1. Geoffrey Hinton e o Deep Learning (Aprendizagem Profunda)

20% Essencial para Entender 80%:

- **O que é?**
  Deep Learning é um tipo de **aprendizado de máquina** que usa redes neurais com **múltiplas camadas** para aprender padrões complexos.
- **Por que revolucionou?**
  Em 2006, Hinton mostrou que redes neurais profundas podiam reconhecer dígitos escritos à mão com **98% de precisão**, algo quase impossível antes.
- **Analogia:**
  Imagine uma fábrica de classificação de cartas:
  - 1ª camada: Separa letras por tamanho.
  - 2ª camada: Identifica curvas e retas.
  - 3ª camada: Reconhece o número (ex: "5").
    Quanto mais camadas, mais detalhes a rede "enxerga".
- **3 Termos-Chave:**
  1. **Neurônio Artificial:** Unidade básica que processa informações (como um mini-decisor).
  2. **Camadas Ocultas:** Etapas intermediárias onde o modelo "aprende" padrões.
  3. **Treinamento:** Ajuste dos neurônios usando exemplos (dados).

---

2. Aplicações do Aprendizado de Máquina

20% Essencial para Entender 80%:

- **Segmentação de clientes:**
  - **O que faz?** Agrupa clientes com comportamentos similares (ex: quem compra livros de ficção).
  - **Analogia:** Organizar amigos em grupos no WhatsApp (esportistas, cinéfilos, gamers).
- **Detecção de fraudes:**

    - **Como funciona?** Identifica transações fora do padrão (ex: compra de R$ 10.000 em outro país).
    - **Exemplo:** Se você só compra pães na padaria, um gasto de R$ 500 em joias aciona um alerta.
- **Recomendações (ex: filmes):**
    - **Base:** Compara seus gostos com os de usuários parecidos.
    - **Analogia:** Um amigo que sempre indica séries baseado no que você já assistiu.

---

3. O que é Aprendizado de Máquina?

20% Essencial para Entender 80%:

- **Definição Simplificada:**
  AM é ensinar computadores a **aprender com dados** (ex: fotos, números, textos) em vez de seguir regras fixas.
- **Exemplo Clássico:**
    - **Algoritmo Tradicional:** Regra fixa para spam: "Se o e-mail tem a palavra 'oferta', bloqueie".
    - **AM:** O computador analisa milhares de e-mails marcados como spam e **descobre sozinho** padrões (ex: palavras como "grátis", "urgente").
- **Analogia:**
    - **Programação Tradicional:** Seguir uma receita de bolo passo a passo.
    - **AM:** Aprender a fazer bolo errando e acertando (ajustando ingredientes até ficar bom).
- **3 Termos-Chave:**
    1. **Dados de Treinamento:** Exemplos usados para ensinar o modelo (ex: e-mails marcados como spam).
    2. **Modelo:** "Cérebro" do sistema que faz previsões (ex: classificar spam).
    3. **Aprendizado:** Processo de ajustar o modelo para reduzir erros.

---

4. AM vs. Inteligência Artificial (IA)

20% Essencial para Entender 80%:

- **Relação:**
    - **IA é o guarda-chuva:** Qualquer sistema que imita inteligência humana (ex: jogar xadrez).
    - **AM é uma parte da IA:** Foca em aprender com dados (ex: reconhecer rostos em fotos).
- **Analogia:**
    - **IA:** Um carro autônomo.
    - **AM:** A habilidade do carro de aprender a evitar obstáculos analisando milhares de horas de direção.

Resumo dos 20% que mais Importam:

1. **Deep Learning** = Redes neurais com muitas camadas para tarefas complexas (ex: reconhecer imagens).
2. **Aplicações de AM** resolvem problemas reais agrupando dados, detectando anomalias ou fazendo recomendações.
3. **AM é aprender com dados**, não seguir regras fixas.
4. **AM é um subconjunto da IA** focado em aprendizado autônomo.

---

1. Por que usar Aprendizado de Máquina (AM) em vez de algoritmos tradicionais?

Comparação Direta (20% Essencial):

Critério	Algoritmos Tradicionais	Aprendizado de Máquina
Regras	Você define regras **manualmente** (ex: "Se X, faça Y").	O sistema **aprende regras** sozinho com dados.
Exemplo	Um filtro de spam com regras fixas (ex: bloquear e-mails com "oferta").	O filtro de spam analisa milhares de e-mails para **descobrir padrões** de spam.
Vantagem	Previsível e eficiente para problemas **simples**.	Lida com problemas **complexos** (ex: reconhecer rostos, prever tendências).
Quando usar?	Tarefas estáticas e bem definidas (ex: calcular média).	Tarefas com **muitos dados** ou padrões ocultos.

Analogia:

- **Algoritmo tradicional:** Seguir uma receita de bolo passo a passo.
- **AM:** Aprender a fazer bolo experimentando ingredientes e ajustando até acertar.

---

2. Passos do AM vs. Algoritmos Tradicionais (Baseado nas Imagens)

am01.png (Algoritmos Tradicionais):

1. **Escrever regras:** Definir manualmente como o sistema deve funcionar.
2. **Estudar o problema:** Entender a lógica por trás da tarefa.
3. **Analisar erros:** Corrigir falhas nas regras pré-definidas.
4. **Avaliar:** Verificar se o resultado está correto.

am02.png (Aprendizado de Máquina):

1. **Treinar o algoritmo:** Alimentar o modelo com dados (ex: imagens de dígitos).
2. **Estudar o problema:** Escolher o modelo certo (ex: rede neural).
3. **Analisar erros:** Ajustar o modelo para reduzir falhas (ex: otimizar pesos).
4. **Avaliar:** Testar a precisão do modelo com novos dados.

Diferença Chave:

- **Tradicional:** Você programa **regras explícitas**.
- **AM:** Você programa a **capacidade de aprender**.

---

3. Conceitos de Agentes e Problemas de IA (20% Essencial)

Resumo Visual:

Copy

[Agente]
|
▼
[Estado Inicial] → [Ações Possíveis] → [Modelo de Transição]
|
▼
[Teste de Objetivo] → [Solução (ótima ou não)]

Explicação Simplificada:

1. **Agente:** Um sistema que age no ambiente (ex: robô aspirador).
2. **Estado:** Situação atual (ex: robô está na sala A).
3. **Ações:** O que o agente pode fazer (ex: mover para a sala B).
4. **Modelo de Transição:** O que acontece após uma ação (ex: se mover para B, o estado muda).
5. **Solução Ótima:** Caminho mais eficiente (ex: menor tempo para limpar a casa).

Analogia (GPS de Carro):

- **Estado inicial:** Carro em São Paulo.
- **Ações:** Virar à esquerda, seguir em frente, etc.
- **Objetivo:** Chegar ao Rio de Janeiro.
- **Solução ótima:** Rota mais rápida (menor custo de tempo).

---

4. Principais Takeaways (20% para 80% de Entendimento):

1. **Use AM quando:**
    - O problema é complexo (ex: reconhecer voz).
    - Você tem muitos dados para treinar o modelo.
2. **Use algoritmos tradicionais quando:**
    - O problema é simples e bem definido (ex: calcular juros).
3. **Agentes de IA:**
    - Agem em um ambiente, buscam soluções com menor custo.
    - A solução ótima depende do modelo de transição e ações possíveis.

Exemplo Prático:

Problema: Um robô deve encontrar a saída de um labirinto.

- **Estado inicial:** Robô no início do labirinto.
- **Ações:** Mover para frente, virar à direita/esquerda.
- **Custo:** Número de movimentos.
- **Solução ótima:** Menor caminho até a saída.

Como o AM ajudaria?
Treinando o robô com simulações de labirintos para ele aprender a evitar becos sem saída! 🤓

---

1. Por que usar Aprendizado de Máquina (AM) em vez da forma tradicional de escrever algoritmos?

20% Essencial:

- **Programação Tradicional:**
    - Você define **regras fixas** para resolver um problema (ex: "Se a temperatura > 30°C, ligue o ventilador").
    - Funciona bem para problemas **simples e previsíveis**.

