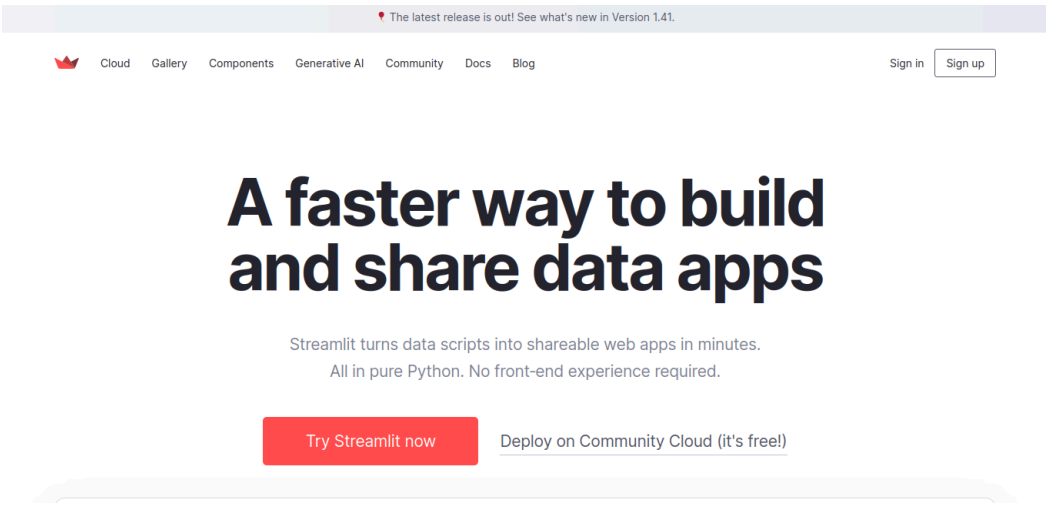
Manual Operacional do RainWeb

O site RainWeb é um produto desenvolvido por alunos da Universidade Federal de Itajubá, do curso de Ciências Atmosféricas para a matéria de Ferramentas de Previsão de Curtíssimo Prazo (Nowcasting) da turma de 2024.2. O site engloba estações automáticas do Centro Nacional de Monitoramento e Alertas de Desastres Naturais (CEMADEN) distribuídas na região sul do estado de Minas Gerais que mostram em tempo real a taxa de precipitação.

Todo o funcionamento do site é feito na linguagem python e é de disponibilizada de forma pública através do Github: https://github.com/Flavio-Aug-San/RainWeb. Esse produto foi feito pelos alunos de graduação: Flávio Augusto dos Santos, Guilherme Lisbôa Silveira e Raul Nicolas Maciel Chaves com a supervisão do professor Enrique Vieira Mattos.

Introdução:

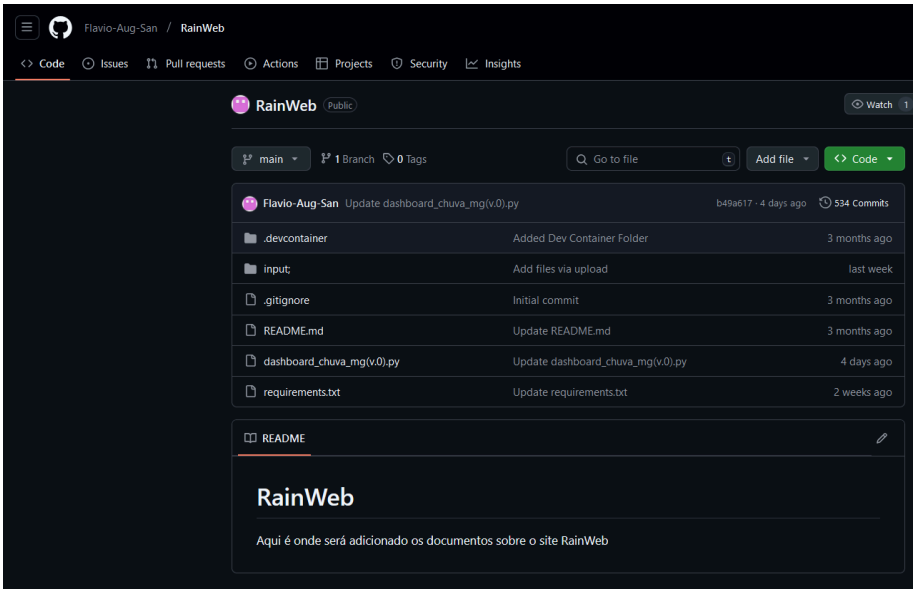
Para o desenvolvimento do dashboard utilizou-se a biblioteca streamlit. O streamlit é um aplicativo que cria sites de maneira rápida e dinâmica, na linguagem python de maneira gratuita. Suas únicas dependências são de ter um cadastro (feito no próprio site do streamlit) de ser necessário ativar o site de maneira regular. O cadastro é feito no campo superior à direita, na caixa 'Sign up'.



