Projeto de Bloco: Ciência de Dados Aplicada

Teste de Performance 3 (TP3)

Instituto Infnet - Rafael Dottori de Oliveira

30/11/2024

Link para o GitHub

https://github.com/R-Dottori/filme-em-foco

Enunciado

Esta é uma etapa importante do seu projeto! Agora, você terá a oportunidade de agregar novas funcionalidades e técnicas à aplicação que vem desenvolvendo, evoluindo para uma solução mais robusta e interativa. O objetivo desta etapa é ampliar as capacidades da sua aplicação, explorando novas ferramentas como a criação de múltiplas páginas, o desenvolvimento de APIs e a coleta de dados de páginas dinâmicas, preparando o terreno para o uso de recursos de Inteligência Artificial nas próximas etapas.

> *Objetivo:*

No terceiro TP do projeto, você deverá revisar a proposta inicial, integrar novos recursos e tecnologias ao projeto, e continuar documentando todo o processo de desenvolvimento. A implementação dessas funcionalidades ampliará o escopo do projeto e oferecerá novas maneiras de interagir com os dados. Além disso, é importante que você comece a considerar como os dados coletados até o momento poderão ser usados em conjunto com as técnicas de LLMs (Large Language Models) que serão abordadas nas etapas seguintes.

> *Dicas:*

```
Mantenha a organização: Continue utilizando Git para
versionamento do código e assegure-se de que seu repositório
esteja bem organizado, com documentação clara para cada
funcionalidade implementada.

Teste as funcionalidades: Certifique-se de que todas as novas
funcionalidades, como o menu de navegação e as rotas da API,
estejam funcionando corretamente.

Simplifique se necessário: Embora o uso de Selenium seja uma
possibilidade poderosa para scraping dinâmico, avalie se essa
```

ferramenta é realmente necessária no contexto do seu projeto. Mantenha a simplicidade sempre que possível.

Visão de longo prazo: Assegure-se de que todas as decisões técnicas tomadas nesta etapa estejam alinhadas com o objetivo final de criar uma solução completa que, na próxima fase, integrará recursos de IA via LLM.

> *Entrega Final do TP3:*

Ao final desta etapa, sua aplicação deverá:

Possuir um menu de navegação que permita o uso de múltiplas páginas;

Estar integrada com uma API construída com FastAPI;

Ter a documentação atualizada, incluindo a revisão do Project Charter e Data Summary Report;

Estar preparada para a próxima etapa, onde serão introduzidos recursos de IA via LLM.

Exercício 1: Revisão e Atualização da Documentação

Revise o Project Charter e o Data Summary Report, atualizando a documentação para refletir as novas funcionalidades e decisões tomadas nesta fase do projeto.

Reavalie o problema de negócio à luz das novas ferramentas (como FastAPI e Selenium) e ajuste suas metas, se necessário.

Atualize a descrição das fontes de dados utilizadas, considerando possíveis novas fontes obtidas com scraping dinâmico.

1-A: Resumo do Aplicativo

Nessa etapa, mudamos a natureza do projeto.

O foco ainda é desenvolver um **aplicativo sobre cinema** seguindo o princípio 3 dos Objetivos de Desenvolvimento Sustentável — Boa Saúde e Bem-Estar.

