📄 Análise e Recomendação de Escolas com base nos dados do IDEB

Objetivo do Projeto

O objetivo é carregar os dados do **IDEB (Índice de Desenvolvimento da Educação Básica)**, pré-processá-los, treinar um modelo de aprendizado de máquina e, por fim, gerar **recomendações de escolas semelhantes** com base em características como rede de ensino, taxa de aprovação e notas do SAEB.

🔹 1. Importação de bibliotecas

```
1   import pandas as pd
2   import numpy as np
3   from sklearn.preprocessing import StandardScaler
4   from sklearn.neighbors import NearestNeighbors
```

Explicação:

- `pandas` e `numpy`: bibliotecas para manipulação e análise de dados.
- `StandardScaler`: ferramenta do Scikit-learn que normaliza os dados para que fiquem numa mesma escala.
- `NearestNeighbors`: algoritmo que encontra os elementos mais próximos com base em similaridade (modelo de aprendizado não supervisionado).

🔹 2. Carregamento dos dados

```python
import pandas as pd

def carregar_dados_ideb(caminho):
    """Carrega dados do IDEB"""
    return pd.read_csv(caminho, encoding='ISO-8859-1')
```

Explicação:

- A função recebe um caminho de arquivo CSV como parâmetro e carrega os dados com `pandas.read_csv`.
- O `encoding='ISO-8859-1'` é usado para garantir a leitura correta de acentos e caracteres especiais em português.

🔹 3. Pré-processamento dos dados

📌 Função: `preprocessar_dados(df)`

Essa função tem como objetivo **limpar**, **selecionar** e **normalizar** os dados do IDEB (Índice de Desenvolvimento da Educação Básica), deixando tudo pronto para análise ou uso em modelos de machine learning.

🔹 1. Começo da função

```python
def preprocessar_dados(df):
    """Filtra e normaliza os dados relevantes"""
```

Essa linha define uma **função** chamada `preprocessar_dados`, e o comentário entre aspas serve como descrição: ela vai filtrar (selecionar) e normalizar os dados mais importantes.

🔸 2. Filtro: ensino médio e último ano

```
# Filtro: último ano disponível e ensino médio apenas
df = df[(df['ano'] == df['ano'].max()) & (df['ensino'] == 'medio')].copy()
```

Aqui estão acontecendo duas coisas:

1. **`df['ano'] == df['ano'].max()`**: mantém **apenas as linhas do ano mais recente** disponível no dataset.

2. **`df['ensino'] == 'medio'`**: filtra para **pegar somente dados do Ensino Médio**.

3. **`.copy()`**: cria uma cópia dos dados filtrados para evitar problemas ao modificar o DataFrame original.

💡 *Exemplo:* se o dataset tem dados de 2017 a 2021, essa linha mantém **somente as escolas do Ensino Médio em 2021**.

🔸 3. Transformar redes em números

```
# Mapeia tipos de rede
redes_dict = {'municipal': 0, 'estadual': 1, 'privada': 2, 'total': 3}
df['rede_num'] = df['rede'].map(redes_dict)
```

Aqui, o código pega a coluna `rede`, que contém valores como "municipal", "estadual", etc., e **transforma em números** com base no dicionário `redes_dict`.

- municipal → 0
- estadual → 1
- privada → 2
- total → 3

Isso é necessário porque os modelos de machine learning **não entendem texto**, apenas números.

`map()` para ser mais preciso, serve para aplicar uma função a cada item de um iterável (como uma lista) e retornar um novo iterável com os resultados. Em outras palavras, ela "mapeia" cada valor original para um novo valor, de acordo com a função que foi passada.

🔸 4. Remover linhas com dados faltando

```
# Remove linhas com dados faltantes
features = ['rede_num', 'taxa_aprovacao',
            'nota_saeb_matematica', 'nota_saeb_lingua_portuguesa']
df = df.dropna(subset=features + ['ideb'])
```

- Aqui definimos a lista de **colunas importantes** para análise.
- `dropna(subset=...)` remove qualquer linha que tenha **valores ausentes (NaN)** nessas colunas.

