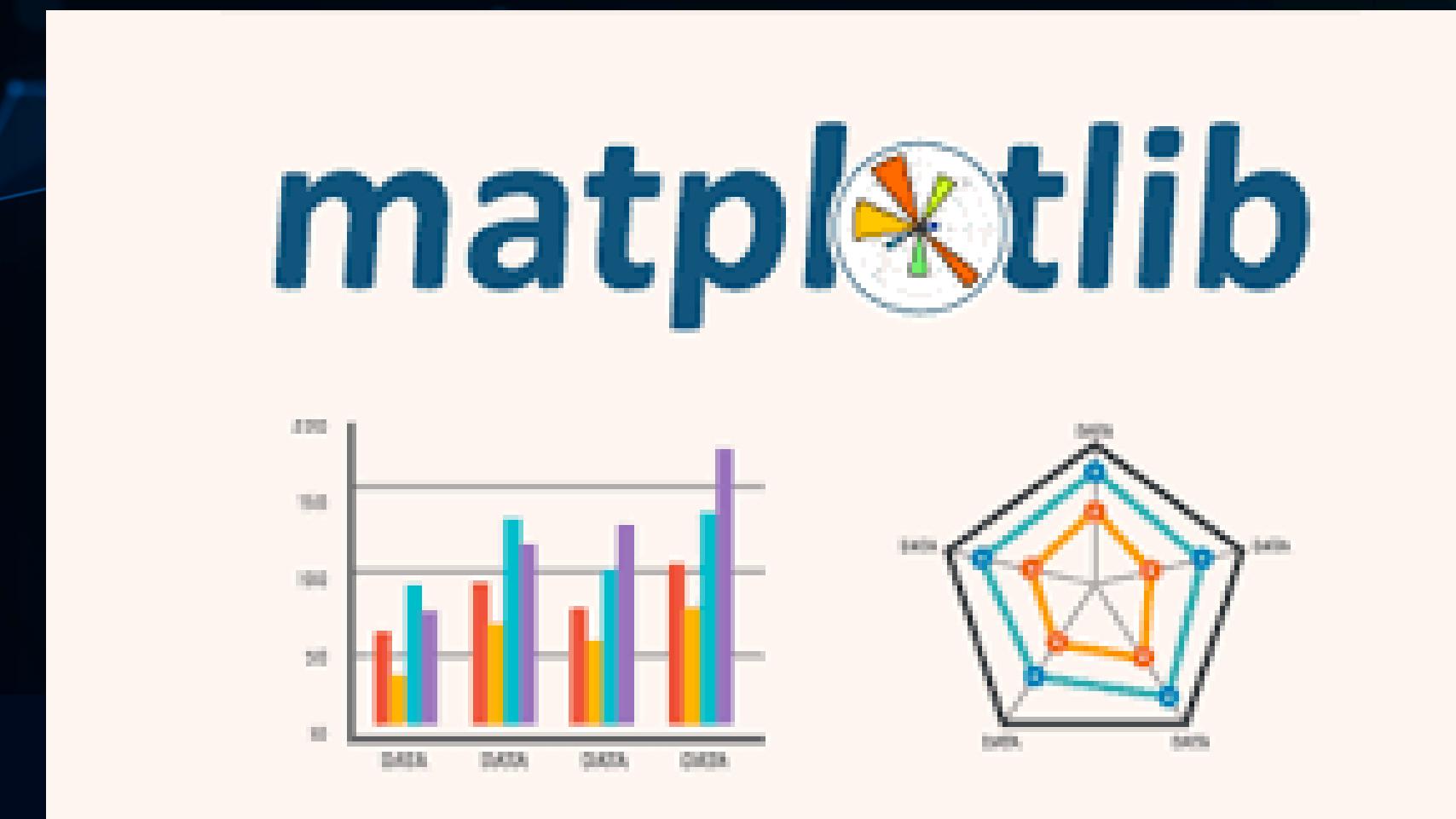
- *Permite criar gráficos*
- *Visualizar padrões*
- *Comparar informações*



MATPLOTLIB

Principal biblioteca de visualização de dados:

- *Permite criar gráficos*
- *Visualizar padrões*
- *Comparar informações*



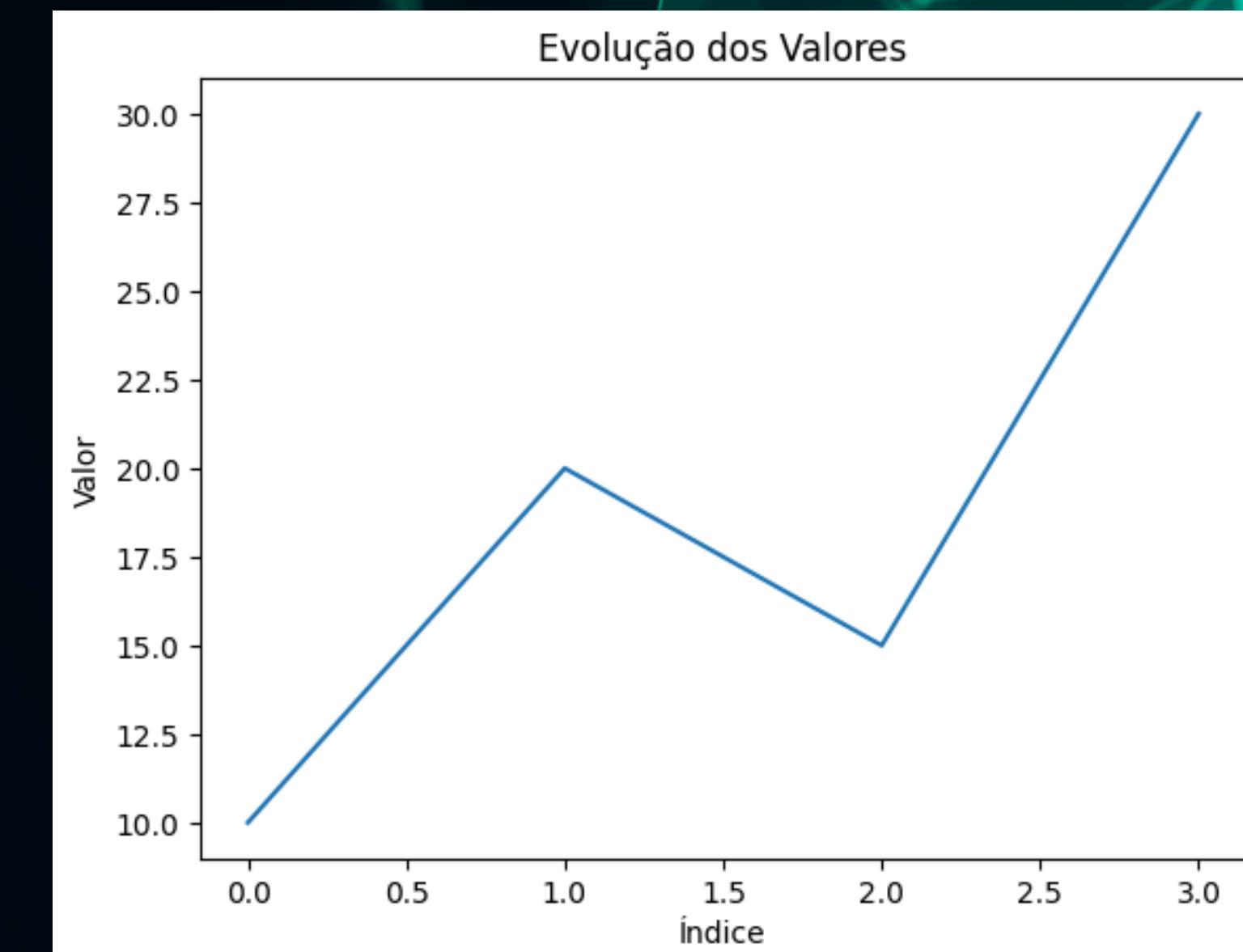
MATPLOTLIB

Grafico em linha (mostrar evolução ao longo do tempo)

```
import matplotlib.pyplot as plt

valores = [10, 20, 15, 30]

plt.plot(valores)
plt.title("Evolução dos Valores")
plt.xlabel("Índice")
plt.ylabel("Valor")
plt.show()
```