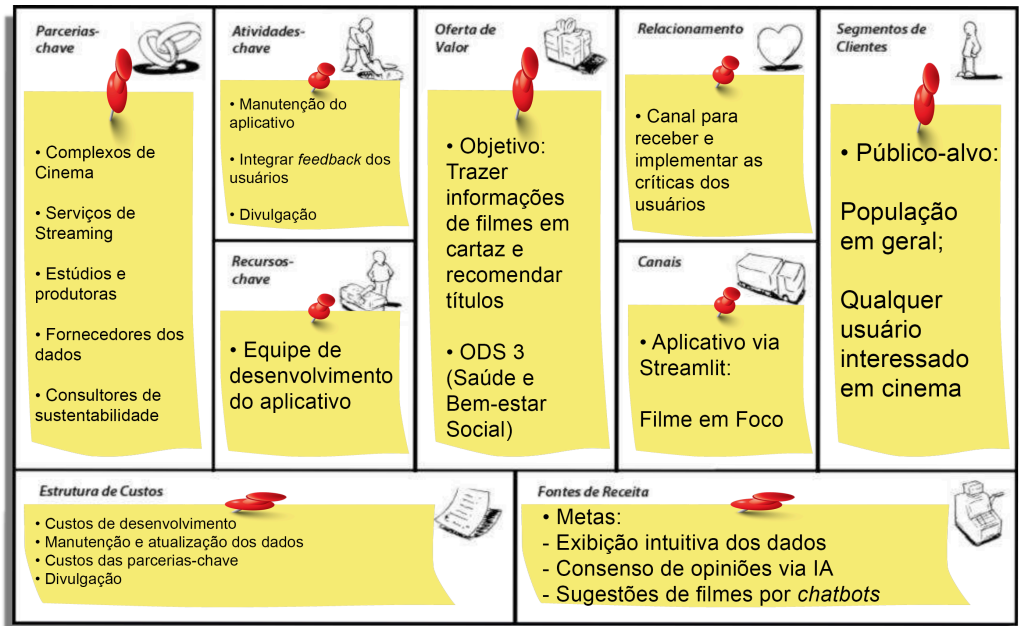
O argumento principal se apoia em artigos e notícias que confirmam que a cultura é um pilar importante para o crescimento de uma sociedade. Isso inclui garantir o acesso e o engajamento com diversos mídias diferentes.

Além disso, a nova direção do aplicativo também irá abarcar as novas funcionalidades exigidas: raspagem dinâmica com Selenium, comunicação dos dados com a aplicação via FastAPI e a inclusão de modelos de inteligência artificial e LLM.

De maneira geral, o aplicativo do Filme em Foco contará com:

- Uma lista com os filmes em cartaz
- Um consenso das críticas dos usuários, gerado por modelos de análise de sentimentos e "resumidores" de texto (summarizers)
- Sugestões de filmes por um *chatbot*

1-B: Project Charter - Business Model Canvas



1-C: Data Summary Report

• API do Ingresso.com:

— Fonte: Consultas na API do Ingresso.com

— Formato: Dados textuais em tabelas .CSV

— Objetivo: Informações dos filmes em cartaz

— — —

• Informações do IMDb:

— Fonte: Raspagem de dados de filmes no portal IMDb

— Formato: Dados textuais contendo informações como título do filme, diretor, análises dos usuários, etc.

— Objetivo: Alimentar os modelos de IA com um catálogo expansivo de filmes.

— — —

• Mais dados de outros portais sobre cinema:

— Fonte: Raspagem de dados de outros portais, como Rotten Tomatoes ou catálogos de serviços de *streaming*

— Formato: Dados textuais contendo informações como título do filme, diretor, análises dos usuários, etc.

— Objetivo: Complementar a fonte do IMDb. Por exemplo, para ter análises somente da crítica especializada.

Exercício 2: Criação de uma Aplicação com Múltiplas Páginas

Evolua a interface da sua aplicação em Streamlit, implementando múltiplas páginas e um menu de navegação que permita ao usuário transitar facilmente entre diferentes seções da aplicação.

Cada página deve representar uma funcionalidade ou análise diferente, como a visualização de dados, gráficos interativos, upload/download de arquivos ou estatísticas geradas a partir dos dados coletados.

Para essa etapa do projeto, vamos desenvolver a base das funcionalidades que usam o FastAPI e o Selenium. O processo de raspagem e criação de rotas é explicado melhor nos exercícios que seguem (3 e 4).

Criamos um aplicativo no Streamlit com múltiplas páginas: uma introdução, uma exibição dos filmes raspados via comunicação com API e uma funcionalidade para adicionar um novo filme na base (também gerenciado pela comunicação com a API).