📌 *Isso garante que só vamos trabalhar com dados completos.*

🔸 5. Normalização dos dados

```
# Normalização
scaler = StandardScaler()
df_norm = pd.DataFrame(
    scaler.fit_transform(df[features]),
    columns=[f"{col}_norm" for col in features]
)
```

O que acontece aqui:

- `StandardScaler()` cria o "normalizador".
- `fit_transform(...)` **padroniza** os dados:
    - Para cada valor, ele subtrai a média da coluna e divide pelo desvio padrão.

    - O resultado é que todas as colunas ficam com **média 0** e **desvio padrão 1**, o que ajuda o modelo a comparar variáveis na mesma escala.

- O resultado é um novo DataFrame `df_norm` com os nomes das colunas modificados (por exemplo, `rede_num_norm`, `taxa_aprovacao_norm`, etc.).

🔸 6. Junta os dados normalizados ao original

```
return pd.concat([df.reset_index(drop=True), df_norm], axis=1)
```

`reset_index(drop=True)`: reorganiza o índice do DataFrame, começando do 0.

`pd.concat(..., axis=1)`: junta os dados **lado a lado**, colando as colunas normalizadas no DataFrame original.

🔹 3. Modelagem

Função: `treinar_modelo(df)`

Essa função tem como objetivo **treinar um modelo de vizinhos mais próximos** (Nearest Neighbors), que é um método para encontrar os pontos (dados) mais parecidos ou próximos entre si com base em algumas características numéricas.

Contexto:

O modelo é usado para identificar, por exemplo, quais escolas são mais parecidas entre si, considerando algumas variáveis normalizadas.

```
from sklearn.neighbors import NearestNeighbors
```

Aqui importamos o algoritmo `NearestNeighbors` do pacote scikit-learn, que vai ajudar a encontrar os dados mais próximos uns dos outros.

```
features_norm = ['rede_num_norm', 'taxa_aprovacao_norm',
                 'nota_saeb_matematica_norm', 'nota_saeb_lingua_portuguesa_norm']
```

Definimos as colunas que serão usadas para treinar o modelo.

Essas colunas são as versões **normalizadas** (padronizadas para a mesma escala) das variáveis: tipo de rede, taxa de aprovação e notas do SAEB em matemática e língua portuguesa.

Usar colunas normalizadas evita que variáveis com valores maiores "pese" mais na comparação.

```
X = df[features_norm].values
model = NearestNeighbors(n_neighbors=3, metric='cosine')
model.fit(X)
return model
```

`df[features_norm]` pega apenas as colunas listadas, formando uma matriz (tabela) com só esses dados.

`.values` transforma esse pedaço do DataFrame em um array do NumPy, que é o formato esperado para o modelo.

Aqui criamos o modelo de vizinhos mais próximos.

`n_neighbors=3` diz que queremos encontrar os **3 vizinhos mais próximos** de qualquer ponto.

`metric='cosine'` define que a medida de distância será o **cosseno de ângulo** entre os vetores de características, que é uma forma comum de medir similaridade (quanto mais próximo de 0, mais parecidos).

🔹 3. Recomendação

```python
import numpy as np

def recomendar_similares(model, df, rede_nome, n=3):
    """Recomenda escolas similares com base na rede"""
    redes_dict = {'municipal': 0, 'estadual': 1, 'privada': 2, 'total': 3}
    rede_num = redes_dict.get(rede_nome)

    if rede_num is None:
        print("Rede inválida.")
        return pd.DataFrame()
```

Importação da biblioteca

- Aqui estamos importando a biblioteca **NumPy**, que é muito usada para cálculos matemáticos e trabalhar com arrays (vetores/matrizes)
- A renomeamos como `np` para usar de forma mais curta no código

Definição da função

- `def` cria uma nova função chamada `recomendar_similares`
- A função recebe 4 parâmetros:
  - `model`: provavelmente um modelo de machine learning treinado
  - `df`: um DataFrame (tabela de dados) que contém informações sobre as escolas
  - `rede_nome`: uma string com o nome da rede de ensino (ex: "municipal")
  - `n=3`: número de recomendações a retornar (valor padrão é 3)

Dicionário de redes

- Cria um dicionário que mapeia nomes de redes de ensino para números
- Por exemplo: "municipal" vira 0, "estadual" vira 1, etc.
- Isso é útil porque modelos de ML geralmente trabalham melhor com números do que com texto

