

Sistema Recomendação de filmes

Yan Mendonça

25 de abril de 2025

1- Introdução

1.1 Motivação

Muitas vezes, quando tenho um tempo livre e quero assistir a um filme, acabo ficando perdido entre tantas opções. Mesmo com várias plataformas de streaming disponíveis, nem sempre é fácil encontrar algo que realmente combine com meu gosto. Às vezes passo mais tempo procurando do que assistindo. Essa dificuldade recorrente me motivou a desenvolver uma ferramenta simples, mas funcional, que pudesse me sugerir filmes com base em títulos que eu já gosto. Assim nasceu este projeto: um sistema de recomendação de filmes em Python que, de forma rápida e prática, me ajuda a decidir o que assistir a partir de preferências previamente informadas. A ideia é automatizar um pouco esse processo e tornar a escolha mais rápida e personalizada.

1.2 Sobre a organização do projeto

O projeto foi desenvolvido em Python e está dividido em 3 arquivos principais: `main.py`, `lista.json` e `auxiliar.py`. O objetivo foi manter o código limpo, modular e fácil de entender.

O arquivo `main.py` é o responsável por iniciar a execução do programa, interagir com o usuário, capturar as preferências de filmes e mostrar as recomendações. Ele funciona como a “porta de entrada” do sistema.

Já o arquivo `auxiliar.py` concentra as funções que fazem o trabalho por trás das recomendações: carregar os filmes do arquivo JSON, identificar os gêneros preferidos e montar a lista de sugestões. Com essa separação, o código fica mais organizado e simples de dar manutenção ou evoluir no futuro.

Além disso, o projeto utiliza um arquivo chamado `lista.json`, que funciona como a base de dados dos filmes. Cada filme está descrito por um título e um gênero assim possibilitado ser alterado a listagem de filme e tendo sempre atualizado. Essa separação entre o “cérebro” (`auxiliar.py`) e o “rosto”

(main.py) do programa ajuda a manter tudo mais organizado. E se eu quiser

mudar como os dados são lidos ou adicionar novos filtros no futuro? Basta editar o módulo certo, sem bagunçar o restante. O código completo está em:

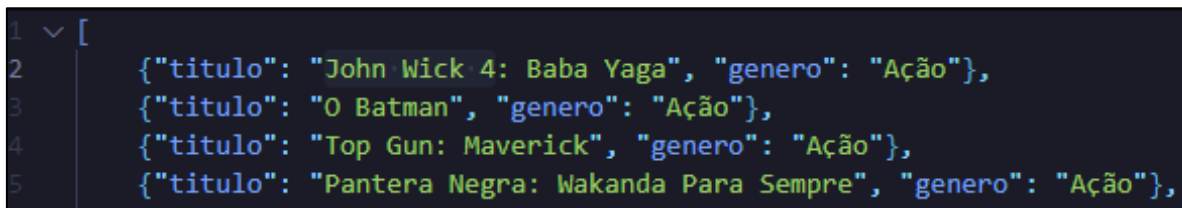
<https://github.com/yanzao420/listafilme>

2. A base de dados - lista.json

Esse arquivo é onde ficam armazenados os filmes. Cada item da lista é um dicionário que contém:

- o título do filme;
- o gênero correspondente.

Exemplo de estrutura:



```
1  [
2    {"titulo": "John Wick 4: Baba Yaga", "genero": "Ação"},
3    {"titulo": "O Batman", "genero": "Ação"},
4    {"titulo": "Top Gun: Maverick", "genero": "Ação"},
5    {"titulo": "Pantera Negra: Wakanda Para Sempre", "genero": "Ação"},
```

Figura 1

Na Figura 1', vemos uma estrutura de dados onde são armazenadas informações de filmes em formato de dicionários. Cada filme possui dois atributos principais: o título, que descreve o nome do filme, e o gênero, que no caso mostrado é sempre "Ação". O sistema usa essas informações para identificar o gênero dos filmes preferidos e para buscar sugestões de títulos semelhantes.