- **Aprendizado de Máquina:**
    - O computador **aprende regras sozinho** analisando dados (ex: analisa milhares de dias de temperatura para decidir quando ligar o ventilador).
    - Ideal para problemas **complexos ou com padrões ocultos** (ex: reconhecer rostos em fotos).

Analogia:

- **Tradicional:** Seguir uma receita de bolo passo a passo.
- **AM:** Aprender a fazer bolo experimentando ingredientes e ajustando até acertar.

Quando usar AM?

✅ Problemas com **muitos dados** (ex: prever vendas).
✅ Tarefas que exigem **adaptação** (ex: detectar fraudes em tempo real).
❌ Problemas **simples e estáticos** (ex: calcular a média de notas).

---

2. Conceitos Adicionais (Agente, Estado, Ações, etc.)

20% Essencial:

Esses conceitos são a base para entender como um **agente de IA** toma decisões. Vamos simplificar:

Termo	Definição Simplificada	Exemplo do Dia a Dia
Agente	"Robô" que interage com um ambiente e toma decisões.	Um carro autônomo.
Estado	Situação atual do agente e do ambiente.	Carro está a 60 km/h na rodovia.
Ação	Escolha que o agente pode fazer em um estado.	Acelerar, frear, ou virar.
Modelo de Transição	Previsão do que acontece após uma ação.	Se acelerar, o carro vai para 70 km/h.
Espaço de Estados	Todas as situações possíveis que o agente pode estar.	Todas as combinações de velocidade e localização do carro.
Objetivo	Meta que o agente quer alcançar.	Chegar ao destino em 1 hora.
Solução Ótima	Melhor caminho (menor custo) para alcançar o objetivo.	Rota mais rápida e sem trânsito.

Analogia Prática:

Imagine que você é um **agente** tentando cozinhar um ovo perfeito:

- **Estado inicial:** Ovo cru na frigideira.
- **Ações:** Aumentar o fogo, virar o ovo, retirar da panela.
- **Modelo de transição:** Se você aumentar o fogo, o ovo queima em 2 minutos.
- **Objetivo:** Ovo frito com gema mole.
- **Solução ótima:** Fritar por 3 minutos em fogo médio.

---

3. Resumindo... (Problemas para IA)

20% Essencial:

Todo problema de IA precisa definir:

1. **Estado inicial:** Onde o agente começa (ex: posição inicial em um labirinto).
2. **Ações possíveis:** O que o agente pode fazer (ex: andar para frente, virar à direita).

3. **Modelo de transição:** Como o ambiente muda após cada ação (ex: andar para frente leva à próxima sala).
4. **Teste de objetivo:** Como saber se o problema foi resolvido (ex: saiu do labirinto).
5. **Custo da solução:** Quanto "esforço" foi gasto (ex: número de movimentos).

Exemplo de Aplicação:

Problema: Encontrar o caminho mais curto de São Paulo ao Rio de Janeiro.

- **Estado inicial:** Carro em SP.
- **Ações:** Pegar a Via Dutra, Rodovia Carvalho Pinto, etc.
- **Custo:** Distância em quilômetros.
- **Solução ótima:** Via Dutra (menor distância).

Principais Takeaways (20% para 80% de entendimento):

1. **AM vs. Tradicional:**
    - Use AM quando o problema é complexo e tem muitos dados.
    - Use algoritmos tradicionais para tarefas simples e previsíveis.
2. **Agentes e Estados:**
    - Um agente toma decisões baseadas em seu estado atual e ações possíveis.
    - A solução ótima é o caminho de menor custo para o objetivo.
3. **Custo e Solução:**
    - Toda ação tem um custo, e a IA busca minimizá-lo (ex: tempo, energia).

---

1. Como as máquinas aprendem?

Analogia:

Imagine que a máquina é um **estudante**, e os **dados** são os livros que ela lê.

- **Modelos estatísticos** são como o "método de estudo" (ex: fazer resumos, mapas mentais).

- **Técnicas de ensino** são as estratégias do professor (ex: treinar com exercícios, corrigir erros).

Exemplo:

Para ensinar um modelo a reconhecer gatos em fotos:

1. **Dados:** Mostre milhares de fotos de gatos e não gatos.
2. **Modelo estatístico:** O algoritmo identifica padrões (ex: orelhas pontudas, bigodes).
3. **Técnica de ensino:** Ajusta os erros (ex: se confundir um cachorro com gato, o modelo "corrige" sua lógica).

---

2. Tipos de Aprendizado de Máquina (20% Essencial)

a) Supervisionado vs. Não Supervisionado

Tipo	Supervisionado	Não Supervisionado
Dados	Rotulados (ex: fotos com tags "gato" ou "não gato").	Sem rótulos (ex: fotos sem classificação).
Objetivo	Prever rótulos (ex: classificar e-mails em spam/não spam).	Descobrir padrões ocultos (ex: agrupar clientes por comportamento).
Analogia	Aprender com um professor corrigindo suas respostas.	Explorar um mapa desconhecido e criar seu próprio guia.

Semi-Supervisionado:

- **Exemplo:** 100 fotos de gatos (rotuladas) + 1.000 fotos sem rótulos.
- **Uso:** Quando rotular dados é caro ou demorado (ex: diagnósticos médicos).

b) Batch (Lote) vs. Online (Gradativo)

Tipo	Batch	Online
Aprendizado	De uma vez, com todos os dados.	Aos poucos, com dados novos em tempo real.
Vantagem	Precisão maior (analisa tudo).	Adaptação rápida (ex: detectar fraudes novas).
Exemplo	Treinar um modelo para reconhecer rostos com um banco de dados fixo.	Atualizar recomendações da Netflix conforme você assiste novos filmes.

c) Baseado em Instância vs. Modelo

Tipo	Baseado em Instância	Baseado em Modelo
Funcionamento	Memoriza exemplos e compara novos casos (ex: "decoreba").	Cria regras gerais a partir dos dados (ex: "entende a lógica").
Exemplo	Diagnóstico médico baseado em casos similares passados.	Prever preços de casas usando uma fórmula matemática.
Medida de Similaridade	Usa distância (ex: "quão parecido é com X?").	Não usa: baseia-se em padrões abstratos.

---

3. Resumo dos 20% que mais Importam

1. **Como aprendem?**
    - Máquinas aprendem como estudantes: com dados (livros), modelos (métodos) e ajustes (correções).
2. **Tipos de AM:**
    - **Supervisionado:** Rotulado (professor corrige).
    - **Não Supervisionado:** Sem rótulo (explora sozinho).
    - **Batch vs. Online:** Aprende tudo de uma vez vs. aprende continuamente.
    - **Instância vs. Modelo:** Decora exemplos vs. cria regras gerais.
3. **Quando usar cada um?**

- **Supervisionado:** Tarefas de classificação (ex: spam).
- **Não Supervisionado:** Descobrir grupos ou padrões (ex: segmentação de clientes).
- **Online:** Sistemas em tempo real (ex: detecção de fraudes).

Exemplo Prático:

Problema: Uma loja quer prever quais clientes vão comprar na Black Friday.

- **Supervisionado:** Usa dados históricos (ex: quem comprou no passado) para prever futuros compradores.
- **Não Supervisionado:** Agrupa clientes em perfis (ex: "economistas", "compradores impulsivos").
- **Online:** Atualiza as previsões diariamente com novos dados de navegação no site.

---

1. Por que Python?

Características Principais:

- **Alto nível:** Sintaxe simples, próxima da linguagem humana.
- **Interpretada:** Execução direta do código, sem compilação prévia.
- **Comunidade ativa:** Bibliotecas poderosas como **NumPy** (cálculos numéricos), **Pandas** (análise de dados) e **Matplotlib** (gráficos).

Exemplo de Uso:

python

#Instalação das bibliotecas (comando no terminal):

#pip install numpy pandas matplotlib

---

2. Tipos de Dados em Python

a) Números:

Tipo	Exemplo	Descrição
`int`	`5`, `-10`	Números inteiros.
`float`	`3.14`, `-0.5`	Números decimais.
`bool`	`True`, `False`	Valores lógicos (verdadeiro/falso).

b) Sequências:

Tipo	Mutável?	Exemplo	Descrição
`str`	Não	`"Python"`	Textos.
`list`	Sim	`[1, "a", True]`	Coleção ordenada e modificável.
`tuple`	Não	`(1, 2, 3)`	Coleção ordenada e imutável.
`dict`	Sim	`{"nome": "Pablo", "idade": 30}`	Pares chave-valor.