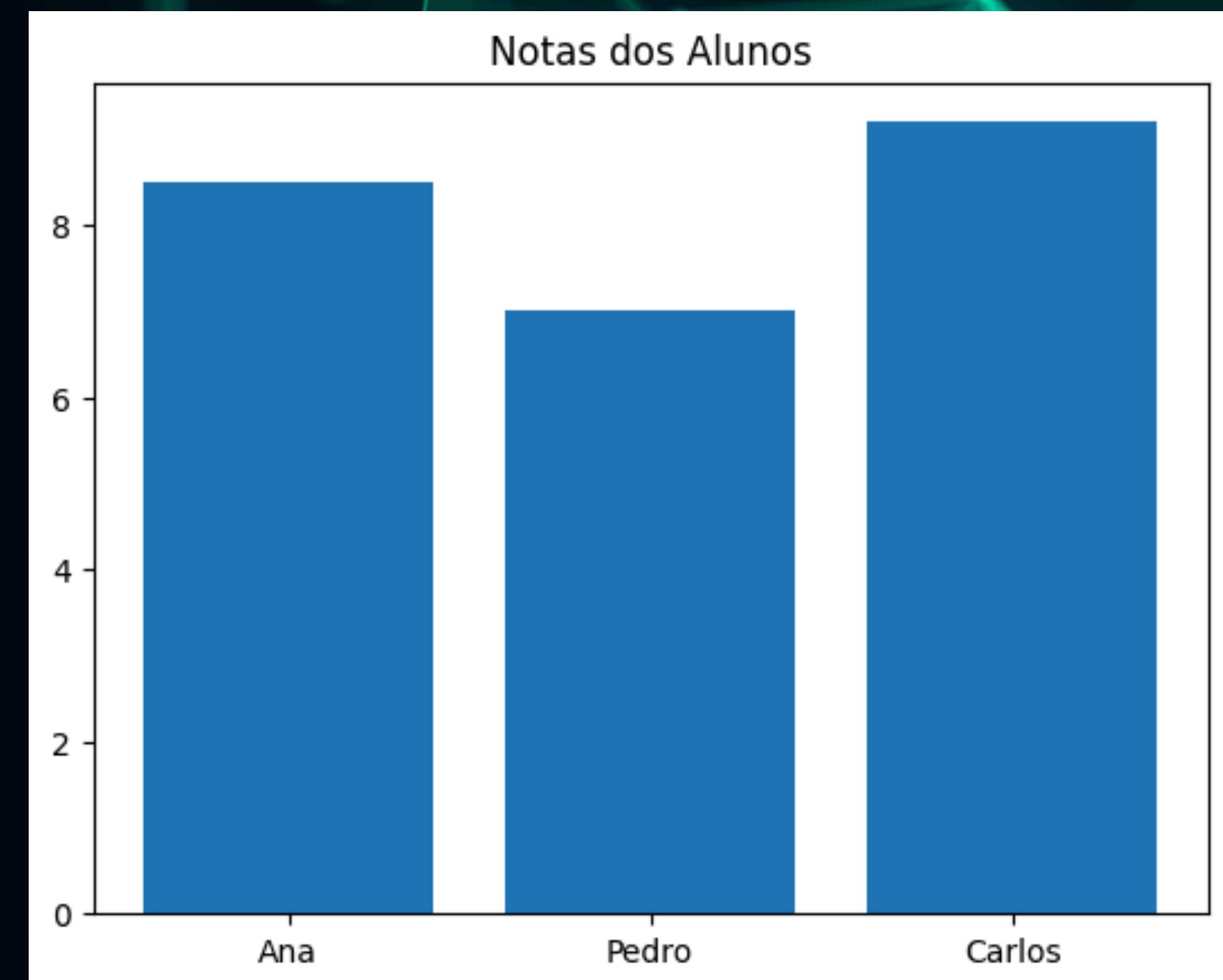


MATPLOTLIB

Gráfico de barras (Bar plot), usado para comparar categorias

```
nomes = ["Ana", "Pedro", "Carlos"]
notas = [8.5, 7.0, 9.2]

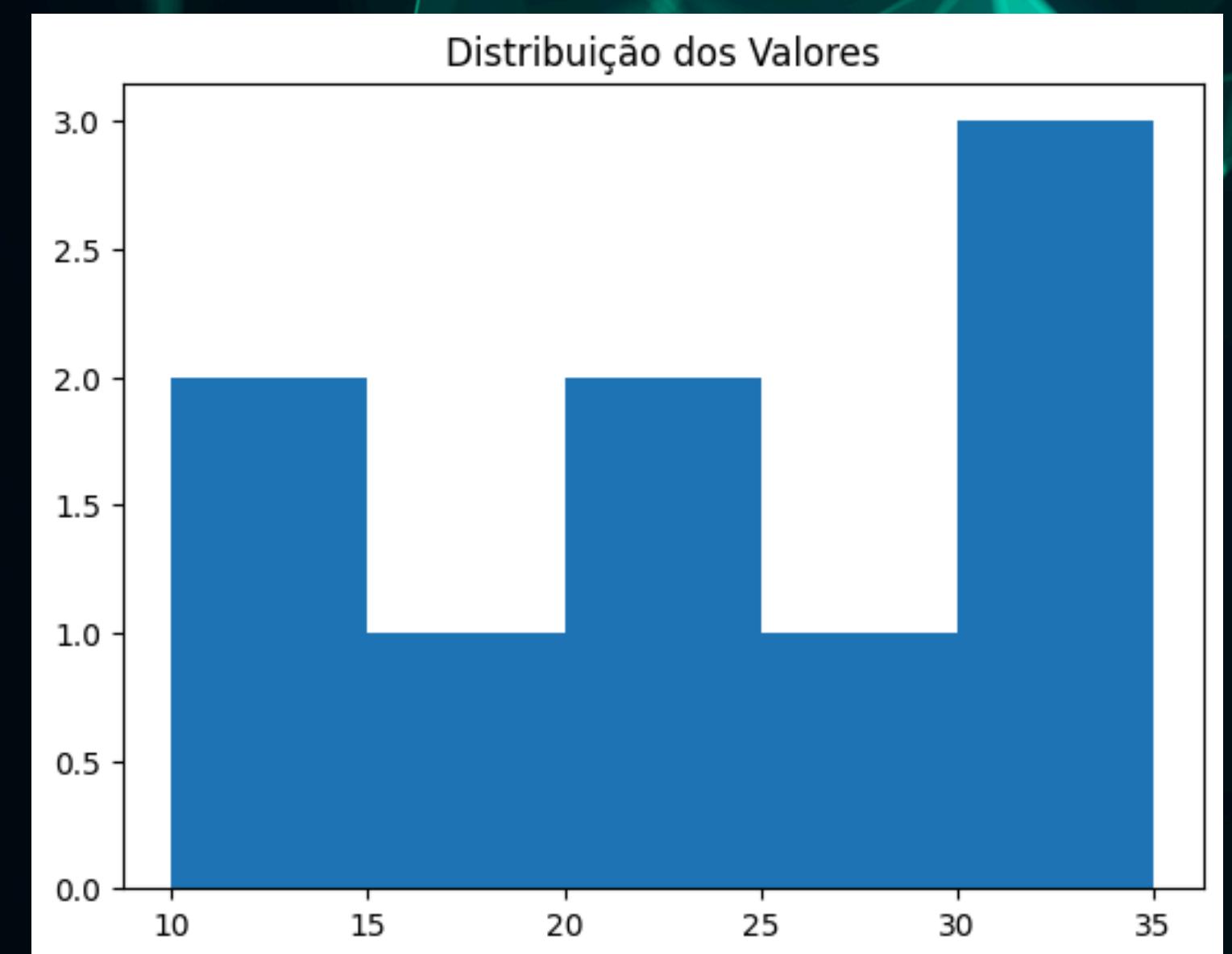
plt.bar(nomes, notas)
plt.title("Notas dos Alunos")
plt.show()
```



MATPLOTLIB

Histograma, indicar a distribuição dos dados

```
valores = [10, 12, 15, 20, 22, 25, 30, 30, 35]  
  
plt.hist(valores, bins=5)  
plt.title("Distribuição dos Valores")  
plt.show()
```