Para rodar o aplicativo de maneira local, ativamos o ambiente virtual digitando no terminal:

> .venv_app/Scripts/activate

Executamos a API com o uvicorn

> uvicorn src.api:app

E enfim executamos o aplicativo do Streamlit

> streamlit run ./src/filme_foco.py

In [83]:
```
%%writefile ./src/filme_foco.py

import streamlit as st
import requests
import pandas as pd

st.set_page_config(
    page_title='Filme em Foco',
```

```
    page_icon='🎥',
)

pag_1_title = 'Página Inicial'
pag_2_title = 'Exibir Filmes'
pag_3_title = 'Adicionar um Filme'


def pagina_um():
    st.title('Filme em Foco')
    st.image('https://daily.kellogg.edu/wp-content/uploads/2018/08/film-interpre


def pagina_dois():
    st.header(pag_2_title)
    filmes = ''
    genero = st.text_input('Buscar por gênero:')
    if genero:
        try:
            resp = requests.get(f'http://127.0.0.1:8000/filmes/genero/{genero}')
            filmes = resp.json()
        except:
            st.error('Erro na comunicação com a API.')
        if filmes != '':
            df_filmes = pd.DataFrame(filmes)
            st.write(df_filmes)

    else:
        try:
            resp = requests.get('http://127.0.0.1:8000/filmes')
            filmes = resp.json()
        except:
            st.error('Erro na comunicação com a API.')
        if filmes != '':
            df_filmes = pd.DataFrame(filmes)
            st.write(df_filmes)


def pagina_tres():
    st.header(pag_3_title)
    titulo = st.text_input('Título do Filme')
    ano = st.number_input('Ano de Lançamento')
    if st.button('Adicionar'):
        if ano and titulo:
            novo_filme = {
                            'titulo': titulo,
                            'titulo_original': titulo,
                            'ano': ano,
                            'duracao': 120,
                            'diretor': 'Diretor',
                            'roteirista': 'Roteirista',
                            'elenco': 'Ator',
                            'nota': 8.0,
                            'num_votos': 10.000,
                            'sinopse': 'Sinopse',
                            'genero': 'Ação',
                            'verba': 100,
                            'receita': 500
                            }
            try:
```

```
                resp = requests.post('http://127.0.0.1:8000/filmes', json=novo_f
                st.success('Filme adicionado com sucesso!')
            except:
                st.error('Erro ao comunicar com a API')
        else:
            st.error('Preencha todos os campos')


st.sidebar.title('Navegação')
pagina = st.sidebar.radio(label='Escolha uma página:', options=(pag_1_title, pag
if pagina == pag_1_title:
    pagina_um()
elif pagina == pag_2_title:
    pagina_dois()
else:
    pagina_tres()
```

```
Overwriting ./src/filme_foco.py
```

Exercício 3: Extração de Dados de Páginas Dinâmicas (Web Scraping)

Utilize o Selenium para realizar o web scraping de páginas dinâmicas, se necessário. Caso seu projeto utilize uma fonte de dados que exija interação com elementos dinâmicos (como formulários ou carregamentos assíncronos), o Selenium será essencial.

Observação: Se não houver necessidade de utilizar Selenium, concentre-se no aprimoramento dos dados coletados com Beautiful Soup ou APIs, mantendo a simplicidade quando possível.

Armazene os dados obtidos em arquivos CSV ou TXT, organizando-os no diretório de data/ para uso na aplicação.

Nessa etapa, faremos a raspagem dos 250 filmes com maior avaliação no IMDb utilizando o Selenium.

Separamos o código em duas células, uma que raspará as informações gerais de cada filme e outra com as análises dos usuários.

Salvaremos os dados em arquivos CSV.