Conversão do nome da rede para número

- Usa o método `.get()` do dicionário para converter o `rede_nome` em seu número correspondente
- Por exemplo, se `rede_nome` for "privada", `rede_num` será 2

Verificação de rede inválida

- Verifica se o `rede_nome` fornecido não existe no dicionário (retornando `None`)
- Se for inválido:
    - Imprime uma mensagem de erro
    - Retorna um DataFrame vazio (precisa ter `import pandas as pd` no código completo)

```
target_df = df[df['rede_num'] == rede_num].copy()
if target_df.empty:
    print("Nenhuma escola encontrada para essa rede.")
    return pd.DataFrame()

features_norm = ['rede_num_norm', 'taxa_aprovacao_norm',
                 'nota_saeb_matematica_norm', 'nota_saeb_lingua_portuguesa_norm']
```

Filtrando o DataFrame pela rede de ensino

- `df['rede_num'] == rede_num`: Cria uma condição que verifica quais linhas do DataFrame têm o valor `rede_num` na coluna 'rede_num'
- `df[condição]`: Filtra o DataFrame, mantendo apenas as linhas que satisfazem a condição
- `.copy()`: Cria uma cópia independente do DataFrame filtrado (evita problemas com modificações acidentais no DataFrame original)
- `target_df`: Armazena o resultado (um novo DataFrame apenas com escolas da rede especificada)

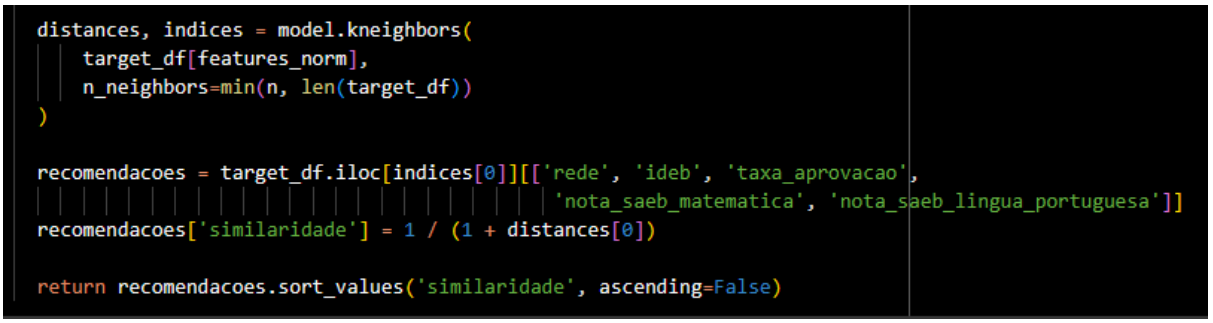
Verificando se há escolas na rede

- `target_df.empty`: Verifica se o DataFrame filtrado está vazio (não há escolas daquela rede)
- Se estiver vazio:
    - Imprime uma mensagem informando que não foram encontradas escolas
    - Retorna um DataFrame vazio (para evitar erros no resto do código)

Definindo as features normalizadas

- Cria uma lista chamada `features_norm` com os nomes das colunas que representam:

- `rede_num_norm`: Rede de ensino normalizada (valores convertidos para uma escala padrão)
- `taxa_aprovacao_norm`: Taxa de aprovação normalizada
- `nota_saeb_matematica_norm`: Nota do SAEB (Sistema de Avaliação da Educação Básica) em Matemática normalizada
- `nota_saeb_lingua_portuguesa_norm`: Nota do SAEB em Língua Portuguesa normalizada



```
distances, indices = model.kneighbors(
    target_df[features_norm],
    n_neighbors=min(n, len(target_df))
)

recomendacoes = target_df.iloc[indices[0]][['rede', 'ideb', 'taxa_aprovacao',
                                            'nota_saeb_matematica', 'nota_saeb_lingua_portuguesa']]
recomendacoes['similaridade'] = 1 / (1 + distances[0])

return recomendacoes.sort_values('similaridade', ascending=False)
```