Ao clicar na caixa de cadastre-se o usuário será encaminhado a outra aba de cadastro, ao clicar em continuar o Streamlit fornece três opções para cadastro: Com o Google, com o GitHub ou com o cadastro próprio do Streamlit. Os autores do RainWeb sugerem a criação do cadastro com o GitHub. Caso seja necessário crie antes uma conta no GitHub na caixa também de 'Sign up'. A utilização do cadastro do Streamlit com o GitHub vincula as duas contas fornecendo as elas interações que independem de uma máquina com python, sendo possível criar sites apenas editando o GitHub, além do mesmo hospedar o site de maneira também gratuita.

RainWeb:

O Github onde se encontra o produto é de domínio público e pode ser acessado por qualquer pessoa pelo link: https://github.com/Flavio-Aug-San/RainWeb/tree/main. Ao entrar na página o usuário será direcionado a seguinte tela:



Cada arquivo presente representa partes importantes do projeto do site. Sendo:

- .devcontainer e .gitignore = pacotes do Streamlit para o funcionamento do site
- input; = dados de estações do CEMADEN que foram utilizadas com base na criação do site.
- README.md = descrição do site.
- dashboard_chuva_mg(v.0).py = programa em python do nosso dashboard onde foram feitos todos os comandos para a criação do produto.
- requirements.txt = todas as bibliotecas utilizadas na programação do dashboard.

Dentre todos os diretórios presentes no GitHub o arquivo requirements.txt e o dashboard_chuva_mg(v.0).py são os mais importantes. O arquivo requirements.txt contém todas as bibliotecas utilizadas no dashboard, elas precisam estar separadas do código. O Streamlit identifica o nome do arquivo 'requirements.txt' (precisa ter o mesmo nome) e baixa em sua nuvem todas as bibliotecas necessárias. Ao adentrar no arquivo requirements.txt são listadas as seguintes bibliotecas:

```
streamlit
folium
pandas
geopandas
leafmap
salem
requests
datetime
numpy
matplotlib
```

1) Streamlit: biblioteca de hospedagem do site.
2) Folium: biblioteca de criação de marcadores em mapas.
3) Pandas: biblioteca de organização de dados em dataframes.
4) Geopandas: biblioteca de mascaramento de dados com base em shapefiles.
5) Leafmap: biblioteca de criação de mapa interativo do Googlemapas.
6) Salem: outra biblioteca de mascaramento de dados com base em shapefiles.
7) Requests: biblioteca de baixar dados.
8) Datetime: biblioteca transformação de tempo em linha de um dataframe.
9) Numpy: biblioteca de processamento de matriz e linhas.
10) Matplotlib: biblioteca de criação de figuras.

Dashboard parte 1:

Entrando no programa dashboard_chuva_mg(v.0).py o produto apresentará a seguinte configurações:

Primeiramente a importação das bibliotecas anteriormente coletadas:

Code Blame 265 lines (211 loc) · 9.81 KB Raw

```python
import streamlit as st
import pandas as pd
import geopandas as gpd
import requests
import numpy as np
from datetime import datetime, timedelta
import leafmap.foliumap as leafmap
import folium
import glob
import calendar
from io import StringIO
import matplotlib.pyplot as plt
from folium.plugins import MarkerCluster

```

Todas as bibliotecas são chamadas antes de serem usadas no código. Dentre as bibliotecas a biblioteca ‘from io import StringIO’ a biblioteca SringIO é muito importante, ela

antigamente faz parte implicitamente dentro do Pandas. As novas atualizações não compõem essa função.

Todas as bibliotecas e funções chamadas neste começo que não estão dentro dos requerimentos, estão implícitas dentro da programação python base, não sendo necessário baixar, somente importar elas.