3. O código principal – main.py

O arquivo main.py é o ponto de partida do sistema de recomendação de filmes. Ele concentra a lógica principal do programa e organiza o fluxo de execução, desde a leitura da base de dados até a exibição das recomendações. Ao ser executado, ele guia o programa por todas as etapas necessárias para entregar ao usuário uma lista de sugestões baseadas em seus gostos pessoais.

O programa começa com a importação de algumas bibliotecas importantes. A biblioteca json é usada para ler os dados do arquivo lista.json, que contém os filmes disponíveis. A estrutura Counter, da biblioteca collections, é utilizada para contar quantas vezes cada gênero aparece, ajudando na

identificação do gênero mais comum entre as preferências do usuário. Por fim, são importadas do arquivo auxiliar.py as funções que executam tarefas específicas, como carregar filmes, coletar preferências e gerar recomendações.

Dentro da função main(), o programa imprime uma mensagem de boas-vindas no terminal para sinalizar o início da execução. Em seguida, ele carrega a base de dados dos filmes utilizando a função carregar_filmes_json(). Esse arquivo contém uma lista de filmes, cada um com um título e um gênero, e serve como referência para todo o funcionamento do sistema.

Segue exemplo na Figura 2:

```
1 import json
2 from collections import Counter
3 from auxiliar import carregar_filmes_json, coletar_preferencias, encontrar_generos_preferidos, recomendar_filmes
4
5
6 def main():
7     print("=== Recomendador de Filmes ===\n")
8
9     filmes = carregar_filmes_json("lista.json")
```

Figura 2

Logo após o carregamento dos dados, o programa solicita ao usuário que informe pelo menos um filme de que goste. Isso é feito com a função coletar_preferencias(), que captura a entrada do usuário e a armazena em uma lista chamada preferencias. Com base nos filmes informados, o programa chama a função encontrar_generos_preferidos(), que procura esses títulos na base de dados e retorna os gêneros correspondentes. Caso nenhum dos filmes digitados pelo usuário seja encontrado no arquivo JSON, o programa exibe uma mensagem informando que os filmes não estão na base e finaliza a execução.

Se os gêneros forem encontrados, o próximo passo é gerar as recomendações. Para isso, o sistema utiliza a função recomendar_filmes(), que identifica o gênero mais frequente entre os preferidos e busca, dentro da base, outros filmes do mesmo gênero que o usuário ainda não tenha citado. O resultado dessa busca é armazenado em uma lista chamada recomendacoes.

Segue o exemplo na figura 3:

```
preferencias = coletar_preferencias()

generos_preferidos = encontrar_generos_preferidos(preferencias, filmes)

if not generos_preferidos:
    print("\nNenhum dos filmes informados está na base de dados.")
    return

recomendacoes = recomendar_filmes(generos_preferidos, filmes, preferencias)
```

Figura 3

Por fim, o programa exibe as recomendações na tela. Se houver sugestões compatíveis, elas são listadas uma a uma. Caso contrário, uma

mensagem é exibida informando que nenhuma recomendação foi encontrada com base nos gostos informados.

A função `main()` é encerrada naturalmente, e o programa possui ainda uma verificação final com `if __name__ == "__main__":` que garante que a execução só ocorra se o arquivo for rodado diretamente, e não quando for importado como módulo em outro código. Essa organização torna o programa simples, direto e fácil de entender, além de permitir que ele seja facilmente expandido com novas funcionalidades, como suporte a múltiplos filmes de entrada ou integração com uma interface gráfica.

Segue o exemplo na figura 4:

```
recomendacoes = recomendar_filmes(generos_preferidos, filmes, preferencias)

print("\n🎬 Recomendações de filmes para você:")
if recomendacoes:
    for filme in recomendacoes:
        print(f"- {filme}")
else:
    print("Nenhuma recomendação encontrada com base nos seus gostos.")

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

Figura 4

4. O módulo auxiliar – auxiliar.py

O arquivo `auxiliar.py` concentra a lógica do sistema de recomendação. Ele foi criado com o objetivo de organizar o código, separando as funcionalidades que trabalham com dados da parte que interage com o usuário (`main.py`). Essa separação ajuda a deixar o programa mais modular, reutilizável e fácil de manter.

4.1 - `carregar_filmes_json(caminho_arquivo)`