Buscando os vizinhos mais próximos

- `model.kneighbors()`: Método do modelo de machine learning que encontra os itens mais similares (vizinhos mais próximos)
  - `target_df[features_norm]`: Fornece as características normalizadas das escolas da rede alvo
  - `n_neighbors=min(n, len(target_df))`: Define quantos vizinhos buscar (o menor valor entre `n` pedido e o número de escolas disponíveis)
- Retorna dois arrays:
  - `distances`: Distâncias calculadas entre as escolas (quanto menor, mais similar)
  - `indices`: Posições/índices das escolas mais similares no DataFrame

Criando o DataFrame de recomendações

- `indices[0]`: Pega os índices das escolas mais similares (o [0] é necessário porque kneighbors retorna uma lista de arrays)
- `target_df.iloc[indices[0]]`: Seleciona as linhas correspondentes aos índices no DataFrame filtrado
- `[['rede', 'ideb', ...]]`: Seleciona apenas as colunas específicas que serão mostradas no resultado

Adicionando a medida de similaridade

- Transforma as distâncias em valores de similaridade (entre 0 e 1)
    - `1 / (1 + distance)`: Fórmula comum para converter distâncias em similaridades
        - Distância 0 → Similaridade 1 (idêntico)
        - Distância grande → Similaridade próxima de 0

Retornando os resultados ordenados

- `sort_values('similaridade', ascending=False)`: Ordena as recomendações pela similaridade (maiores primeiro)
- Retorna o DataFrame final com as escolas mais similares, ordenadas da mais para a menos similar

Fluxo Completo:

1. O modelo encontra as escolas mais próximas no espaço de características
2. Selecionamos as informações relevantes dessas escolas
3. Convertemos distâncias em similaridades (mais intuitivo)
4. Ordenamos e retornamos os resultados

```python
def recomendar_similares(df, modelo, rede_escolhida, top_n=5):
    filtro = df[df['rede'] == rede_escolhida]

    if filtro.empty:
        print(f"Nenhum dado encontrado para a rede: {rede_escolhida}")
        return

    idx = filtro.index[0]
    similares = modelo[idx]

    indices_recomendados = similares.argsort()[::-1][1:top_n+1]  # ignora o próprio
    recomendados = df.iloc[indices_recomendados]

    print(recomendados[['ano', 'rede', 'ensino', 'ideb']])
```

Definição da função

- Define uma função que recebe:
    - `df`: DataFrame com os dados das escolas
    - `modelo`: Modelo de similaridade pré-calculado
    - `rede_escolhida`: Rede de ensino para filtrar (ex: "municipal")
    - `top_n`: Quantidade de recomendações (padrão = 5)

Filtragem por rede(Filtro)

Cria um DataFrame **filtrado** contendo apenas escolas da rede especificada

Verificação de dados(IF)

- Se não houver escolas da rede especificada:
    - Imprime mensagem informativa
    - Retorna `None` (encerra a função)

Identificação da escola de referência(IDX)

- Pega o índice da primeira escola encontrada na rede
- (Esta implementação assume que queremos recomendar similares à primeira escola encontrada da rede)

Busca por similares

- Usa o modelo para obter as similaridades da escola de referência
- `modelo` parece ser uma matriz de similaridade pré-computada

Seleção dos mais similares

- `argsort()`: Ordena os índices pelos valores de similaridade
- `[::-1]`: Inverte a ordem (do maior para o menor)
- `[1:top_n+1]`: Pega do 2º ao top_n+1º item (ignora o 1º que é a própria escola)

Criação do resultado(Recomendados)

Usa os índices encontrados para selecionar as escolas mais similares no DataFrame original

Exibição dos resultados(Print Recomendados)

Mostra apenas as colunas especificadas das escolas recomendadas

```python
def preprocessar_dados(df):
    df = df.copy()

    # Converte colunas para o formato correto, se necessário
    df['ano'] = df['ano'].astype(int)

    # Filtra apenas o ensino médio
    df = df[df['ensino'] == 'medio']

    # Reset index para segurança
    df = df.reset_index(drop=True)

    # Trata valores ausentes nas colunas numéricas
    colunas_numericas = df.select_dtypes(include=[np.number]).columns
    df[colunas_numericas] = df[colunas_numericas].fillna(df[colunas_numericas].mean())

    return df
```

