Documentação da API Flask

1 Descrição Geral

Este código define uma API utilizando o framework Flask. A API permite a recuperação de dados de diferentes categorias (produção, processamento, comercialização, importação e exportação) no formato CSV, hospedados em um servidor remoto. Ele faz uso de bibliotecas como `requests`, `pandas` e `BeautifulSoup` para manipulação e processamento dos dados. Os dados são transformados em JSON e retornados via rotas RESTful.

2 Bibliotecas Utilizadas

- **Flask**: Usado para criar o servidor web e definir as rotas da API.
- **BeautifulSoup**: Importada, mas não utilizada no código.
- **pandas**: Utilizado para manipulação dos dados tabulares vindos do CSV.
- **requests**: Faz a requisição HTTP para obter os arquivos CSV remotos.
- **StringIO**: Usado para converter o conteúdo da resposta HTTP em um objeto semelhante a arquivo.
- **numpy**: Importada, mas não utilizada diretamente no código.

3 Estrutura do Código

3.1 Variáveis Globais

- `app`: Inicializa o aplicativo Flask.
- `port`: Define a porta onde o servidor Flask será executado.
- `url_base_csv`: Define a URL base de onde os arquivos CSV serão baixados.

3.2 Função Principal `get_data(page: str)`

Objetivo: Recuperar os dados CSV de uma URL específica, processá-los e retornar um JSON com o conteúdo.

Passos:

```
response = requests.get(url_base_csv + page + ".csv")
```

Listing 1: Exemplo de requisição HTTP

1. **Requisição HTTP**: Usa `requests.get()` para baixar o arquivo CSV com base no nome da página fornecida. O nome do arquivo é formado pela junção da URL base com o nome da página, terminando em `.csv`.

```
csv_data = StringIO(response.content.decode("utf-8"))
df = pd.read_csv(csv_data, sep=";")
df.fillna(0, inplace=True)
```

Listing 2: Exemplo de processamento de dados

2. **Processamento de Dados**: O conteúdo da resposta HTTP é lido como uma string, convertido para um formato de arquivo via `StringIO`, e então lido como um DataFrame utilizando `pandas`. Valores `NaN` são substituídos por 0 para evitar inconsistências.

```python
if 'control' in df.columns:
    df['control'] = df['control'].astype(str)
```

Listing 3: Tratamento especial para a coluna control

3. **Tratamento Especial para Coluna `control`**: - Caso a coluna `control` esteja presente, ela é convertida para o tipo string.

```python
mask_upper = df['control'].str.isupper()
indices = df[mask_upper].index.tolist()
indices.append(len(df))
```

Listing 4: Exemplo de máscara para maiúsculas

- Os valores de `control` são verificados para identificar se estão em maiúsculas. Esses valores são usados como divisores para segmentar o DataFrame em diferentes seções, com base nos índices das linhas onde há texto em maiúsculas.

```python
for i in range(len(indices) - 1):
    df_temp = df.iloc[indices[i]:indices[i + 1]]
    key = df_temp.iloc[0]['control']
    df_temp = df_temp.drop(columns=['control'])
    df_dict[key] = df_temp.to_dict(orient='records')
```

Listing 5: Segmentação do DataFrame

- O DataFrame é segmentado e organizado em um dicionário, onde cada chave representa o valor da primeira linha da seção identificada por `control`. As outras colunas são convertidas em um dicionário de registros.

4. **Retorno**: O DataFrame (ou seu dicionário segmentado) é convertido para JSON e retornado.

4 Tabela de Rotas da API

Rota	Verbo	Descrição	Estrutura de Retorno
/producaoCSV	GET	Retorna dados de produção de uvas.	Lista de dicionários com os registros CSV em formato JSON.
/processamentoCSV/tipo/{tipo}	GET	Retorna dados de processamento por tipo (Viniferas, Americanas, Mesa, Semclass).	Dicionário com seções baseadas em valores de `control`, cada uma contendo uma lista de registros.
/comercializacaoCSV	GET	Retorna dados de comercialização de uvas e derivados.	Lista de dicionários com os registros CSV em formato JSON.
/importacaoCSV/tipo/{tipo}	GET	Retorna dados de importação por tipo (Vinhos, Espumantes, Frescas, Passas, Suco).	Lista de dicionários com os registros CSV em formato JSON.
/exportacaoCSV/tipo/{tipo}	GET	Retorna dados de exportação por tipo (Vinho, Espumantes, Uva, Suco).	Lista de dicionários com os registros CSV em formato JSON.

5 Tratamento de Erros

- Em caso de falha na requisição HTTP, a função retorna uma mensagem de erro com o código de status apropriado.

```python
return jsonify({"error": f"Failed to retrieve data, status code: {response.
    status_code}"}), response.status_code

```

- Em caso de exceções no processo de requisição, uma mensagem de erro é retornada com o código de status 500.

```
except requests.RequestException as e:
    return jsonify({"error": str(e)}), 500


```

6 Como Rodar o Código

6.1 Sem Ambiente Virtual (venv)

Instale as dependências diretamente no ambiente global do Python.

Passos:

1. Certifique-se de ter o `pip` instalado.
2. No diretório do projeto, rode o seguinte comando para instalar as dependências:

```
pip install flask pandas requests

```

3. Execute o servidor com:

```
python app.py

```

6.2 Com Ambiente Virtual (venv)

Crie um ambiente virtual para isolar as dependências.

Passos:

1. Crie o ambiente virtual:

```
python -m venv venv

```

2. Ative o ambiente virtual:

```
# Windows
venv\Scripts\activate
# Linux/Mac
source venv/bin/activate

```

3. Instale as dependências:

```
pip install flask pandas requests

```

4. Execute o servidor:

```
python app.py

```

6.3 Parando o Ambiente Virtual

Para desativar o ambiente virtual, execute:

```
deactivate
```

7 Arquivo `requirements.txt`

O arquivo `requirements.txt` deve conter as dependências necessárias para rodar o código. Abaixo está um exemplo do que esse arquivo pode conter:

```
beautifulsoup4==4.12.3
blinker==1.8.2
certifi==2024.8.30
charset-normalizer==3.3.2
click==8.1.7
colorama==0.4.6
Flask==3.0.3
```

```
idna==3.8
itsdangerous==2.2.0
Jinja2==3.1.4
MarkupSafe==2.1.5
numpy==2.1.1
pandas==2.2.2
python-dateutil==2.9.0.post0
pytz==2024.1
requests==2.32.3
six==1.16.0
soupsieve==2.6
tzdata==2024.1
urllib3==2.2.2
Werkzeug==3.0.4
```