

PYTHON PARA CIÊNCIA DE DADOS



# Variáveis Categóricas em Python

Dados categóricos representam grupos, classes ou rótulos finitos — não valores contínuos. Em Python, usamos `pandas.Categorical` para trabalhar com esse tipo de dado de forma eficiente e semanticamente correta.



Eduardo Ogasawara

[eduardo.ogasawara@cefet-rj.br](mailto:eduardo.ogasawara@cefet-rj.br)

<https://eic.cefet-rj.br/~eogasawara>

# O Que São Dados Categóricos?

## Conceito Fundamental

Dados categóricos pertencem a um conjunto finito de categorias possíveis. Ao invés de representar um contínuo numérico, eles indicam rótulos ou grupos específicos.

Em pandas, o tipo `Categorical` traz semântica explícita de categorias e é mais eficiente que strings repetidas, economizando memória e processamento.

## Exemplo Básico

```
import pandas as pd

cores = pd.Categorical([
    "vermelho",
    "azul",
    "verde",
    "azul"
])

cores
```

- ❑ **Saída:** ['vermelho', 'azul', 'verde', 'azul']  
Categories (3, object): ['azul', 'verde', 'vermelho']

## Estrutura Interna: Códigos e Categorias

### Códigos Inteiros

Internamente, um `Categorical` armazena valores como códigos inteiros compactos, economizando memória significativamente quando há repetições.

`cores.codes`

**Saída:** `array([2, 0, 1, 0], dtype=int8)`

### Lista de Categorias

As categorias reais ficam armazenadas separadamente no atributo `categories`, explicitando o "universo" completo de valores possíveis.

`cores.categories`

**Saída:** `Index(['azul', 'verde', 'vermelho'], dtype='object')`

## Categorias Não Ordenadas vs. Ordenadas



### Não Ordenadas

Categorias sem hierarquia lógica — como cores ou tipos de produto.  
Não há relação de "maior" ou "menor" entre elas.



### Ordenadas

Categorias com ordem lógica definida, como níveis de satisfação (baixo < médio < alto). Permitem comparações e ordenação corretas.

Em pandas, definimos a ordem usando o parâmetro `ordered=True` ao criar o categórico:

```
nivel = pd.Categorical(  
    ["médio", "baixo", "alto"],  
    categories=["baixo", "médio", "alto"],  
    ordered=True  
)  
nivel
```

- Saída:** ['médio', 'baixo', 'alto']  
Categories (3, object): ['baixo' < 'médio' < 'alto']

# Criando Categoria Ordenada: Níveis de Dor

Valores numéricos frequentemente representam níveis discretos em escalas. Transformá-los em categorias ordenadas explicita a hierarquia e melhora a análise.



## Código de Exemplo

```
pain = [0, 3, 2, 2, 1]  
  
fpain = pd.Categorical(  
    pain,  
    categories=[0, 1, 2, 3],  
    ordered=True  
)  
  
fpain
```

## Resultado

### Saída:

```
[0, 3, 2, 2, 1]  
Categories (4, int64): [0 < 1  
< 2 < 3]
```

A ordem fica explícita através do símbolo < entre as categorias.

# Adicionando Rótulos Descritivos

1

## Níveis Numéricicos

Começamos com valores de 0 a 3 representando intensidade

```
pain = [0, 3, 2, 2, 1]
mapa = {0: "sem", 1: "baixa", 2: "média", 3: "alta"}
pain_txt = [mapa[x] for x in pain]
```

```
fpain_lbl = pd.Categorical(
    pain_txt,
    categories=["sem", "baixa", "média", "alta"],
    ordered=True
)
fpain_lbl
```

2

## Mapeamento

Criamos um dicionário que converte números em textos descritivos

3

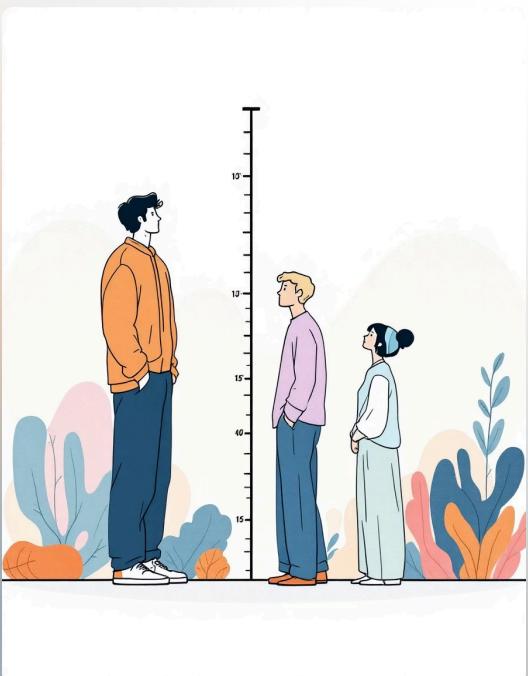
## Categorias Textuais

Resultado final mantém ordem lógica com rótulos intuitivos

- **Saída:** ['sem', 'alta', 'média', 'média', 'baixa']  
Categories (4, object): ['sem' < 'baixa' < 'média' < 'alta']

# Caso Prático: Classificação por Altura

Vamos classificar pessoas em categorias de altura usando dados reais. As classes seguem ordem lógica: **baixa, média e alta**.



## Dados de Entrada

```
import numpy as np  
  
weight = np.array([  
    60, 72, 57, 90, 95, 72  
])  
  
height = np.array([  
    1.75, 1.80, 1.65,  
    1.90, 1.74, 1.91  
])  
  
subject = np.array([  
    "A", "B", "C",  
    "D", "E", "F"  
])
```

## Vetor de Alturas

```
height
```

**Saída:**

```
array([1.75, 1.8, 1.65, 1.9, 1.74, 1.91])
```

Seis medidas em metros que serão categorizadas segundo critérios definidos.