Plataforma de Entrega de Dados (PED):

É importante ressaltar que o site coleta dados de pluviômetros de estações automáticas do CEMADEN. O mesmo fornece dados por meio de um portal chamado Plataforma de Entrega de Dados (PED). Esse portal entrega os dados de pluviômetros a empresas e instituições de maneira mais prática por meio de solicitações feitas com requisições automáticas com programação. Para conseguir baixar esses dados é necessário fazer um cadastro no PED e salvar os tokens email e senha do cadastro para fornecer quando fazer a requisição.

Uma pessoa pública pode fazer 12 requisições de dados por minutos. Ao se ultrapassar esse limiar os dados não são baixados, mais informações são adicionadas no perfil inicial do PED do usuário.

Dashboard parte 2:

As seguintes informações mostram o shapefile de Minas Gerais e os dados que utilizamos como padrão na construção do site. Também é salvo o login e senha do PED em forma de variável.

```
# URLs e caminhos de arquivos
shp_mg_url = 'https://github.com/giuliano-macedo/geodata-br-states/raw/main/geojson/br_states/br_mg.json'
csv_file_path = 'input;/filtered_data.csv'

# Login e senha do CEMADEN (previamente fornecidos)
login = '[redacted]'
senha = '[redacted]'

# Carregar os dados do shapefile de Minas Gerais
mg_gdf = gpd.read_file(shp_mg_url)

# Estações Selecionadas do Sul de Minas Gerais
codigo_estacao = ['314790701A','310710901A','312870901A','315180001A','316930702A','314780801A','315250101A','313240401A','313360001A','311410501A','311360201A','313300601A']

# Carregar os dados das estações
df1 = pd.read_csv(csv_file_path)
gdf = gpd.GeoDataFrame(df1, geometry=gpd.points_from_xy(df1['longitude'], df1['latitude']))
```

Como queremos coletar dados apenas para as estações do Sul de Minas, decidimos buscar baixar o dado de 12 estações que entram dentro das 12 requisições do PED. As estações escolhidas foram das cidades: Passos, Boa Esperança, Guaxupé, Poços de Caldas, Três Corações, Passa-Vinte, Pouso Alegre, Itajubá, Itapeva, Carmo de Minas, Careaçu e Itamonte.

Em seguida colocamos os tokens em url que receberão os tokens de forma automática, como forma de informação. O link (http://sgaa.cemaden.gov.br/SGAA/rest/controle-token/tokens) é o mesmo mostrado na imagem abaixo. Em seguida, salvamos em variáveis as informações do dia atual do computador, para auxiliar no acumulado de precipitação diária.

```python
# Recuperação do token
token_url = 'http://sgaa.cemaden.gov.br/SGAA/rest/controle-token/tokens'
login_payload = {'email': login, 'password': senha}
response = requests.post(token_url, json=login_payload)
content = response.json()
token = content['token']

# sigla do estado do Brasil
sigla_estado = 'MG'

# Data de hoje
agora = datetime.now()

# Dia, mês e ano de hoje
dia_atual = agora.day
mes_atual = agora.month
ano_atual = agora.year
```

Para a coleta de mais dias de precipitação, foram feitos os seguintes comandos. Quando está no começo do mês, exemplo dia 01 de outubro, e queremos comparar com o acumulado de 24 horas os de 48 horas é necessário utilizar o comando do if para a coleta dessa informação, caso não seja feito o programa não irá entender pois os valores serão dispostos mês a mês.

```python
44    sigla_estado = 'MG'
45
46    # Data de hoje
47    agora = datetime.now()
48
49    # Dia, mês e ano de hoje
50    dia_atual = agora.day
51    mes_atual = agora.month
52    ano_atual = agora.year
53
54    # Calcula o mês e ano anteriores para a data inicial
55    if mes_atual == 1:
56        mes_anterior = 12
57        ano_anterior = ano_atual - 1
58    else:
59        mes_anterior = mes_atual - 1
60        ano_anterior = ano_atual
61        mes_pos = mes_atual + 1
62
63    # Formata as datas
64    diai = '01'
65    data_inicial = f'{ano_atual}{mes_anterior:02d}{diai}'
66    data_final = f'{ano_atual}{mes_atual:02d}{dia_atual:02d}'
67    data_inicial = pd.to_datetime(data_inicial)
68    data_final = pd.to_datetime(data_final)
```