---

3. Trabalhando com Strings

a) Slicing (Fatiamento):

strMinhaString = 'Pablo é professor de ADS'

print(strMinhaString[0]) # Saída: 'P' (primeiro caractere)
print(strMinhaString[1:5]) # Saída: 'ablo' (índices 1 a 4)
print(strMinhaString[5:]) # Saída: ' é professor de ADS' (do índice 5 até o fim)
print(strMinhaString[:5]) # Saída: 'Pablo' (do início até o índice 4)
print(strMinhaString[-6]) # Saída: 'd' (6º caractere a partir do fim)
print(strMinhaString[::2]) # Saída: 'Pboépoesrd D' (pula de 2 em 2 caracteres)

#Exemplo com números:

numeros = '123456789'
print(numeros[1:7:2]) # Saída: '246' (índices 1, 3, 5)
print(numeros[::-1]) # Saída: '987654321' (inverte a string)

b) Concatenação e Multiplicação:

```
nome = 'Pablo'
nome += ' é professor'
print(nome) # Saída: 'Pablo é professor'

print('z' * 10) # Saída: 'zzzzzzzzzz'
print('2' + '3') # Saída: '23' (concatenação)
print(2 + 3) # Saída: 5 (soma numérica)
```

c) Métodos Úteis:

```
texto = 'Pablo é professor de ADS'

print(texto.upper()) # Saída: 'PABLO É PROFESSOR DE ADS'
print(texto.lower()) # Saída: 'pablo é professor de ads'

#Substituir parte da string:

novo_texto = texto.replace('Pablo', 'Carlos')
print(novo_texto) # Saída: 'Carlos é professor de ADS'
```

d) Imutabilidade das Strings:

```
texto = 'Casa'

#texto[0] = 'c' → ERRO! (strings são imutáveis)

#Solução: Criar uma nova string

novo_texto = 'c' + texto[1:]
print(novo_texto) # Saída: 'casa'
```

---

4. Aspas e Strings Multilinha

a) Aspas Simples vs. Duplas:

```
print("Hello 'World'!") # Saída: Hello 'World'!
print('Hello "World"!') # Saída: Hello "World"!

#Usando escape:

print('Hello 'World'!') # Saída: Hello 'World'!
```

b) Strings Multilinha:

```
texto = '''Este é um exemplo
de string em múltiplas
linhas.'''

print(texto)

#Saída: Este é um exemplo de string em múltiplas linhas.
```

---

5. Operações com Listas e Tuplas

a) Listas (Mutáveis):

```
lista = [1, "Python", True]
lista[0] = 10 # Alteração permitida
print(lista) # Saída: [10, 'Python', True]
```

b) Tuplas (Imutáveis):

```
tupla = (1, "Python", True) #tupla[0] = 10 → ERRO! (não pode ser alterada)
```

---

6. Principais Bibliotecas

a) NumPy (Arrays Numéricos):

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

# Saída: [2 4 6]
```

---

b) Pandas (Análise de Dados):

```
import pandas as pd
dados = {'Nome': ['Ana', 'João'], 'Idade': [25, 30]}
tabela = pd.DataFrame(dados)
print(tabela)

#Saída:

Nome Idade 0 Ana 25 1 João 30
```

c) Matplotlib (Gráficos):

```
import matplotlib.pyplot as plt
plt.plot([1, 2, 3], [4, 5, 1])
plt.xlabel('Eixo X')
plt.ylabel('Eixo Y')
plt.show()
```

---

7. Resumo dos 20% Essenciais

1. **Strings:**
   - Use `[start:stop:step]` para fatiar.
   - Métodos como `upper()`, `lower()`, e `replace()` são essenciais.
   - Strings são imutáveis: sempre crie novas strings para modificações.
2. **Listas vs. Tuplas:**
   - Listas são flexíveis (`lista = [1, 2, 3]`).
   - Tuplas são imutáveis (`tupla = (1, 2, 3)`).
3. **Bibliotecas:**
   - **Pandas** para dados tabulares.
   - **NumPy** para cálculos numéricos.
   - **Matplotlib** para visualização.

---

1. Trabalhando com Strings: Split e Fatiamento

a) Dividindo Strings em Listas (`split()`):

#Exemplo 1: Split por espaços

```
strMinhaString = 'Pablo é professor de ADS'
lista_palavras = strMinhaString.split()
print(lista_palavras)
```

#Saída: ['Pablo', 'é', 'professor', 'de', 'ADS']

#Exemplo 2: Split por vírgulas

```
pizzas = 'calabreza, portuguesa, marguerita, frango catupiri, bacon com frango, pepperoni'
lista_pizzas = pizzas.split(', ')
print(lista_pizzas)
```

#Saída: ['calabreza', 'portuguesa', 'marguerita', 'frango catupiri', 'bacon com frango', 'pepperoni']

Observação:

- `split()` sem argumentos divide por espaços.
- `split(', ')` divide por vírgula seguida de espaço.

---

2. Formatação de Strings (f-strings e %)

f-strings (Moderno e Recomendado):