(*https://www.imdb.com/chart/top/*)

In [ ]:
```
from selenium import webdriver
from selenium.webdriver.common.by import By
import time
import csv

driver = webdriver.Chrome()
driver.get('https://www.imdb.com/chart/top/')

lista_filmes = []
```

```
try:
    time.sleep(5)
    urls = driver.find_elements(By.CSS_SELECTOR, 'ul > li > div > div > div > di
    url_filmes = [filme.get_attribute('href') for filme in urls]

    for url in url_filmes:
        driver.get(url)
        time.sleep(5)
        try:
            # Título Traduzido
            try:
                titulo_pt = driver.find_element(By.CSS_SELECTOR, 'span.hero__pri
            except:
                titulo_pt = ''

            # Título Original
            try:
                titulo_og = driver.find_element(By.CSS_SELECTOR, 'div.sc-ec65ba0
                titulo_og = titulo_og[17:] # Removendo a parte do "Título origin
            except:
                titulo_og = titulo_pt

            # Ano e Duração
            try:
                duracao_ano_bruto = driver.find_elements(By.CSS_SELECTOR, 'div.s
                duracao_ano = [info.text for info in duracao_ano_bruto]
                ano = duracao_ano[0]
                duracao = duracao_ano[2]
                if 'h' in duracao:
                    pos_h = duracao.find('h')
                    horas = int(duracao[:pos_h]) * 60
                    pos_min = duracao.find('m')
                    minutos = int(duracao[pos_h + 1:pos_min])
                    duracao = horas + minutos
                else:
                    pos_min = duracao.find('m')
                    duracao = int(duracao[:pos_min])
            except:
                ano = duracao = 0

            # Equipe
            try:
                ficha_tecnica = driver.find_elements(By.CSS_SELECTOR, 'div.sc-70
                diretor = [membro.text for membro in ficha_tecnica if '_dr_' in
                roteirista = [membro.text for membro in ficha_tecnica if '_wr_'
                elenco = [membro.text for membro in ficha_tecnica if '_st_' in m
            except:
                diretor = roteirista = elenco = ''

            # Nota
            try:
                info_nota = driver.find_element(By.CSS_SELECTOR, 'div[class="sc-
                pos_barra = info_nota.find('/10')
                nota = float(info_nota[:pos_barra].replace(',', '.'))
                num_votos = info_nota[pos_barra + 3:].lstrip().replace(',', '.')
                if 'mil' in num_votos:
                    pos_m = num_votos.find('m')
                    num_votos = float(num_votos[:pos_m]) * 1_000
                elif 'mi':
                    pos_m = num_votos.find('m')
```

```
                    num_votos = float(num_votos[:pos_m]) * 1_000_000
            except:
                nota = num_votos = 0

            # Sinopse
            try:
                sinopse = driver.find_element(By.CSS_SELECTOR, 'span[class="sc-3
            except:
                sinopse = ''

            # Verba e Receita
            try:
                verba = driver.find_element(By.CSS_SELECTOR, 'li[data-testid="ti
                pos_dinheiro_verba = verba.find('$') + 1
                pos_estimativa = verba.find('(estimativa)')
                verba = int(verba[pos_dinheiro_verba:pos_estimativa].strip().rep

                receita = driver.find_element(By.CSS_SELECTOR, 'li[data-testid="
                pos_dinheiro_receita = receita.find('$') + 1
                receita = int(receita[pos_dinheiro_receita:].strip().replace('.'
            except:
                verba = receita = 0

            # Gênero
            try:
                elemento_scroll = driver.find_element(By.CSS_SELECTOR, 'hgroup')
                driver.execute_script("arguments[0].scrollIntoView();", elemento
                time.sleep(2)
                generos_bruto = driver.find_elements(By.CSS_SELECTOR, 'div[class
                generos = [genero.text for genero in generos_bruto]
            except:
                generos = ''

            # Adicionando a lista
            filme_raspado = {'titulo': titulo_pt,
                             'titulo_original': titulo_og,
                             'ano': ano,
                             'duracao': duracao,
                             'diretor': diretor,
                             'roteirista': roteirista,
                             'elenco': elenco,
                             'nota': nota,
                             'num_votos': num_votos,
                             'sinopse': sinopse,
                             'genero': generos,
                             'verba': verba,
                             'receita': receita}

            lista_filmes.append(filme_raspado)
        except:
            print('Erro dentro de um filme')
except:
    print('Erro ao obter URLs')
    pass

driver.quit()



# Salvando o conteúdo
```

```
colunas = lista_filmes[0].keys()

with open('./data/top_250_filmes.csv', 'w', newline='', encoding='utf-8') as arq
    writer = csv.DictWriter(arquivo, fieldnames=colunas)
    writer.writeheader()
    writer.writerows(lista_filmes)
```