A função em seguida é a função de baixar os dados, ela coleta o código da estação e as datas do dia atual, 24 horas e 48 horas. Após isso, a função baixa o dado (com base nos tokens fornecidos) de cada estação e os organiza. O dado do PED chega com todas as informações em formato de apenas uma linha com ‘\n’ e ‘;’ separando as informações de cada linha e coluna. O comando ‘dados_filtrado’ organiza as informações de forma correta e as salva em um dataframe chamado ‘df’. É importante destacar que o comando ‘StringIO’ transforma o conteúdo salvado em uma informação, que futuramente vai poder ser

entendida e aplicada nos agrupamentos de precipitação. Após isso, todas as informações de todas as estações são agrupadas em outro dataframe chamado ‘dados_estacoes’.

```python
def baixar_dados_estacoes(codigo_estacao, data_inicial, data_final, sigla_estado):
    # Lista para armazenar os dados de todas as estações
    dados_estacoes = {}

    for codigo in codigo_estacao:
        # Lista para armazenar os dados de cada mês de uma estação
        dados_completos = []

        for ano_mes_dia in pd.date_range(data_inicial, data_final, freq='M'):
            ano_mes = ano_mes_dia.strftime('%Y%m')  # Formato '202401'

            # URL e parâmetros da requisição
            sws_url = 'http://sws.cemaden.gov.br/PED/rest/pcds/dados_pcd'
            params = dict(
                rede=11, uf=sigla_estado, inicio=ano_mes, fim=ano_mes, codigo=codigo
            )

            # Requisição dos dados
            r = requests.get(sws_url, params=params, headers={'token': token})
            dados = r.text

            # Remover a linha de comentário e converter para DataFrame
            linhas = dados.split("\n")
            dados_filtrados = "\n".join(linhas[1:])  # Remove a primeira linha (comentário)

            df = pd.read_csv(StringIO(dados_filtrados), sep=";")

            # Armazena os dados no acumulado
            dados_completos.append(df)

        # Combina os dados de todos os meses para a estação
        if dados_completos:
            dados_estacoes[codigo] = pd.concat(dados_completos)

    return dados_estacoes
```

Seguindo o código, para o acumulado da hora atual, do acumulado das 24 horas e das 48 horas, serem plotados no dashboard foi feita a função de mostrar gráficos. Ela é uma forma de plotar no site os acumulados feitos com cores diferentes para cada acumulado, que serão calculados à frente.

```python
# Função para exibir gráficos de precipitação
def mostrar_graficos(codigo_estacao):
    # Verificar se o código da estação existe no dicionário
    if codigo_estacao not in somas_por_estacao:
        st.error(f"Estação {codigo_estacao} não encontrada.")
        return

    # Obter os valores para a estação selecionada
    soma_dia_atual = somas_por_estacao[codigo_estacao]["dia_atual"]
    soma_24h = somas_por_estacao[codigo_estacao]["ultimas_24h"]
    soma_48h = somas_por_estacao[codigo_estacao]["ultimas_48h"]

    # Preparar os dados para o gráfico
    horas = ['Dia Atual', '24 Horas', '48 Horas']
    chuva_valores = [soma_dia_atual, soma_24h, soma_48h]

    # Criar o gráfico
    fig, ax = plt.subplots(figsize=(5, 3))
    ax.bar(horas, chuva_valores, color=['blue', 'orange', 'green'])
    ax.set_ylabel('Precipitação (mm)')
    ax.set_title(f'Precipitação para a Estação {codigo_estacao}')

    # Exibir o gráfico no Streamlit
    st.pyplot(fig)
```

Uma das bibliotecas mais importantes é a Leafmap, com ela é possível aplicar mapas interativos de forma parecida ao GoogleMaps. O comando que adiciona ele no dashboard é mostrado a seguir, onde a variável ‘m’ é atribuição e configuração do mapa, e o segundo comando é a centralização dele no site.

```python
m = leafmap.Map(center=[-21, -45],zoom_start = 8,draw_control=False, measure_control=False, fullscreen_control=False, attribution_control=True)

# Defina o layout da página como largo
st.set_page_config(layout="wide")
```