```
nome = 'Pablo'
print(f'Meu nome é: {nome}.')
```

#Saída: Meu nome é: Pablo.

Caracteres Especiais e Parâmetros:

Parâmetro	Uso	Exemplo
`%s`	Substitui por string.	`print('Nome: %s' % 'Ana')`
`%r`	Representação "crua" (inclui aspas).	`print('Texto: %r' % 'Olá')` → `'Olá'`
`\t`	Tabulação (espaço de tab).	`print('Nome:\tAna')`
`%x.yf`	Formata números decimais.	`%5.2f` → 2 casas decimais, 5 caracteres no total.

---

3. Tuplas (Imutáveis) vs. Listas (Mutáveis)

a) Criando e Usando Tuplas:

#Tupla de sabores de pizza

tupla_pizzas = ('calabresa', 'marguerita', 'portuguesa')

#Tentar modificar → ERRO! #tupla_pizzas[0] = 'frango' # Não permitido!

#Métodos de Tuplas:

print(tupla_pizzas.count('calabresa')) // Saída: 1 (quantas vezes aparece)
print(tupla_pizzas.index('marguerita')) // Saída: 1 (índice do elemento)

b) Listas (Para Comparação):

lista_pizzas = ['calabresa', 'marguerita', 'portuguesa']
lista_pizzas[0] = 'frango' // Modificação permitida!
print(lista_pizzas)
// Saída: ['frango', 'marguerita', 'portuguesa']

Diferença Chave:

- **Tuplas** são imutáveis (usadas para dados fixos, como coordenadas).
- **Listas** são mutáveis (usadas para coleções dinâmicas).

---

4. Principais Takeaways

1. `split()`:
    - Transforma strings em listas.
    - Use `split(', ')` para separar por vírgulas e espaços.
2. **Formatação de Strings:**
    - Prefira **f-strings** (ex: `f'Nome: {nome}'`).
    - Use `%` apenas para compatibilidade com código antigo.
3. **Tuplas vs. Listas:**
    - **Tuplas:** Imutáveis, usadas para dados fixos.
    - **Listas:** Mutáveis, usadas para dados que mudam.
4. **Métodos de Tuplas:**

- `count()` : Conta ocorrências de um elemento.
- `index()` : Retorna o índice de um elemento.

Exemplo Integrado (Juntando Tudo):

Dividindo uma string em lista

clientes = 'Ana, João, Maria, Pedro'
lista_clientes = clientes.split(', ')

Formatando com f-strings

for cliente in lista_clientes:
print(f'Cliente: {cliente.upper()}')

Saída:

Cliente: ANA

Cliente: JOÃO

Cliente: MARIA

Cliente: PEDRO

Exercícios

Exercício 1: Primeiro Nome via Slice e Split

Objetivo: Extrair o primeiro nome de uma string usando duas técnicas diferentes.

Solução:

Passo A: Carregar o nome completo

**strMeuNome = "Pablo Coelho de Souza" **

Passo B1: Usando SLICE (fatiamento) Encontra o índice do primeiro espaço e fatia até lá

`espaco_index = strMeuNome.index(' ') primeiro_nome_slice = strMeuNome[:espaco_index] print("Via SLICE:", primeiro_nome_slice)`

Passo B2: Usando SPLIT Divide a string em uma lista e pega o primeiro elemento

`primeiro_nome_split = strMeuNome.split()[0] print("Via SPLIT:", primeiro_nome_split)`

Explicação:

- **Slice:** Usamos `index(' ')` para encontrar onde o primeiro espaço está e fatiar até esse ponto.
- **Split:** Dividimos a string em uma lista de palavras (`['Pablo', 'Coelho', 'de', 'Souza']`) e selecionamos o primeiro item (`[0]`).

Saída:

Via SLICE: Pablo
Via SPLIT: Pablo

Exercício 2: Testando Formatação de Strings

Objetivo: Entender os comportamentos de `%s` , `%r` , `%d` , e `%f` .

Solução:

python

Copy

print('O nome é %s.' % 'Pablo') # Saída: O nome é Pablo.
print('O nome é %r.' % 'Pablo') # Saída: O nome é 'Pablo'. (com aspas)
print('O nome é %s.' % 'Pablo \t Coelho') # Saída: O nome é Pablo Coelho. (com tabulação)
print('O nome é %r.' % 'Pablo \t Coelho') # Saída: O nome é 'Pablo \t Coelho'. (mostra o \t)

print('O Pablo tem R 3.75 na carteira.
print('O Pablo tem R 3 na carteira. (%d trunca para inteiro)

print('Ponto flutuante: %5.2f' % 13.144) # Saída: Ponto flutuante: 13.14 (5 caracteres, 2 decimais)
print('Ponto flutuante: %1.0f' % 13.144) # Saída: Ponto flutuante: 13 (1 caractere, 0 decimais)
print('Ponto flutuante: %1.5f' % 13.144) # Saída: Ponto flutuante: 13.14400 (5 decimais)
print('Ponto flutuante: %10.2f' % 13.144) # Saída: Ponto flutuante: 13.14 (10 caracteres)
print('Ponto flutuante: %25.2f' % 13.144) # Saída: Ponto flutuante: 13.14 (25 caracteres)

Explicação:

- `%s` : Exibe o valor como string, ignorando caracteres especiais (como `\t`).

- `%r`: Mostra a representação "crua" (raw), incluindo aspas e caracteres especiais.
- `%d`: Converte o valor para inteiro (trunca decimais).
- `%x.yf`:
  - `x`: Número total de caracteres (incluindo o ponto).
  - `y`: Número de casas decimais.
  - Exemplo: `%10.2f` alinha o número em 10 caracteres com 2 casas decimais.

Exercício 3: Tuple com Letras Repetidas

Objetivo: Criar uma tuple com as primeiras letras do alfabeto, repetindo 'a' três vezes.

Objetivo: Entender os comportamentos de `%s`, `%r`, `%d`, e `%f`.

Solução:

python

Copy

print('O nome é %s.' % 'Pablo') # Saída: O nome é Pablo.
print('O nome é %r.' % 'Pablo') # Saída: O nome é 'Pablo'. (com aspas)
print('O nome é %s.' % 'Pablo \t Coelho') # Saída: O nome é Pablo Coelho. (com tabulação)
print('O nome é %r.' % 'Pablo \t Coelho') # Saída: O nome é 'Pablo \t Coelho'. (mostra o \t)

print('O Pablo tem R 3.75 na carteira.
print('O Pablo tem R 3 na carteira. (%d trunca para inteiro)

print('Ponto flutuante: %5.2f' % 13.144) # Saída: Ponto flutuante: 13.14 (5 caracteres, 2 decimais)
print('Ponto flutuante: %1.0f' % 13.144) # Saída: Ponto flutuante: 13 (1 caractere, 0 decimais)
print('Ponto flutuante: %1.5f' % 13.144) # Saída: Ponto flutuante: 13.14400 (5 decimais)
print('Ponto flutuante: %10.2f' % 13.144) # Saída: Ponto flutuante: 13.14 (10 caracteres)
print('Ponto flutuante: %25.2f' % 13.144) # Saída: Ponto flutuante: 13.14 (25 caracteres)

Explicação:

- `%s`: Exibe o valor como string, ignorando caracteres especiais (como `\t`).
- `%r`: Mostra a representação "crua" (raw), incluindo aspas e caracteres especiais.

- `%d`: Converte o valor para inteiro (trunca decimais).
- `%x.yf`:
  - `x`: Número total de caracteres (incluindo o ponto).
  - `y`: Número de casas decimais.
  - Exemplo: `%10.2f` alinha o número em 10 caracteres com 2 casas decimais.

Exercício 3: Tuple com Letras Repetidas

Objetivo: Criar uma tuple com as primeiras letras do alfabeto, repetindo 'a' três vezes.

Solução:

Objetivo: Entender os comportamentos de `%s`, `%r`, `%d`, e `%f`.

Solução:

print('O nome é %s.' % 'Pablo') # Saída: O nome é Pablo.
print('O nome é %r.' % 'Pablo') # Saída: O nome é 'Pablo'. (com aspas)
print('O nome é %s.' % 'Pablo \t Coelho')