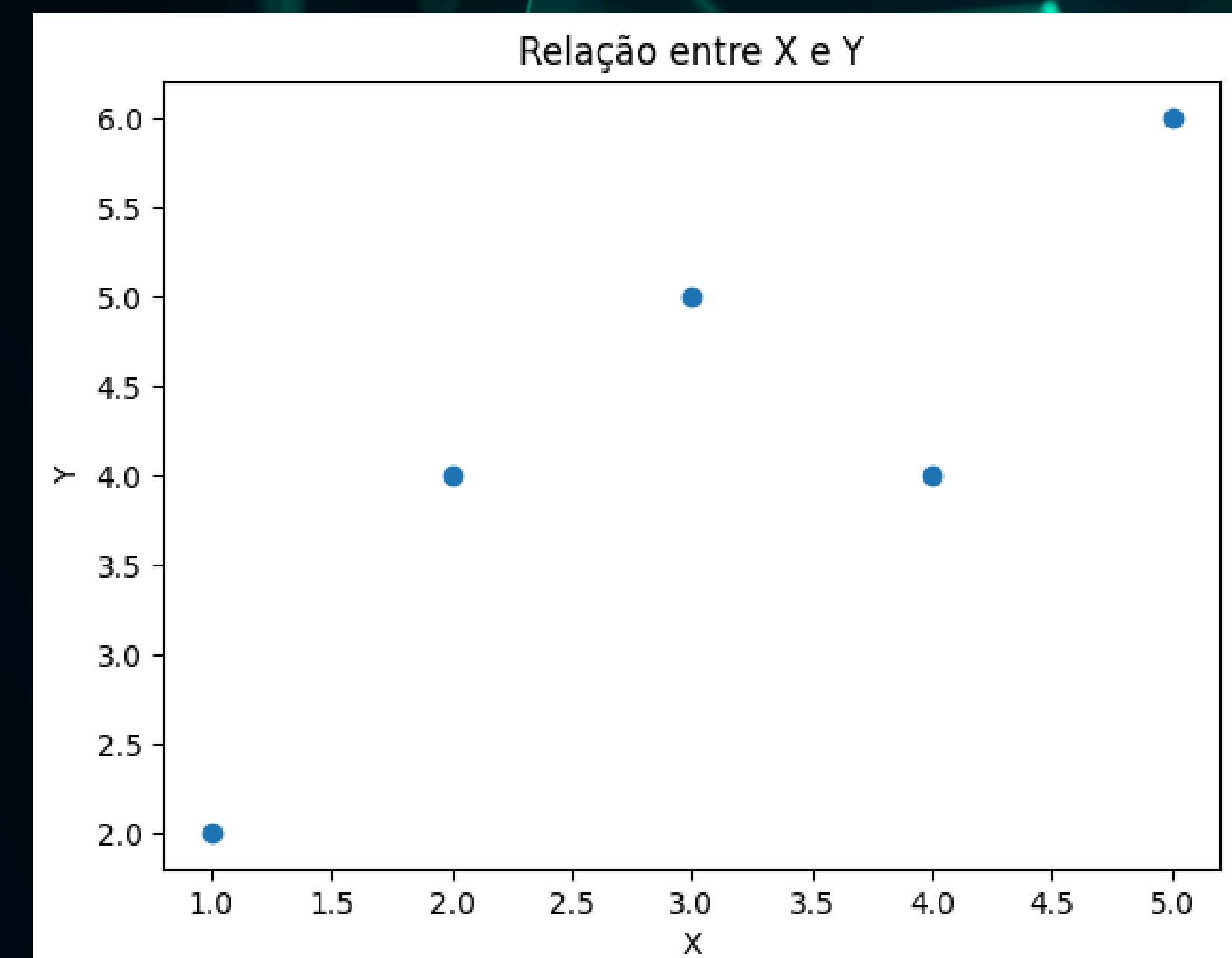


MATPLOTLIB

Dispersão, analisar a relação entre duas variáveis

```
x = [1, 2, 3, 4, 5]
y = [2, 4, 5, 4, 6]

plt.scatter(x, y)
plt.title("Relação entre X e Y")
plt.xlabel("X")
plt.ylabel("Y")
plt.show()
```



3 - EDA

EDA (Exploratory Data Analysis) é um processo de explorar e entender um dataset antes de aplicar os modelos de ML

Os objetivos do EDA, são:

- *Entender comportamento dos dados*
- *Identificar padrões*
- *Detectar erros e inconsistências*
- *Descobrir Insights*
- *Preparar os dados para modelagem*

CHECKLIST DE UM EDA

1. Entender contexto do dataset
2. Inspecionar estruturas dos dados
3. Verificar valores ausentes
4. Gerar resumo estatístico
5. Analisar variáveis categóricas
6. Criar visualizações

PERGUNTAS IMPORTANTES QUE DEVEMOS NOS PERGUNTAR

- O que esse dataset representa?
- Qual é o problema que queremos resolver ?
- Qual é a variável alvo?
- As variáveis fazem sentido?
- Existe algum viés?

Sem contexto, a análise perde significado.