In [ ]:

```
from selenium import webdriver
from selenium.webdriver.common.by import By
import time
import csv

driver = webdriver.Chrome()
driver.get('https://www.imdb.com/chart/top/')

lista_analises = []

try:
    time.sleep(5)
    urls = driver.find_elements(By.CSS_SELECTOR, 'ul > li > div > div > div > di
    url_filmes = [filme.get_attribute('href')[:-15] + 'reviews/' for filme in ur

    for url in url_filmes:
        driver.get(url)
        time.sleep(5)
        try:
            # Título
            try:
                titulo = driver.find_element(By.CSS_SELECTOR, 'h2').text
            except:
                titulot = ''

            # Análises
            try:
                analises = [analise.text for analise in driver.find_elements(By.
            except:
                analises = ''

            # Adicionando a lista
            filme_raspado = {'titulo': titulo,
                             'analises': analises}

            lista_analises.append(filme_raspado)
        except:
            print('Erro dentro de um filme')
except:
    print('Erro ao obter URLs')
    pass

driver.quit()



# Salvando o conteúdo
colunas = lista_analises[0].keys()

with open('./data/top_250_analises.csv', 'w', newline='', encoding='utf-8') as a
    writer = csv.DictWriter(arquivo, fieldnames=colunas)
```

```
    writer.writeheader()
    writer.writerows(lista_analises)
```

Exercício 4: Desenvolvimento de APIs com FastAPI

Configure o ambiente de desenvolvimento para a criação de APIs com FastAPI.

Crie uma API simples com rotas e endpoints que permitam interagir com os dados da aplicação. Exemplo de funcionalidades da API:

```
Consulta de dados (GET)

Envio de novos dados (POST)
```

Implemente ao menos duas rotas em sua API, e documente-as adequadamente, explicando sua função e como elas podem ser usadas para interação com os dados.

Começamos a montar nossa API. Primeiro, vamos separar os códigos em três arquivos:

Um arquivo "models.py" com os modelos de validação dos dados. Isso garante que a API só aceite os dados no formato exato que indicarmos.

Um arquivo "routers.py" configurando as funções em si para cada rota. A vantagem de manter essas funcionalidades fora do arquivo principal é nos auxiliar a expandir o projeto e manter os códigos. Por exemplo, se tívessemos uma quantidade enorme de rotas, poderíamos separá-las em mais arquivos ainda, para deixar os códigos mais fáceis de entender.

Por fim, um arquivo principal "api.py" que executa a API. Esse arquivo é responsável por referenciar todas as rotas que criamos.

Para esse exercício, criaremos três rotas:

1 - Uma rota GET em "/filmes", que exibirá todas as informações que coletamos no exercício de raspagem de dados.

2 - Outra rota GET em "filmes/genero/{genero}" para testar uma funcionalidade básica de filtro/busca. Assim, podemos exibir os filmes que pertencem a um gênero específico.

3 - Uma rota POST, que nos permite enviar novos dados de um filme. Como dito anteriormente, é necessário seguir o modelo criado para que os dados sejam aceitos.