Cópia do DataFrame(Copy)

- Cria uma cópia independente do DataFrame original para evitar modificar os dados originais
- Boa prática para evitar efeitos colaterais

Conversão de tipos de dados(astype)

- Converte a coluna 'ano' para o tipo inteiro (int)
- Garante que operações numéricas funcionem corretamente nesta coluna

Filtragem por nível de ensino(df = df[df['ensino'] == 'medio'])

- Filtra o DataFrame para manter apenas registros onde a coluna 'ensino' tem valor 'medio'
- Remove dados de outros níveis de ensino (fundamental, etc.)

Reindexação

- `reset_index()`: Recria os índices do DataFrame de forma sequencial
- `drop=True`: Descarta o índice antigo (não o mantém como nova coluna)
- Importante após filtrar linhas para evitar problemas com índices inconsistentes

Tratamento de valores ausentes

- `select_dtypes(include=[np.number])`: Seleciona apenas colunas numéricas
- `.columns`: Pega os nomes dessas colunas
- `fillna()`: Substitui valores ausentes (NaN)
- `.mean()`: Calcula a média de cada coluna para usar no preenchimento
- Resultado: Todas as colunas numéricas terão seus valores ausentes substituídos pela média da coluna

Retorno(Return)

Retorna o DataFrame pré-processado

```
print("Colunas com NaN antes de tratar:")
print(df.isna().sum())
```

`df.isna().sum()`

- `df.isna()`: Cria um DataFrame booleano do mesmo tamanho, onde:
  - `True` indica valor ausente (NaN)
  - `False` indica valor presente
- `.sum()`: Soma os valores `True` em cada coluna (como `True`=1 e `False`=0)
  - Resultado: Número de valores ausentes por coluna

```
def treinar_modelo(df):
    colunas_numericas = df.select_dtypes(include=[np.number]).drop(columns=['projecao'])  # <- exclui a coluna
    modelo = NearestNeighbors(metric='cosine')
    modelo.fit(colunas_numericas)
    return modelo
```

Explicação da Função `treinar_modelo()`

Esta função prepara e treina um modelo de machine learning para encontrar itens similares (no caso, escolas) baseado em suas características numéricas. Vamos entender detalhadamente:

Seleção de Colunas Numéricas

- `select_dtypes(include=[np.number])`: Seleciona apenas colunas com dados numéricos
- `.drop(columns=['projecao'])`: Remove especificamente a coluna 'projecao' (provavelmente para não usá-la como feature)
- Resultado: DataFrame apenas com as características numéricas relevantes

Criação do Modelo

- `NearestNeighbors`: Algoritmo que encontra itens similares (vizinhos mais próximos)
- `metric='cosine'`: Usa similaridade cosseno como medida de distância
    - Calcula o ângulo entre vetores (ótimo para dados não-normalizados)
    - Range: -1 (opostos) a 1 (idênticos), com 0 indicando independência

Treinamento do Modelo

- `.fit()`: Treina o modelo com os dados fornecidos
- O modelo aprenderá as relações entre as escolas no espaço multidimensional das features

Retorno do Modelo

Retorna o modelo treinado pronto para fazer recomendações

```
    print("\n🔹 Gerando recomendações...\n")
    redes_para_testar = ['estadual', 'municipal', 'privada']

    for rede in redes_para_testar:
        print(f"▶ Recomendação para rede: {rede}")
        resultado = recomendar_similares(modelo, df_proc, rede)
        if not resultado.empty:
            print(resultado.to_markdown(tablefmt='grid'))
        else:
            print("Nenhuma recomendação encontrada.\n")

if __name__ == '__main__':
    main()
```

Explicação da Função `main()`

Esta é a função principal que orquestra todo o fluxo do programa, desde o carregamento até o pré-processamento dos dados. Vamos analisar passo a passo:

Definição do caminho do arquivo

- Define o local do arquivo CSV contendo os dados do IDEB (Índice de Desenvolvimento da Educação Básica)
- O caminho `/content/` sugere que o código está rodando no Google Colab