Saída: O nome é Pablo Coelho. (com tabulação)

print('O nome é %r.' % 'Pablo \t Coelho')

Saída: O nome é 'Pablo \t Coelho'. (mostra o \t)

print('O Pablo tem RS %s na carteira.' % 3.75) #Saída: O Pablo tem RS 3.75 na carteira.

print('O Pablo tem RS %d na carteira.' % 3.75) #Saída: O Pablo tem R$ 3 na carteira. (%d trunca para inteiro)

print('Ponto flutuante: %5.2f' % 13.144) #Saída: Ponto flutuante: 13.14 (5 caracteres, 2 decimais)

print('Ponto flutuante: %1.0f' % 13.144)
#Saída: Ponto flutuante: 13 (1 caractere, 0 decimais)

print('Ponto flutuante: %1.5f' % 13.144)
#Saída: Ponto flutuante: 13.14400 (5 decimais)

print('Ponto flutuante: %10.2f' % 13.144) #Saída: Ponto flutuante: 13.14 (10 caracteres)

print('Ponto flutuante: %25.2f' % 13.144) #Saída: Ponto flutuante: 13.14 (25 caracteres)

Explicação:

- `%s` : Exibe o valor como string, ignorando caracteres especiais (como `\t`).
- `%r` : Mostra a representação "crua" (raw), incluindo aspas e caracteres especiais.
- `%d` : Converte o valor para inteiro (trunca decimais).
- `%x.yf` :
  - `x` : Número total de caracteres (incluindo o ponto).
  - `y` : Número de casas decimais.
  - Exemplo: `%10.2f` alinha o número em 10 caracteres com 2 casas decimais.

Exercício 3: Tuple com Letras Repetidas

Objetivo: Criar uma tuple com as primeiras letras do alfabeto, repetindo 'a' três vezes.

Solução:

Criando a tuple alfabeto = ('a', 'a', 'a', 'b', 'c', 'd', 'e')

Imprimindo resultados

print("Tamanho da Tuple:", len(alfabeto)) # Saída: 7
print("Quantidade de 'a's:", alfabeto.count('a')) # Saída: 3
print("Posição de 'c':", alfabeto.index('c')) # Saída: 4

Explicação:

- A tuple inclui 3 letras 'a' seguidas por 'b', 'c', 'd', 'e'.
- `len()` retorna o número total de elementos.

- `count('a')` conta quantas vezes 'a' aparece.
- `index('c')` retorna a posição da primeira ocorrência de 'c' (índice 4).

Exercício 4: Análise de Tuple Existente

Objetivo: Entender métodos básicos de tuples.

Solução:

myTuple = ('a', 'a', 'a', 'b', 'c', 'd', 'e')

print(len(myTuple)) # Saída: 7 (tamanho da tuple)
print(myTuple.count('a')) # Saída: 3 (quantidade de 'a's)
print(myTuple.index('c')) # Saída: 4 (posição de 'c')

Explicação:

- **`len()`**: Conta todos os elementos da tuple.
- **`count('a')`**: Conta ocorrências do elemento 'a'.
- **`index('c')`**: Retorna o índice da primeira ocorrência de 'c'.

Resumo Final

1. **Slice vs. Split:**
   - Use `slice` quando souber a posição exata do corte.
   - Use `split` para dividir a string em partes baseadas em um separador.
2. **Formatação com `%`:**
   - `%s` para strings, `%r` para representação crua, `%d` para inteiros, `%f` para decimais.
3. **Tuples:**
   - Imutáveis e eficientes para dados fixos.
   - Métodos úteis: `count()`, `index()`, `len()`.

---

1. Sets (Conjuntos)

O que são?

Coleções **não ordenadas** de elementos **únicos**.

- **Uso principal:** Remover duplicatas e verificar pertencimento de forma eficiente.
- **Características:**
    - **Mutáveis:** Você pode adicionar/remover elementos.
    - **Não indexados:** Não é possível acessar elementos por índice.

Exemplos Práticos:

Criando um set frutas = {"maçã", "banana", "laranja"}
frutas.add("uva") # Adiciona "uva"
print(frutas) Saída: {'maçã', 'banana', 'laranja', 'uva'} (ordem aleatória)

Removendo duplicatas de uma lista lista_repetida = [1, 2, 2, 3, 3, 3]
sem_duplicatas = set(lista_repetida)
print(sem_duplicatas) Saída: {1, 2, 3}

Características:

- Coleções **não ordenadas** de elementos **únicos**.
- Úteis para remover duplicatas e verificar pertencimento.

Exemplo 1: Criando um Set

setMeuConjunto = set()
setMeuConjunto.add('a')
setMeuConjunto.add('d')
setMeuConjunto.add('b')
setMeuConjunto.add(2)
print(setMeuConjunto) # Saída: {'d', 'b', 2, 'a'} (ordem aleatória)

Exemplo 2: Removendo Duplicatas

lista_repetida = [1,1,1,2,2,21,3,4,51,23,1,2,3,5,8]
set_sem_duplicatas = set(lista_repetida)
print(set_sem_duplicatas) # Saída: {1, 2, 3, 4, 5, 8, 51, 21, 23}

2. Trabalhando com Arquivos

Etapas Básicas:

1. **Abrir o arquivo:** Especificar o modo de abertura (`r` para leitura, `w` para escrita, `a` para append).
2. **Manipular o conteúdo:** Ler ou escrever dados.
3. **Fechar o arquivo:** Liberar recursos do sistema.

O que é o modo `'a'` (append)?

O modo `'a'` (abreviação de **append**, que significa "acrescentar" em inglês) é usado para **adicionar conteúdo ao final de um arquivo existente**, sem apagar o que já está escrito.

Características principais:

1. **Não sobrescreve o arquivo:**
   - Se o arquivo já existir, o conteúdo novo é adicionado **após o último caractere** do arquivo.
   - Se o arquivo não existir, ele será **criado automaticamente**.
2. **Uso típico:**
   - Adicionar logs, atualizar registros ou escrever dados novos sem perder o histórico.

Comparação com outros modos de abertura:

Modo	Nome	Funcionalidade
`'r'`	Read (leitura)	Abre o arquivo apenas para leitura.
`'w'`	Write (escrita)	Abre o arquivo para escrita, **apagando o conteúdo anterior**.
`'a'`	Append (acrescentar)	Adiciona conteúdo ao final do arquivo (**sem apagar o que já existe**).

Exemplo Prático:

Escrevendo em um arquivo com `'a':

Abre o arquivo em modo append (ou cria se não existir)

with open("dados.txt", "a", encoding="utf-8") as arquivo:
arquivo.write("Nova linha de texto.\n") # \n para pular linha

Resultado:

- Se o arquivo `dados.txt` já tivesse: Linha 1.
  Linha 2.
- Após executar o código, ficaria: Linha 1.
  Linha 2.
  Nova linha de texto.

Observações Importantes:

1. **Quebras de linha (`\n`):**
   - Se você não incluir `\n`, o conteúdo será adicionado na **mesma linha** do último caractere.
2. **Modos combinados:**
   - Existe também o modo `'a+'`, que permite **ler e escrever** no arquivo (útil para atualizações mais complexas).
3. **Segurança:**
   - Use `with open(...) as` para garantir que o arquivo seja **fechado automaticamente**, mesmo se ocorrer um erro.

Quando usar `'a'` ?

- Quando você precisa **preservar o conteúdo original** do arquivo e apenas adicionar novos dados.
- Exemplos: registro de atividades, histórico de transações, logs de sistemas.

Exemplo Seguro (usando `with`):

Escrevendo em um arquivo with open("dados.txt", "w") as arquivo:
arquivo.write("Olá, mundo!\n")

Lendo um arquivo