Agora os próximos passos é fazer os cálculos dos dataframes para cada estação. Realizando o acumulado atual, 24 horas e 48 horas. Inicialmente coletamos os dados baixados e é feito uma varredura para remover possíveis dados vazios. Em seguida em ’df’, selecionamos na coluna ‘sensor’ todos os dados diferentes de ‘intensidade_precipitacao’. Essa coluna ‘sensor’ apresenta duas variáveis, ‘chuva’ e intensidade_precipitacao’. A variável ‘chuva’ resulta no acúmulo de chuva que ocorreu no intervalo de 10 minutos, representando os valores reais de chuva. Já ‘intensidade_precipitacao’ representa a intensidade de precipitação no intervalo de 10 minutos, sendo inviável o acúmulo. Em seguida adicionamos um index ‘data_hora’ que será responsável por transformar a data e hora em uma informação que será possível de ser acumulada no tempo. E todo o processo é salvo em um dicionário chamado dado2.

```python
# Baixar dados da estação
dados1 = baixar_dados_estacoes(codigo_estacao, data_inicial, data_final, sigla_estado)

# Remover chave se o valor for vazio (DataFrame vazio)
for codigo in list(dados1.keys()):
    valor = dados1[codigo]

    if isinstance(valor, pd.DataFrame) and valor.empty:
        del dados1[codigo]  # Remove a chave se for um DataFrame vazio
dados2 = {}
for codigo in dados1.keys():
  df = dados1[codigo][dados1[codigo]['sensor'] != 'intensidade_precipitacao']
  df['datahora'] = pd.to_datetime(df['datahora'])
  df = df.set_index('datahora')
  dados2[codigo] = df
```

Em seguida criamos um dicionário ‘somas_por_estacao’ vazio que vai receber todos dados calculados a seguir. Em dado2 é feito um ‘looping’ onde cada estação antes salva vai passar pelo processo de acúmulo, porém antes do acumulo de cada tempo é necessário salvar em variáveis (‘inicio_dia_atual’; ‘inicio_24h’; ‘inicio_48h’) o delta tempo para ser feito o cálculo.

Seguindo o código, coletando os deltas e é feito o somatória para cada tempo, em cada estação de dados2. A forma de fazer a identificação do tempo para o somatório é utilizando o tempo ‘.loc’ que se acumula com base nos deltas. Em seguida, novamente tudo é salvo no dicionário agora chamado ‘somas_por_estacao’ com as colunas de soma atual, soma 24 horas e soma 48 horas para cada estação.

```python
# Criação do dicionário para armazenar os resultados
somas_por_estacao = {}

# Data/hora atual para referência
agora = datetime.now()

# Iterar sobre os dataframes em dados2
for codigo_estacao, df in dados2.items():
    # Garantir que o index esteja no formato datetime
    df.index = pd.to_datetime(df.index)

    # Filtrar os dados para o dia atual, últimas 24 horas e últimas 48 horas
    inicio_dia_atual = agora.replace(hour=0, minute=0, second=0, microsecond=0)
    inicio_24h = agora - timedelta(hours=24)
    inicio_48h = agora - timedelta(hours=48)

    soma_dia_atual = df.loc[df.index >= inicio_dia_atual, 'valor'].sum()
    soma_24h = df.loc[df.index >= inicio_24h, 'valor'].sum()
    soma_48h = df.loc[df.index >= inicio_48h, 'valor'].sum()

    # Armazenar os resultados em somas_por_estacao
    somas_por_estacao[codigo_estacao] = {
        "dia_atual": soma_dia_atual,
        "ultimas_24h": soma_24h,
        "ultimas_48h": soma_48h
    }
```

Com os dados atribuídos para cada estação é necessário plotar os resultados no mapa interativo para cada localização da estação. No mapa há quadrados verdes que

mostram a localização das estações e ao clicar em algum deles, este retornará os acúmulos de cada tempo feitos anteriormente em uma janela. O código a seguir faz esse processo, de coletar a localização de cada estação e identificá-las no mapa interativo.

```
# Adicionar marcadores das estações meteorológicas
for i, row in gdf_mg.iterrows():
    # Adicionar marcador com valor
    folium.RegularPolygonMarker(
        location=[row['latitude'], row['longitude']],
        color='black',
        opacity=1,
        weight=1,
        fillColor='green',
        fillOpacity=1,
        numberOfSides=4,
        rotation=45,
        radius=8,
        popup=f"{row['municipio']} (Código: {row['codEstacao']})"
    ).add_to(m)

m.add_gdf(
    mg_gdf,
    layer_name="Minas Gerais",
    style={"color": "black", "weight": 1, "fillOpacity": 0, "interactive": False},
    info_mode=None
)
```