3 - EDA

Comandos básicos de um EDA

Primeiro precisamos inspecionar a estrutura do problema

```
#Com o head visualizamos as primeiras linhas  
df.head()
```

	Nome	Idade	Nota
0	Ana	21	8.5
1	Pedro	25	7.0
2	Carlos	30	9.2

#Estrutura

df.shape

(3, 3)

3 - EDA

Comandos básicos de um EDA

Primeiro precisamos inspecionar a estrutura do problema

```
#Verificando dados faltantes
df.isnull().sum()

      0
Nome  0
Idade 0
Nota  0
dtype: int64
```

```
df.describe()

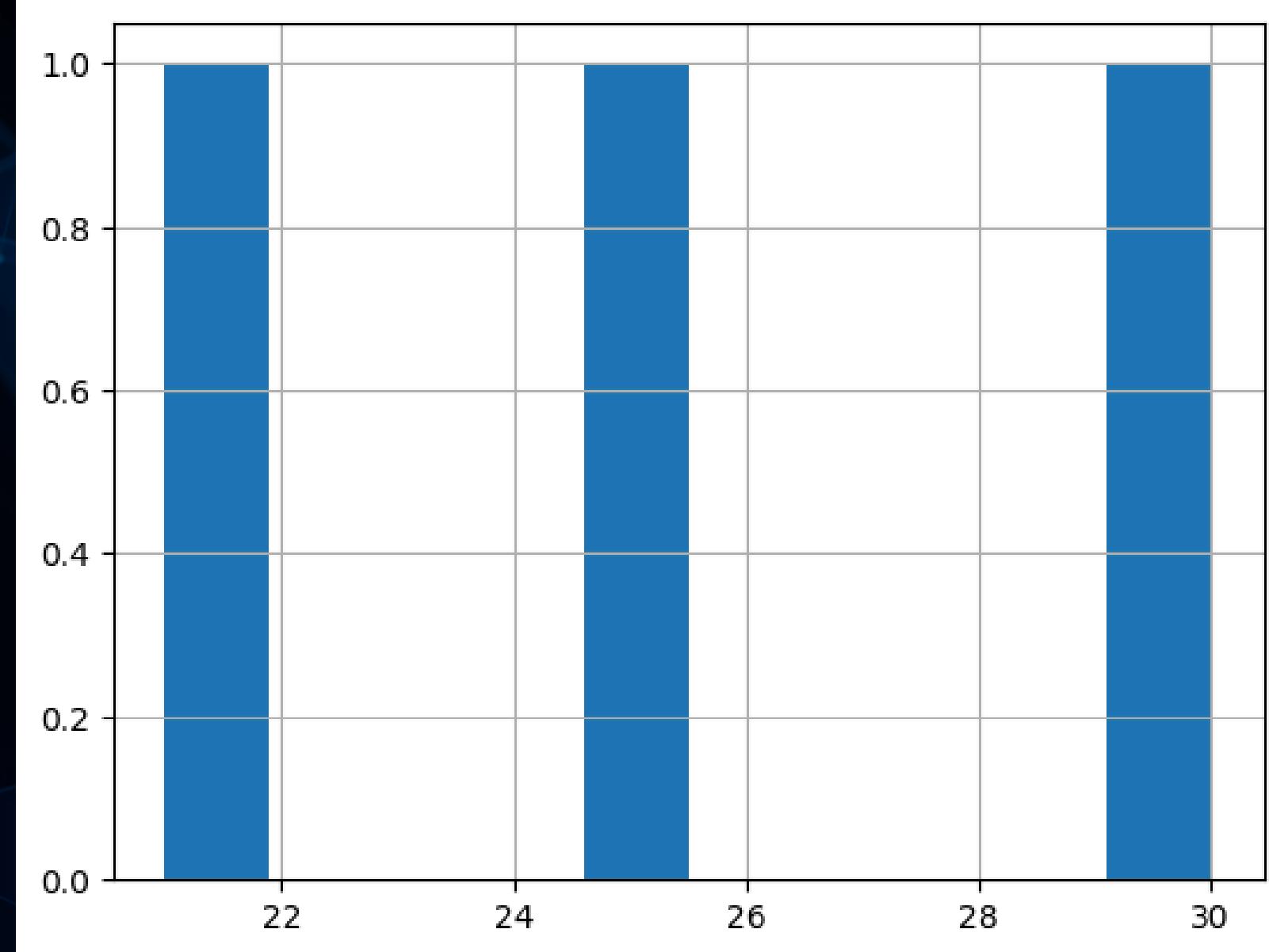
           Idade      Nota
count    3.000000  3.000000
mean     25.333333  8.233333
std      4.509250  1.123981
min     21.000000  7.000000
25%    23.000000  7.750000
50%    25.000000  8.500000
75%    27.500000  8.850000
max    30.000000  9.200000
```