```
with open("dados.txt", "r") as arquivo:
conteudo = arquivo.read()
print(conteudo) # Saída: "Olá, mundo!"
```

a) Criar e Escrever:

#Passo 1: Criar arquivo (modo 'x')

```
f = open("meuArquivo.txt", "x")
f.close()
```

#Passo 2: Escrever (modo 'a' para append)

```
f = open("meuArquivo.txt", "a")
f.write("Minha primeira linha.")
f.write("Minha segunda linha.\nTerceira linha") # \n pula linha
f.close()
```

b) Ler Arquivo:

```
f = open("meuArquivo.txt", "r")
conteudo = f.read()
print(conteudo)
f.close()
```

Saída:

Minha primeira linha.Minha segunda linha.

Terceira linha

Observação: A primeira linha não pulou porque faltou `\n` após a primeira escrita.

---

3. Operadores de Comparação

Tabela de Operadores:

Operador	Descrição	Exemplo
`==`	Igualdade	`3 == 3` → `True`
`!=`	Diferente	`3 != 4` → `True`
`>`	Maior que	`3 > 4` → `False`
`<`	Menor que	`3 < 4` → `True`
`>=`	Maior ou igual	`3 >= 3` → `True`
`<=`	Menor ou igual	`3 <= 4` → `True`

Exemplo com Cadeias (AND/OR):

a, b, c = 1, 2, 3
print(a == b or a == a) # True (OR: uma condição verdadeira)
print(a == b and a == a) # False (AND: ambas precisam ser verdadeiras)

---

4. Estruturas de Decisão (if/elif/else)

Funcionamento:

- **`if`**: Verifica uma condição. Se verdadeira, executa o bloco associado.
- **`elif`**: Verifica condições adicionais se a anterior for falsa.
- **`else`**: Executa um bloco se todas as condições anteriores forem falsas.

Exemplo Básico:

idade = 18

if idade < 12:
print("Criança")
elif 12 <= idade < 18:
print("Adolescente")
else:
print("Adulto")

---

5. Estruturas de Repetição (FOR)

Uso: Iterar sobre elementos de uma sequência (listas, strings, etc.).

Exemplos:

Loop em uma lista numeros = [1, 2, 3]
for num in numeros:
print(num * 2) # Saída: 2, 4, 6

Loop com range

for i in range(0, 10, 2):
print(i) # Saída: 0, 2, 4, 6, 8

Loop com enumerate (índice e valor)

for indice, valor in enumerate(["a", "b", "c"]):
print(f"Índice: {indice}, Valor: {valor}")

a) Loop com Range:

for i in range(0, 10, 2): # Início: 0, Fim: 10, Passo: 2
print(i)

Saída: 0, 2, 4, 6, 8

b) Loop em Listas:

lista = [1, 2, 3, 4, 5]
for numero in lista:
print(numero) # Imprime cada número da lista

c) Verificar Par/Ímpar:

for num in range(1, 11):
if num % 2 == 0:
print(f"{num} é par")
else:
print(f"{num} é ímpar")

---

6. Função ZIP

Propósito:

Combinar elementos de múltiplas sequências em pares/tuplas.

Exemplo:

Combinando Listas:

nomes = ['Ana', 'João', 'Maria']
idades = [25, 30, 28]

for nome, idade in zip(nomes, idades):
print(f"{nome} tem {idade} anos")

Saída: Ana tem 25 anos João tem 30 anos Maria tem 28 anos

---

7. Operações com Listas

Operações Comuns:

- **Adicionar elementos:** `append()`, `insert()`, `extend()`.
- **Remover elementos:** `remove()`, `pop()`, `clear()`.
- **Ordenação:** `sort()`, `sorted()`.
- **Busca:** `index()`, `count()`.

Exemplos:

Criando uma lista lista = [1, 2, 3]
lista.append(4) # [1, 2, 3, 4]
lista.sort(reverse=True) # [4, 3, 2, 1]

List comprehension (criação concisa) quadrados = [x**2 for x in range(5)] # [0, 1, 4, 9, 16]

a) Verificar Existência (IN):

lista = [1, 2, 3]
print(1 in lista) # Saída: True

b) Maior e Menor Elemento:

lista = [5, 3, 8, 1]
print(max(lista)) # Saída: 8

print(min(lista)) # Saída: 1

---

8. Gerando Números Aleatórios

Módulo `random`:

- **`randint(a, b)`**: Gera um inteiro aleatório entre `a` e `b`.
- **`random()`**: Gera um float aleatório entre 0 e 1.
- **`choice(sequencia)`**: Escolhe um elemento aleatório de uma sequência.

Exemplos:

import random

//Inteiro entre 1 e 10 numero = random.randint(1, 10)

Escolha aleatória lista = ["cara", "coroa"]
escolha = random.choice(lista)

Embaralhar uma lista cartas = ["Ás", "Rei", "Dama"]
random.shuffle(cartas)

Exemplo com Random:

import random

#10 números aleatórios entre 1 e 100 numeros = [random.randint(1, 100) for _ in range(10)]

#10 números únicos entre 1 e 100 numeros_unicos = random.sample(range(1, 101), 10)

Resumo dos 20% Essenciais

1. **Sets:**
   - Use para remover duplicatas: `set(lista)`.
   - Elementos únicos e não ordenados.
2. **Arquivos:**
   - Modos: `x` (criar), `a` (append), `r` (ler).

- Sempre feche o arquivo ou use `with open(...) as f`.

3. **Operadores Lógicos:**
   - `and` exige **todas** condições verdadeiras.
   - `or` exige **pelo menos uma** condição verdadeira.
4. **Loops (FOR):**
   - Use `range()` para sequências numéricas.
   - Use `zip()` para combinar listas.
5. **Random:**
   - `randint` para números aleatórios.
   - `sample` para listas únicas.

Pandas e **Matplotlib**

Para que servem?

- **Pandas:** É a ferramenta para pegar dados "bagunçados" (planilhas, arquivos de texto, etc.) e transformá-los em tabelas organizadas e fáceis de manipular no Python. Pense no Pandas como um "Excel superpoderoso" dentro do seu código. Essencial para **limpar, organizar e explorar** os dados.
- **Matplotlib:** É a ferramenta para **desenhar gráficos** a partir dos seus dados. Depois que o Pandas organiza a tabela, o Matplotlib ajuda a visualizar as informações, encontrar padrões, tendências, etc. Uma imagem vale mais que mil linhas de tabela!

A Estrutura Principal do Pandas: O DataFrame

- A ideia central do Pandas é o **DataFrame**. Imagine uma planilha (como Excel ou Google Sheets), com linhas e colunas, onde cada coluna tem um nome (um cabeçalho). É isso que um DataFrame representa no Python.

  Exemplo: Quando você tem dados como:

```
cars  passings
0   BMW         3
1 Volvo         7
2  Ford         2
```

Isso é um DataFrame. A coluna cars tem os nomes dos carros, a coluna passings tem números. A coluna da esquerda (0, 1, 2) é o **índice**, como o número da linha na planilha.

Como Colocar Dados em um DataFrame? (O mais comum!)

- Frequentemente, seus dados estarão em arquivos, como .csv (valores separados por vírgula). O Pandas torna super fácil ler esses arquivos.
- Exemplo:

```
import pandas as pd
# Lê o arquivo 'data.csv' e guarda na variável 'df'
df = pd.read_csv('data.csv')
# Mostra o conteúdo do DataFrame (geralmente as primeiras
e últimas linhas)
print(df)
```

Esse comando pd.read_csv() é um dos mais usados. Ele pega o arquivo data.csv (que tem nome, dia de nascimento, etc.) e o transforma naquela estrutura de tabela organizada (DataFrame) dentro da variável df.

1. **Como Espiar os Dados no DataFrame?**
    - Quando você tem muitos dados, não quer imprimir tudo.
    - Exemplo: Para ver só o **começo** da tabela:

```
print(df.head()) # Mostra as 5 primeiras linhas por padrão
print(df.head(10)) # Mostra as 10 primeiras linhas
```

Exemplo: Para ver só o **final** da tabela:

```
print(df.tail()) # Mostra as 5 últimas linhas por padrão
```

Exemplo: Para ter um **resumo** da estrutura (tipos de dados, quantas linhas não vazias):

```
print(df.info())
```

Isso é útil para ver rapidamente se há dados faltando (como no exemplo das Calorias onde 164 non-null indica que faltam 5 valores em 169 linhas).

Por que Limpar Dados é Crucial?

- Machine Learning aprende com exemplos. Se os exemplos (dados) estiverem errados, incompletos ou bagunçados, o aprendizado será ruim ("lixo entra, lixo sai").
- Problemas comuns: células vazias, formatos errados (texto onde deveria ser número), valores impossíveis (idade negativa?), linhas duplicadas.

Como Lidar com Dados Faltando (O básico)?

- **Opção 1: Remover linhas com dados vazios.** Se você tem muitos dados e poucas linhas com falhas, pode simplesmente removê-las.
    - Exemplo:

```
# Cria NOVO DataFrame sem as linhas que tinham algum valor vazio
new_df = df.dropna()
# OU: Remove as linhas do DataFrame ORIGINAL (cuidado!)
df.dropna(inplace=True)
```

Opção 2: Preencher os vazios. Às vezes, remover linhas não é bom (perde-se informação). Você pode preencher os buracos com um valor.

- Exemplo: Preencher todos os vazios com 130:

```
df.fillna(130, inplace=True)
```

Exemplo: Preencher os vazios apenas na coluna 'Calorias' com 130:

```
df["Calorias"].fillna(130, inplace=True)
```

Exemplo Inteligente: Preencher com a **média**, **mediana** ou **moda** da coluna. É uma forma mais robusta de não distorcer tanto os dados.

MÉDIA: `mean` **MEDIANA:** `median` **MODA:** `mode`

```
# Calcula a média da coluna 'Calorias'
media_calorias = df["Calorias"].mean()
# Preenche os vazios na coluna 'Calorias' com a média calculada
df["Calorias"].fillna(media_calorias, inplace=True)
# Poderia usar .median() ou .mode()[0] também
```

(Lembre-se: Média é a soma dividida pela quantidade; Mediana é o valor do meio com dados ordenados; Moda é o valor que mais aparece).

Como Visualizar Dados com Matplotlib (O básico)?

- Depois de ter o DataFrame df limpo e organizado pelo Pandas, você pode usar o Matplotlib para desenhar.
- Exemplo: Um gráfico simples com todas as colunas numéricas:

```
import matplotlib.pyplot as plt
# Assume que 'df' é seu DataFrame Pandas já carregado e talvez limpo
df.plot() # Cria um gráfico (geralmente de linhas)
plt.show() # Mostra o gráfico na tela
```

O plt.show() é essencial para exibir o gráfico gerado.

Exemplo: Gráfico de Pizza (para mostrar proporções):
Imagine que você tem uma coluna survived no arquivo dados4.csv com 0 (não sobreviveu) e 1 (sobreviveu). Você quer ver a proporção.

```
# (Código anterior para contar numA e numB...)
dados = [numA, numB] # Lista com a contagem de 0s e 1s
plt.pie(dados, labels=['Não Sobreviveu', 'Sobreviveu'])
# Cria o gráfico de pizza
plt.title('Proporção de Sobreviventes') # Adiciona um título
plt.show() # Mostra o gráfico
```

Exemplo: Gráfico de Dispersão (para ver relação entre duas variáveis):
Você quer ver se existe uma relação entre a Duração do exercício e as Calorias queimadas.