Agora com a coleta de dados e somatória no tempo de forma automática, quando o usuário clicar em alguma das estações do mapa será disposto a ele o download dos dados da respectiva estação automática. Porém, antes é necessário fazer o filtro de coleta de dados com base na escolha do usuário, mudando o dia com base na classificação escolhida no dashboard. Dessa forma, todo o código abaixo faz essa seleção, onde o usuário delimita o intervalo de tempo em ‘tipo_busca’. Todo o código adiante é de seleção do tempo que é fornecido ao usuário para o download dos dados de cada estação.

```
st.sidebar.header("Filtros de Seleção")
modo_selecao = st.sidebar.radio("Selecionar Estação por:", ('Código'))

# Seleção da estação
if modo_selecao == 'Código':
    estacao_selecionada = st.sidebar.selectbox("Selecione a Estação", gdf_mg['codEstacao'].unique())
    # Certifique-se de que o código da estação é extraído corretamente
    codigo_estacao = gdf_mg[gdf_mg['codEstacao'] == estacao_selecionada]['codEstacao'].values[0]

# Adicionar um controle para "Recarregar Dados" quando a data for alterada
tipo_busca = st.sidebar.radio("Tipo de Busca:", ('Diária'))

if tipo_busca == 'Diária':
    data_inicial = st.sidebar.date_input("Data", value=data_inicial)
else:
    ano_selecionado = st.sidebar.selectbox("Selecione o Ano", range(2020, datetime.now().year + 1))
    mes_selecionado = st.sidebar.selectbox("Selecione o Mês", range(1, 13))
    data_inicial = datetime(ano_selecionado, mes_selecionado, 1)
    data_final = datetime(ano_selecionado, mes_selecionado + 1, 1) - timedelta(days=1) if mes_selecionado != 12 else datetime(ano_selecionado, 12, 31)

# Adicionar um controle de flag para verificar se os dados precisam ser recarregados
data_inicial_str = data_inicial.strftime('%Y%m%d')
data_final_str = data_final.strftime('%Y%m%d')
```

Ao final é feita a função de download dos dados a partir do intervalo de tempo atribuído pelo usuário. Dentro, ela identifica a estação clicada pelo usuário e o tempo selecionado e baixa. Quando não encontrado o dado, ela fornece a informação ‘Nenhum dado encontrado para o período selecionado’, evitando possíveis erros. Os termos no final desse código são funções específicas do Streamlit para o funcionamento e a plotagem correta do site.

```
    # Verificar se os dados já estão carregados
    if 'dados_baixados' not in st.session_state or st.session_state.data_inicial != data_inicial_str or st.session_state.data_final != data_final_str:
        # Se os dados não estão carregados ou a data foi alterada, atualize os dados
        st.session_state.dados_baixados = baixar_dados_estacoes(codigo_estacao, data_inicial, data_final, sigla_estado)
        st.session_state.data_inicial = data_inicial_str
        st.session_state.data_final = data_final_str

    # Exibir os dados baixados
    dados_baixados = st.session_state.dados_baixados

    if dados_baixados:
        st.subheader(f"Dados da Estação: {estacao_selecionada} (Código: {codigo_estacao})")
        st.write(dados_baixados)
    else:
        st.warning("Nenhum dado encontrado para o período selecionado.")

    if st.sidebar.button("Baixar Dados"):
        data_inicial_str = data_inicial.strftime('%Y%m%d')
        data_final_str = data_final.strftime('%Y%m%d')
        dados_baixados = dados['codEstacao']

        if not dados_estacao.empty:
            st.subheader(f"Dados da Estação: {estacao_selecionada} (Código: {codigo_estacao})")
            st.write(dados_baixados)
        else:
            st.warning("Nenhum dado encontrado para o período selecionado.")

    # Checkbox na barra lateral para alternar exibição do gráfico
    mostrar = st.sidebar.checkbox("Gráfico de Precipitação")

    # Exibir ou ocultar o gráfico conforme o estado do checkbox
    if mostrar:
        # Exibir o gráfico para a estação selecionada
        mostrar_graficos(codigo_estacao)
    # Mostrar o mapa em Streamlit
    m.to_streamlit(width=1300,height=775)

    st.write(somas_por_estacao)

    st.write(dados2)
```