Carregamento dos dados

- Mostra mensagem indicando o início do carregamento
- Chama a função `carregar_dados_ideb()` (que deve estar definida em outro lugar)
- Exibe a dimensão dos dados carregados (número de linhas e colunas) usando `df.shape`

Pré-processamento dos dados

- Inicia o estágio de pré-processamento com uma mensagem
- Usa um bloco `try-except` para tratamento seguro de erros:
    - Tenta executar `preprocessar_dados(df)`
    - Se der certo, mostra mensagem de sucesso
    - Se falhar, captura a exceção (`e`) e mostra mensagem de erro
    - O `return` encerra a função se ocorrer erro

```
print("\n🔹 Treinando modelo...")
try:
    modelo = treinar_modelo(df_proc)
    print("Modelo treinado com sucesso.")
except Exception as e:
    print(f"Erro no treinamento: {e}")
    return

print("\n🔹 Gerando recomendações...\n")
redes_para_testar = ['estadual', 'municipal', 'privada']
```

Treinamento do Modelo

- **Mensagem de status**: Indica o início do processo de treinamento
- **Bloco try-except**: Tratamento seguro de erros durante o treinamento
    - `treinar_modelo(df_proc)`: Chama a função de treinamento com os dados pré-processados
    - Em caso de sucesso, confirma o treinamento
    - Em caso de erro, mostra a mensagem de erro e encerra a função (`return`)

Preparação para Recomendações

- **Mensagem de status**: Indica o início da fase de geração de recomendações
- **Lista de redes**: Define quais redes de ensino serão analisadas
    - 'estadual': Escolas do governo estadual
    - 'municipal': Escolas do governo municipal
    - 'privada': Escolas particulares

Fluxo Completo (implícito):

1. Para cada rede na lista `redes_para_testar`:
    - Chamaria `recomendar_similares(df_proc, modelo, rede)`
    - Mostraria os resultados para cada tipo de escola

Por que esta estrutura é importante?

1. **Modularidade**: Separa claramente as etapas do processo
2. **Robustez**: O tratamento de erros evita falhas catastróficas
3. **Escalabilidade**: Fácil adicionar mais redes para testar
4. **Clareza**: Mensagens informam o progresso do sistema

```
    print("\n🔹 Gerando recomendações...\n")
    redes_para_testar = ['estadual', 'municipal', 'privada']

    for rede in redes_para_testar:
        print(f"▶ Recomendação para rede: {rede}")
        resultado = recomendar_similares(modelo, df_proc, rede)
        if not resultado.empty:
            print(resultado.to_markdown(tablefmt='grid'))
        else:
            print("Nenhuma recomendação encontrada.\n")

if __name__ == '__main__':
    main()
```

Loop de Recomendações por Rede de Ensino

1. **Iteração pelas redes**: Percorre cada tipo de rede na lista `redes_para_testar`
2. **Cabeçalho de seção**: Mostra qual rede está sendo analisada (com símbolo ▸ para destaque)
3. **Geração de recomendações**: Chama a função `recomendar_similares()` passando:
    - O modelo treinado
    - Os dados pré-processados
    - O tipo de rede atual

Tratamento dos Resultados

1. **Verificação de resultados**:
    - Se o DataFrame `resultado` não estiver vazio (`not empty`)
    - Exibe os resultados formatados como tabela Markdown com bordas (`grid`)
2. **Caso sem resultados**:
    - Mostra mensagem indicando que não foram encontradas recomendações
    - Adiciona quebra de linha (`\n`) para separação visual