Para executar a aplicação, rodamos no terminal (com o ambiente virtual ativado):

> uvicorn src.api:app

Com o FastAPI rodando, em outro terminal podemos exibir todos os filmes ou procurar pelos títulos de drama:

> http GET localhost:8000/filmes

> http GET localhost:8000/filmes/genero/drama

Por fim, para adicionar um filme com o método POST:

> http POST localhost:8000/filmes titulo="Teste" titulo_original="Test" ano=1999 duracao=120 diretor="João" roteirista="João" elenco="Maria" nota=8 num_votos=10 sinopse="Sinopse do filme" genero="Drama" verba=1000 receita=2000

In [18]:
```
%%writefile ./src/models.py

from pydantic import BaseModel

class ModeloFilme(BaseModel):
    titulo: str
    titulo_original: str
    ano: int
    duracao: int
    diretor: str
    roteirista: str
    elenco: str
    nota: float
    num_votos: float
    sinopse: str
    genero: str
    verba: int
    receita: int
```

```
Overwriting ./src/models.py
```

In [62]:
```
%%writefile ./src/routers.py

from fastapi import APIRouter
from .models import ModeloFilme
import pandas as pd
from typing import List

router = APIRouter()

@router.get('/filmes', response_model=List[ModeloFilme])
async def get_filmes():
    filmes = pd.read_csv('./data/top_250_filmes.csv').to_dict(orient='records')
    return filmes


@router.get('/filmes/genero/{genero}', response_model=List[ModeloFilme])
async def get_filmes_genero(genero: str):
    filmes = pd.read_csv('./data/top_250_filmes.csv')
    filmes_gen = filmes[filmes['genero'].str.contains(genero, case=False)].to_di
    return filmes_gen


@router.post('/filmes', response_model=ModeloFilme)
async def post_filme(filme: ModeloFilme):
    filmes = pd.read_csv('./data/top_250_filmes.csv')
    novo_filme = pd.DataFrame([filme.dict()])
    filmes = pd.concat([filmes, novo_filme], ignore_index=True)
```

```
    filmes.to_csv('./data/top_250_filmes.csv', index=False)
    return filme.dict()
```

```
Overwriting ./src/routers.py
```

In [63]:
```
%%writefile ./src/api.py

from fastapi import FastAPI
from .routers import router

app = FastAPI()

app.include_router(router)

@app.get('/')
async def raiz():
    return {'mensagem': 'Página raiz'}
```

```
Writing ./src/api.py
```

Exercício 5: Preparação para Uso de Inteligência Artificial com LLMs

Nesta etapa, comece a pensar nos dados que você coletou até agora e como eles podem ser utilizados em tarefas baseadas em LLMs nas próximas entregas.

Considere os tipos de dados disponíveis e as possíveis aplicações com LLMs, como:

```
Análise de Texto Gerado: Usar os dados coletados para gerar
resumos automáticos de textos longos, facilitando a compreensão
e análise de documentos.

Classificação de Sentimentos: Aplicar um modelo de LLM para
classificar os sentimentos em textos (positivos, negativos ou
neutros) coletados de notícias, redes sociais, ou outras fontes.

Perguntas e Respostas (Q&A): Usar um LLM para construir um
sistema de perguntas e respostas a partir dos dados disponíveis,
respondendo a perguntas relevantes com base nos conteúdos
coletados.

Geração de Texto: Automatizar a criação de relatórios ou
insights com base nos dados brutos, utilizando LLMs para gerar
textos descritivos.
```

Mantenha o foco na preparação do projeto para que, na próxima etapa, as funcionalidades de IA via LLM sejam facilmente integradas e aplicadas a esses cenários.

Podemos adiantar algumas ideias que integrarão LLMs com os dados que raspamos:

• Avaliar as críticas em texto dos usuários com análises de sentimento.

• Resumir a opinião geral sobre filmes utilizando "resumidores" (*summarizers*)

• Criar um *chatbot* que possa conversar sobre cinema, incluindo indicar filmes baseados nos gostos do usuário