3 - EDA

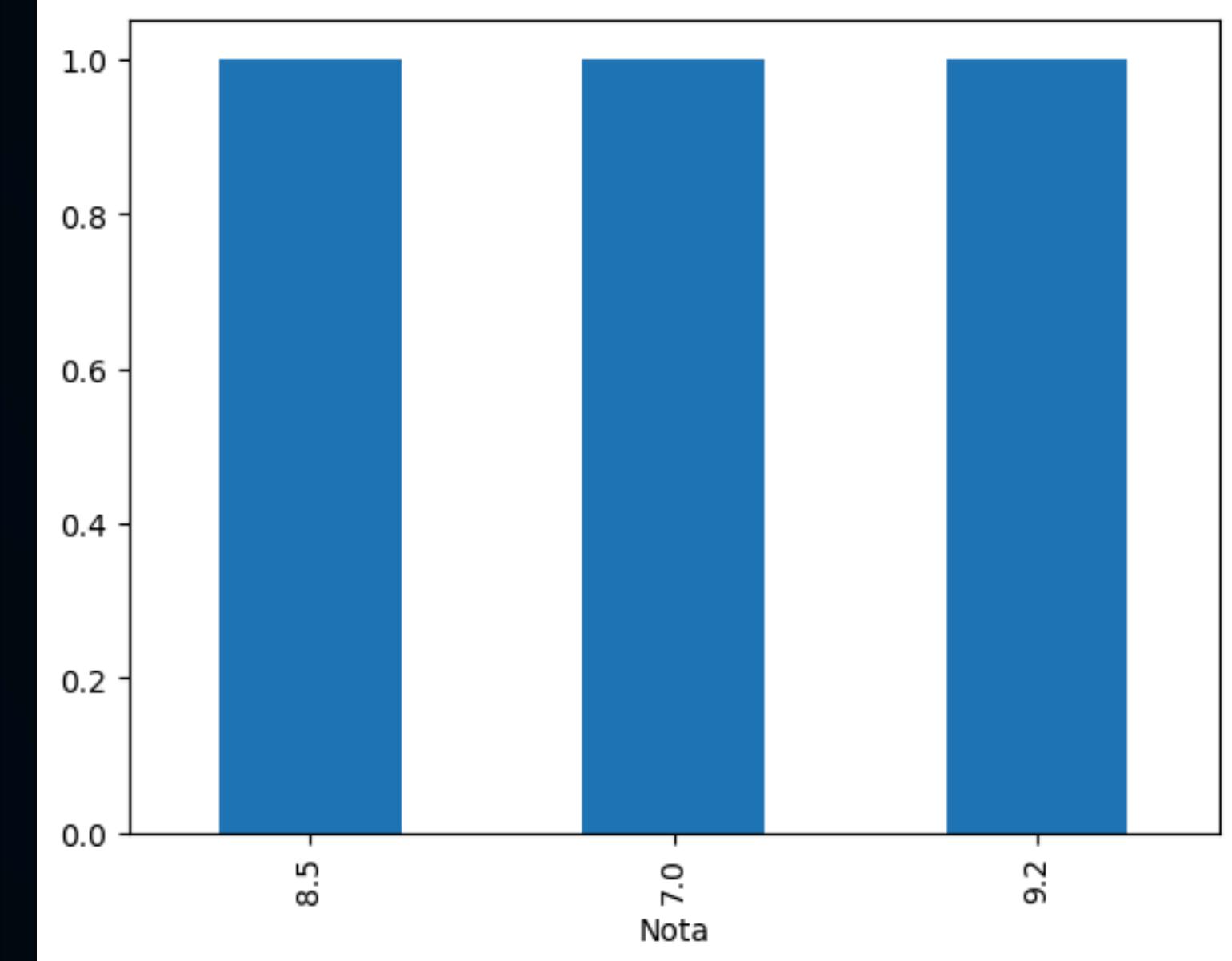
Comandos básicos de um EDA

Criando visualizações

```
df["Idade"].hist()
```



```
df["Nota"].value_counts().plot(kind="bar")
```



MATERIAL

Nesse link temos acesso ao github da aula, com
codigo

→ O exercico final esta no notebook