Execução do Programa

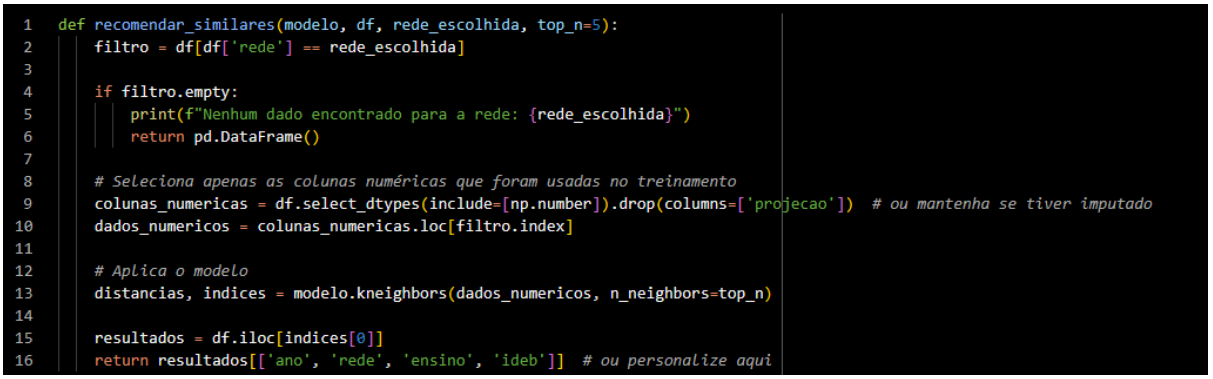
1. **Padrão Python**: Garante que o código só execute quando o script é rodado diretamente
2. **Ponto de entrada**: Chama a função `main()` que orquestra todo o fluxo do programa

Fluxo Completo Visualizado:

1. Para cada rede de ensino (estadual, municipal, privada):
    - Mostra qual rede está sendo processada
    - Gera recomendações de escolas similares
    - Exibe os resultados formatados ou mensagem de ausência
2. Formatação profissional usando:
    - Símbolos visuais (▸)
    - Tabelas bem estruturadas
    - Mensagens claras para casos vazios

Esta estrutura fornece:

- Organização clara dos resultados
- Visualização profissional dos dados
- Tratamento elegante de casos sem resultados
- Execução controlada do programa principal



```python
def recomendar_similares(modelo, df, rede_escolhida, top_n=5):
    filtro = df[df['rede'] == rede_escolhida]

    if filtro.empty:
        print(f"Nenhum dado encontrado para a rede: {rede_escolhida}")
        return pd.DataFrame()

    # Seleciona apenas as colunas numéricas que foram usadas no treinamento
    colunas_numericas = df.select_dtypes(include=[np.number]).drop(columns=['projecao'])  # ou mantenha se tiver imputado
    dados_numericos = colunas_numericas.loc[filtro.index]

    # Aplica o modelo
    distancias, indices = modelo.kneighbors(dados_numericos, n_neighbors=top_n)

    resultados = df.iloc[indices[0]]
    return resultados[['ano', 'rede', 'ensino', 'ideb']]  # ou personalize aqui
```

Explicação Detalhada da Função `recomendar_similares()`

Esta função é responsável por gerar recomendações de escolas similares com base em um modelo de machine learning pré-treinado. Vamos analisar cada parte:

Filtragem por Rede de Ensino

- Filtra o DataFrame para incluir apenas escolas da rede especificada (municipal, estadual ou privada)
- Cria um novo DataFrame (`filtro`) apenas com essas escolas

Verificação de Dados Existentes(If Filtro)

- Verifica se existem escolas da rede especificada
- Se não existirem:
    - Imprime uma mensagem informativa
    - Retorna um DataFrame vazio para evitar erros

Seleção de Colunas Numéricas

- `select_dtypes`: Seleciona apenas colunas com dados numéricos
- `drop(columns=['projecao'])`: Remove a coluna 'projecao' (se existir)
- `loc[filtro.index]`: Pega apenas os dados numéricos das escolas filtradas

Aplicação do Modelo

- `kneighbors`: Método do modelo que encontra os vizinhos mais próximos
- `n_neighbors=top_n`: Especifica quantas recomendações retornar
- Retorna:
    - `distancias`: Quão similares são as escolas recomendadas
    - `indices`: Posições das escolas recomendadas no DataFrame original

Formatação dos Resultados

- `iloc[indices[0]]`: Seleciona as escolas recomendadas usando seus índices
- Retorna apenas as colunas especificadas (personalizável conforme necessidade)

Fluxo de Trabalho:

1. Filtra escolas por rede
2. Verifica se existem dados
3. Prepara os dados numéricos
4. Busca escolas similares usando o modelo
5. Retorna as recomendações formatadas

Personalização Possível:

- Você pode alterar as colunas retornadas modificando a lista no final
- Pode adicionar cálculo de scores de similaridade baseado nas distâncias
- Pode incluir tratamento especial para outliers