```
def carregar_filmes_json(caminho_arquivo):
    with open(caminho_arquivo, "r", encoding="utf-8") as f:
        return json.load(f)
```

Essa função é responsável por abrir o arquivo `lista.json`, que contém os dados dos filmes (título e gênero), e retornar esses dados em formato de lista de dicionários. Ela utiliza a função `json.load()` para interpretar o conteúdo do arquivo.

Complexidade: $O(n)$, onde n é o número de filmes no arquivo, pois todos precisam ser lidos e carregados na memória.

4.2 - coletar_preferencias()

```
def coletar_preferencias():  
    print("Digite 1 filmes que você gosta:")  
    preferencias = []  
    for i in range(1):  
        filme = input("- Filme: ").strip()  
        preferencias.append(filme)  
    return preferencias
```

Essa função interage com o usuário, pedindo que ele informe o nome de um filme de que gosta. No código atual, ela coleta apenas um filme, mas pode facilmente ser adaptada para receber mais entradas.

Complexidade:

- $O(1)$ no estado atual (apenas uma entrada).
- Pode se tornar $O(p)$ se for adaptada para aceitar múltiplas preferências.

4.3 - encontrar_generos_preferidos(preferencias, filmes)

Essa função compara os filmes informados pelo usuário com os que existem na base de dados (filmes). Se houver uma correspondência, ela extrai o gênero do filme e adiciona à lista de gêneros preferidos. Para isso, a função realiza uma busca dupla: para cada preferência, percorre toda a lista de filmes.

```
def encontrar_generos_preferidos(preferencias, filmes):  
    generos = []  
    for filme in preferencias:  
        for item in filmes:  
            if item["titulo"].lower() == filme.lower():  
                generos.append(item["genero"])  
    return generos
```

Complexidade:

- $O(p \times n)$, onde p é o número de preferências e n é o número de filmes na base.
- A comparação `lower()` é constante, mas o laço duplo causa crescimento proporcional.

4.4 - recomendar_filmes(generos_preferidos, filmes, preferencias)

Essa função é responsável por gerar a recomendação final. Ela segue os seguintes passos:

- Usa Counter para descobrir qual é o gênero mais frequente entre os informados pelo usuário.
- Percorre a lista de filmes e filtra apenas os que têm esse gênero e ainda não foram escolhidos como preferidos.
- Retorna uma lista de sugestões.

```
def recomendar_filmes(generos_preferidos, filmes, preferencias):
    contagem = Counter(generos_preferidos)
    genero_mais_comum = contagem.most_common(1)[0][0]
    recomendacoes = []

    for item in filmes:
        if item["genero"] == genero_mais_comum and item["titulo"] not in preferencias:
            recomendacoes.append(item["titulo"])

    return recomendacoes
```

Complexidade:

- Contagem dos gêneros: $O(g)$, onde g é o número de gêneros (em geral, pequeno).
- Laço de filtragem dos filmes: $O(n)$.

5. Conclusão do módulo

O auxiliar.py é um ótimo exemplo de como dá para escrever código simples, direto e funcional, sem precisar complicar. As funções são curtas, bem nomeadas e fazem exatamente o que se propõem a fazer. Mesmo sendo um projeto pequeno, ele já aborda conceitos muito importantes, como leitura de arquivos JSON, filtragem de dados, contagem de elementos e buscas lineares. O que mais chama a atenção nesse módulo é o equilíbrio entre simplicidade e eficiência. Tudo foi pensado para ser fácil de entender e também fácil de modificar no futuro. Se eu quiser mudar a forma como os filmes são lidos, ou incluir outros critérios de recomendação, posso fazer isso tranquilamente sem mexer na estrutura principal.

Além disso, como o código está bem separado do main.py, ele pode ser reaproveitado em outros projetos parecidos — seja um recomendador de músicas, livros ou até séries. Isso mostra o quanto a organização e a clareza no código fazem diferença, especialmente em projetos que podem crescer com o tempo.