```
# Supondo que 'x' tem os valores da coluna 'Duracao' e 'y' os da coluna 'Calorias'
plt.scatter(df['Duracao'], df['Calorias']) # Cria o gráfico de dispersão
plt.title('Duração do Exercício vs Calorias Queimadas') # Título
plt.xlabel('Duração (minutos)') # Rótulo do eixo X
plt.ylabel('Calorias') # Rótulo do eixo Y
plt.show() # Mostra o gráfico
```

Outros Pontos Importantes (Complemento):

- **Series:** Uma única coluna de um DataFrame é chamada de Series. Você pode criar uma Series a partir de uma lista ou dicionário, e dar nomes (índices) personalizados às linhas (index = ["x", "y", "z"]). É o "tijolo" fundamental do DataFrame.
- **Selecionando Dados (loc):** O loc permite pegar linhas (e colunas) pelo rótulo (índice ou nome da coluna). print(myvar.loc[0]) pega a linha com índice 0. print(myvar.loc[[0, 1]]) pega as linhas com índice 0 e 1.
- **to_string():** Se você quiser realmente ver todos os dados de um DataFrame grande, use df.to_string(). Sem ele, o Pandas mostra só o começo e o fim para não poluir a tela.
- **pd.options.display.max_rows:** Controla quantas linhas o Pandas mostra por padrão. Você pode aumentar se precisar.
- **Ler JSON:** Pandas também lê arquivos .json (pd.read_json()), que é outro formato comum para guardar dados estruturados.
- **Correção de Formato:** pd.to_datetime() é útil para garantir que colunas de data estejam no formato correto.
- **Ajustando Dados Errados:** df.loc[linha, coluna] = novo_valor permite corrigir um valor específico. Você pode usar isso em um loop (for) para corrigir vários valores (ex: valores de Duracao acima de 120). Ou df.drop(linha) para remover uma linha inteira com dado errado.
- **Removendo Duplicatas:** df.duplicated() mostra quais linhas são cópias exatas de outras. df.drop_duplicates(inplace=True) remove essas cópias.

Resumo da Ópera (Foco Pareto):

Use **Pandas** para carregar seus dados (principalmente de arquivos CSV com pd.read_csv) para dentro de um **DataFrame** (a tabela). Use head(), tail(), e info() para dar uma espiada. **Limpe os dados**, especialmente tratando valores vazios com dropna() ou fillna() (usando a média/mediana é uma boa prática). Depois, use **Matplotlib** (muitas vezes direto do Pandas com df.plot(), ou com

funções como plt.scatter() e plt.pie()) para **visualizar** esses dados e entender o que eles dizem. Não se esqueça do plt.show() para ver seus gráficos!

Dominando esses pontos essenciais, você já tem uma base muito sólida para começar a trabalhar com dados para Aprendizagem de Máquina! O resto são detalhes e ferramentas mais específicas que você aprenderá conforme a necessidade.

Agora vamos mergulhar no **Scikit-learn** (pronuncia-se "sai-kit learn"), que é como a "caixa de ferramentas principal" do Aprendizado de Máquina em Python. Se o Pandas é para preparar os dados, o Scikit-learn é para construir e usar os "cérebros" que aprendem com esses dados.

1. **O que é Scikit-learn?**
    - É uma biblioteca (um conjunto de códigos prontos) para Python focada especificamente em **Aprendizagem de Máquina**.
    - É como ter um kit com várias ferramentas e modelos prontos para você usar, sem precisar inventar tudo do zero.
    - É Open Source (código aberto e gratuito) e muito popular na comunidade.
    - Oferece ferramentas não só para os modelos em si, mas também para tarefas essenciais como:
        - **Pré-processamento:** Preparar os dados (vamos ver mais sobre isso).
        - **Seleção de Modelos:** Ajudar a escolher o melhor "cérebro" para seu problema.
        - **Avaliação:** Medir quão bem seu modelo está aprendendo e prevendo.
2. **A Ideia Central: fit e predict**
    - A maioria dos modelos no Scikit-learn funciona com um fluxo básico de duas etapas:
        - **fit(X, y):** É o processo de **treinamento** ou **aprendizado**. Você mostra ao modelo (estimator) os dados de entrada (as **características**, chamadas de X) e as respostas corretas correspondentes (os **resultados** ou **rótulos**, chamados de y). O modelo "estuda" essa relação.
        - **predict(X_novo):** Depois que o modelo foi treinado (depois do fit), você usa o método predict para fazer **previsões** sobre novos dados (X_novo) para os quais você não sabe a resposta. O modelo usa o que aprendeu para dar um palpite.

- **Analogia:** Pense em ensinar uma criança a reconhecer frutas (fit). Você mostra várias fotos (X) e diz o nome de cada fruta (y): "Isto é uma maçã", "Isto é uma banana". Depois de aprender, você mostra uma foto nova que ela nunca viu (X_novo) e pergunta: "O que é isto?". A resposta dela é a predict.
- **Exemplo Conceitual (Salários):**
  - X (Características): Uma tabela onde cada linha é um ex-aluno e as colunas são ['Instituição', 'Curso', 'Ano de Conclusão', 'Nota Média'].

```
X = [
     ['IESB', 'ADS', '2021', '9'],
     ['IESB', 'ADS', '2021', '7'],
     ['outra', 'ADS', '2021', '9'],
     ['IESB', 'ADS', '2023', '9']
    ]
```

```
- y (Resultados): Uma lista com o salário inicial correspondente a cada ex-aluno em X.

  ```
  y = ['3.500', '2.000', '2.500', '4.000']
  ```


- **Treinamento:** modelo.fit(X, y) - O modelo aprende a relação entre as característica

- **Predição:** Você pergunta: "Qual o salário para X_novo = ['IESB', 'ADS', '2024', '9'
```

1.
   -
     - salario_previsto = modelo.predict(X_novo)
     - O modelo responderia algo como ['4.200'] (valor hipotético baseado no aprendizado).

A Linguagem da Máquina: Números!

- Modelos de Machine Learning são fundamentalmente **matemáticos/estatísticos**. Eles não entendem "IESB" ou "verde". Eles entendem **números**.
- Portanto, qualquer dado que não seja número (texto, categorias) precisa ser **convertido em números** antes de ser usado no fit ou predict.

- Isso vale para imagens (pixels são números), sons (amostras são números) e, crucialmente, para **texto** e **dados categóricos**.

Transformando Texto/Categorias em Números: A Base

- **Dados Categóricos:** São dados que representam categorias ou qualidades, como "cor" (verde, vermelho, azul), "instituição" (IESB, outra), "forma" (quadrado, triângulo).
- **Tokenização (Ideia Geral):** É o processo de quebrar texto em pedaços menores (tokens) e atribuir um número a cada pedaço único, criando um "mapa" ou "dicionário" (léxico/vocabulário).
- **Método 1: Label Encoding (Codificação de Rótulo)**
    - **Conceito:** Simplesmente atribui um número inteiro sequencial para cada categoria única.
    - **Exemplo Tabela:**

ID	Cor	Forma		ID	Cor	Forma
1	Verde	Quadrado	→	1	1	1
2	Vermelho	Triangulo	→	2	2	2
3	Vermelho	Quadrado	→	3	2	1
4	Azul	Triangulo	→	4	3	2
5	Verde	Circulo	→	5	1	3

(Aqui, Verde=1, Vermelho=2, Azul=3; Quadrado=1, Triangulo=2, Circulo=3)

- **Exemplo Código (Pandas):** Cria uma coluna numérica representando a categoria.

  ```
  import pandas as pd

  setCores = ('verde','vermelho','azul')
  dfCores = pd.DataFrame(setCores, columns=['Cores'])

  # Converte a coluna para o tipo 'category' do Pandas
  dfCores['Cores'] = dfCores['Cores'].astype('category')

  # Cria uma nova coluna ('Cores_cat') com os códigos numéricos
  dfCores['Cores_cat'] = dfCores['Cores'].cat.codes
  print(dfCores)
  ```

  Saída:

  ```
  Cores  Cores_cat
  0     verde          1
  1  vermelho          2
  2      azul          0 # Pandas pode começar do 0
  ```

- **Exemplo Código (Scikit-learn):** Faz a mesma coisa usando LabelEncoder.