# Método 1: Lógica Condicional



## Loop com Condições

Iteramos sobre cada altura aplicando regras if-elif-else para determinar a categoria.



## Testes de Limites

Altura < 1.7m = baixa; entre 1.7 e 1.9m = média; acima de 1.9m = alta.



## Geração de Rótulos

Cada teste resulta em um rótulo textual adicionado à lista de categorias.

```
labels = []
for h in height:
    if h < 1.7:
        labels.append("baixa")
    elif h < 1.9:
        labels.append("média")
    else:
        labels.append("alta")
```

```
labels
```

- Saída:** ['média', 'média', 'baixa', 'alta', 'média', 'alta']

## Convertendo em Categoria Ordenada

### Aplicando Ordem Semântica

Após gerar os rótulos textuais, transformamos a lista em um objeto Categorical ordenado.

A ordem explícita permite comparações corretas e análises mais sofisticadas dos dados categóricos.

```
height_cat = pd.Categorical(  
    labels,  
    categories=[  
        "baixa",  
        "média",  
        "alta"  
    ],  
    ordered=True  
)  
height_cat
```

- ❑ **Saída:** ['média', 'média', 'baixa', 'alta', 'média', 'alta']  
Categories (3, object): ['baixa' < 'média' < 'alta']

## Método 2: pd.cut() – Cortes por Intervalos



### Vetorizado

Evita loops explícitos, processando todos os valores de uma vez com alta performance.



### Ajustável

Pontos de corte são facilmente modificáveis através do parâmetro `bins`.



### Automático

O resultado já nasce como objeto categórico ordenado, pronto para análise.

```
height_cat2 = pd.cut(  
    height,  
    bins=[0, 1.7, 1.9, np.inf],  
    labels=["baixa", "média", "alta"],  
    ordered=True  
)  
height_cat2
```

- ❑ **Saída:** ['média', 'média', 'baixa', 'média', 'média', 'alta']  
Categories (3, object): ['baixa' < 'média' < 'alta']

# Vantagens do Método por Intervalos



## Performance Superior

Ideal para grandes volumes de dados, pois opera de forma vetorizada sem loops Python explícitos.



## Flexibilidade

Os pontos de corte (bins) são facilmente ajustáveis, permitindo experimentação rápida com diferentes classificações.



## Resultado Direto

O output já é um objeto categórico ordenado, eliminando etapas adicionais de conversão.

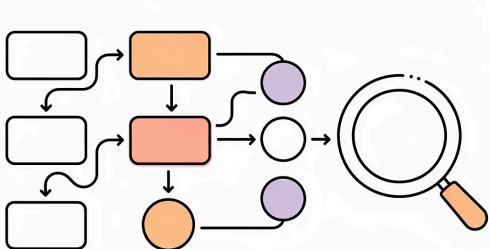
## Contagem de Valores

```
pd.value_counts(height_cat2)
```

▢ **Saída:**  
média 4  
baixa 1  
alta 1  
dtype: int64

## Fluxo de Trabalho Real: Conversão e Inspeção

Na prática, dados frequentemente já existem como textos em DataFrames. Podemos convertê-los facilmente para categorias e inspecionar sua estrutura interna.



```
import pandas as pd

s = pd.Series([
    "baixa",
    "média",
    "alta",
    "média"
])

s_cat = s.astype("category")
s_cat
```

### ▢ Saída da Série:

```
0    baixa
1    média
2    alta
3    média
dtype: category
Categories (3, object): ['alta', 'baixa', 'média']
```

### Inspeção Interna

```
s_cat.cat.categories
```

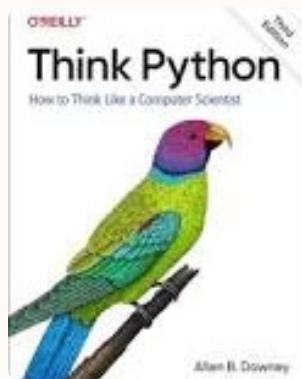
```
▢ Index(['alta', 'baixa', 'média'],
       dtype='object')
```

```
s_cat.cat.codes
```

```
▢ array([1, 2, 0, 2], dtype=int8)
```

Esta inspeção revela que categoria não é apenas string — possui estrutura otimizada e semântica própria.

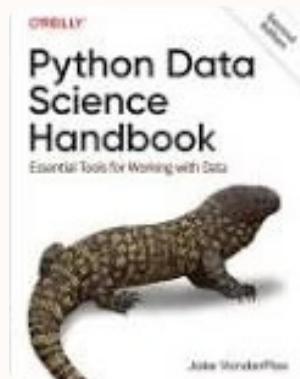
# Referências



## Think Python

Downey, A. *Think Python: How to Think Like a Computer Scientist*. O'Reilly Media.

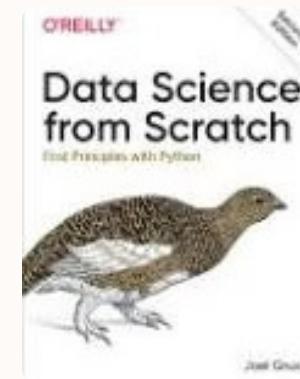
An essential introduction to programming fundamentals and computational thinking using Python.



## Python Data Science Handbook

VanderPlas, J. *Python Data Science Handbook*. O'Reilly Media.

A comprehensive guide to essential tools for working with data in Python, including NumPy, Pandas, and visualization libraries.



## Data Science from Scratch

Grus, J. *Data Science from Scratch*. O'Reilly Media.

Learn data science fundamentals by building algorithms and tools from the ground up using Python.