  ```
  import pandas as pd
  from sklearn.preprocessing import LabelEncoder

  setCores = ('verde','vermelho','azul','azul','azul',
  'rosa','vermelho')
  dfCores = pd.DataFrame(setCores, columns=['Cores'])

  meuLabelEncoder = LabelEncoder() # Cria o codificador
  # Aplica o codificador à coluna 'Cores' e cria 'Cores_cat'
  dfCores['Cores_cat'] = meuLabelEncoder.fit_transform(dfCores
  ['Cores'])
  print(dfCores)
  ```

1. ○ ■ Saída:

```
Cores  Cores_cat
0     verde          2 #Núm. dependem ordem alfabética/aparição
1  vermelho          3
2      azul          0
3      azul          0
4      azul          0
5      rosa          1
6  vermelho          3
```

```
- **PROBLEMA do Label Encoding:** O modelo pode interpretar uma **ordem ou magnitude** q
```

- **Método 2: One-Hot Encoding (Codificação "Quente Única")**
  - **Conceito:** Resolve o problema do Label Encoding. Em vez de uma única coluna com números diferentes, ele cria **novas colunas**, uma para cada categoria possível. O valor será 1 na coluna correspondente à categoria daquela linha e 0 nas outras.
  - **Exemplo Tabela:**

ID	Cor	Forma		ID	...	Azul	Verde	Verm.	Círc.
1	Verde	Quadrado	→	1	...	0	1	0	0
2	Vermelho	Triangulo	→	2	...	0	0	1	0
3	Vermelho	Quadrado	→	3	...	0	0	1	0
4	Azul	Triangulo	→	4	...	1	0	0	0
5	Verde	Circulo	→	5	...	0	1	0	1

  - **Exemplo Código (Scikit-learn):** Geralmente feito após o Label Encoding ou direto na coluna categórica.

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder, OneHotEncoder

setCores = ('verde','vermelho','azul','azul','azul',
'rosa','vermelho')
dfCores = pd.DataFrame(setCores, columns=['Cores'])

# Passo 1: Label Encoding (como antes)
meuLabelEncoder = LabelEncoder()
dfCores['Cores_cat'] = meuLabelEncoder.fit_transform(dfCores
['Cores'])

# Passo 2: One-Hot Encoding
enc = OneHotEncoder(handle_unknown='ignore', sparse_output=
False) # Cria o OneHotEncoder
# Aplica na coluna _numérica_ 'Cores_cat' e cria um DataFrame
com as novas colunas
# O .toarray() é importante se sparse=True (padrão antigo),
ou use sparse_output=False
dfEnc = pd.DataFrame(enc.fit_transform(dfCores[['Cores_cat']]))

# Junta as novas colunas (0, 1, 2, 3) ao DataFrame original
dfCores = dfCores.join(dfEnc)
print(dfCores)
```

Saída:

```
Cores  Cores_cat    0    1    2    3
0     verde          2  0.0  0.0  1.0  0.0  #Col2 (verde) é 1
1  vermelho          3  0.0  0.0  0.0  1.0  #Col3(vermelho) é 1
2      azul          0  1.0  0.0  0.0  0.0  #Col0 (azul) é 1
3      azul          0  1.0  0.0  0.0  0.0
4      azul          0  1.0  0.0  0.0  0.0
5      rosa          1  0.0  1.0  0.0  0.0  #Col1(rosa) é 1
6  vermelho          3  0.0  0.0  0.0  1.0
```

- **Passo Final Importante:** Depois de criar as colunas One-Hot, **remova** a coluna or

```
# Remove a coluna 'Cores_cat' (índice 1 se for a segunda
coluna)
dfCores.drop(['Cores_cat'], axis=1, inplace=True)
# Poderia remover a 'Cores' também: dfCores.drop(['Cores'],
axis=1, inplace=True)
# Ou remover por posição (cuidado se a ordem mudar):
# dfCores.drop(dfCores.iloc[:, 1:2], axis=1, inplace=True)
# Remove a segunda coluna
print(dfCores)
```

Saída Final (pronta para o modelo):

```
Cores    0    1    2    3
0     verde  0.0  0.0  1.0  0.0
1  vermelho  0.0  0.0  0.0  1.0
2      azul  1.0  0.0  0.0  0.0
3      azul  1.0  0.0  0.0  0.0
4      azul  1.0  0.0  0.0  0.0
5      rosa  0.0  1.0  0.0  0.0
6  vermelho  0.0  0.0  0.0  1.0
# (Geralmente a coluna 'Cores' original também seria removida)
```

Outros Pontos Mencionados (Complemento):

- **Estimators:** É o nome genérico que o Scikit-learn dá aos seus **modelos** (os "cérebros"). Eles podem ser:
    - Supervisionados: Aprendem com X e y (como o exemplo do salário e o RandomForestClassifier).
    - Não Supervisionados: Aprendem padrões apenas com X (sem respostas y).
    - Transformadores: Não são modelos de previsão, mas ferramentas para modificar os dados (como LabelEncoder e OneHotEncoder).
- **Tipos de Dados Aceitos:** Scikit-learn trabalha bem com Arrays do Numpy (a base numérica do Python científico) e DataFrames do Pandas (depois que você converteu tudo para número!).
- **Exemplo RandomForestClassifier:**

```
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier(random_state=0) # Cria o modelo
X = [[ 1, 2, 3], [11, 12, 13]] # Duas amostras com 3 características
cada
y = [0, 1] # Resultado 0 para a primeira amostra, 1 para a segunda
clf.fit(X, y) # Treina o modelo: aprende que [1,2,3] -> 0 e
[11,12,13] -> 1
# Pede previsões para novas amostras:
print(clf.predict([[4, 5, 6], [14, 15, 16]]))
# Saída: [0 1] -> O modelo generalizou que números baixos
(~1,2,3) levam a 0
# e números altos (~11,12,13) levam a 1.
```

Este é um exemplo muito simples, mas ilustra o fluxo fit -> predict.

- **Outras Formas de Tokenização (para texto livre):**
    - **Por Letras:** Token = cada letra. Vocabulário pequeno (só alfabeto + símbolos), mas a sequência numérica fica muito longa e complexa para o modelo entender relações.
        - O IESB ... -> [15, 0, 9, 5, 19, ...] (onde 15=O, 0=espaço, 9=I, etc.)
    - **Por Palavras:** Token = cada palavra. Faz mais sentido semanticamente, mas o vocabulário fica enorme (ex: 320 mil palavras em português) e lida mal com palavras novas ou erros de digitação.
        - O IESB tem ... -> [1, 290, 8900, ...] (onde 1=O, 290=IESB, etc. - números ilustrativos)

    - **Por Subpalavras (Radical, Prefixo, Sufixo - N-gramas):** Um meio-termo eficiente. Quebra palavras em partes menores (beb-, -er, -endo). Reduz o vocabulário comparado a palavras inteiras e lida melhor com variações (beber, bebendo) e palavras novas.

- **Embedding:**
    - Uma técnica mais avançada que Tokenização/One-Hot Encoding, especialmente para texto.
    - Representa palavras ou tokens não como um único número (ID) ou um vetor esparso (One-Hot), mas como um **vetor denso** de números (ex: [0.12, -0.45, 0.88, ...], com dezenas ou centenas de dimensões).
    - A **posição** desse vetor em um espaço multi-dimensional captura o **significado semântico** e o **contexto** da palavra. Palavras similares (ex: "rei", "rainha") terão vetores próximos nesse espaço.
    - O mesmo token (ex: a palavra "banco") pode ter embeddings diferentes dependendo do contexto ("sentar no banco" vs "ir ao banco").

Resumo da Ópera (Foco Pareto):

Scikit-learn é sua caixa de ferramentas para construir modelos de Machine Learning. O fluxo principal é **fit(X, y) para treinar** (mostrar exemplos X e respostas y) e **predict(X_novo) para usar** (prever respostas para novos dados). Lembre-se: modelos entendem **números**. Dados categóricos (texto) precisam ser convertidos. **Label Encoding** é simples (um número por categoria) mas pode confundir o modelo. **One-Hot Encoding** é mais seguro (cria colunas 0/1 para cada categoria) e é o método preferido para dados categóricos nominais.
Dominar fit, predict e como preparar dados numéricos (incluindo One-Hot Encoding para categorias) é o essencial para começar com Scikit-learn.