Documentação da API Flask e Testes

1 Descrição Geral

Este código define uma API utilizando o framework Flask para recuperar e processar dados CSV de produção, processamento, comercialização, importação e exportação de uvas e derivados. Ele inclui a autenticação via JWT e possui testes unitários para verificar o funcionamento das rotas.

2 Servidor: Estrutura do Código

2.1 Bibliotecas Utilizadas

- **Flask**: Usado para criar o servidor web e definir as rotas da API.
- **Flask-JWT-Extended**: Usado para adicionar autenticação JWT à API.
- **pandas**: Para manipulação dos dados CSV.
- **requests**: Para fazer as requisições HTTP e obter os arquivos CSV.
- **StringIO**: Para converter o conteúdo da resposta HTTP em um objeto semelhante a um arquivo.

2.2 Autenticação JWT

A API usa autenticação JWT para proteger as rotas. Antes de acessar qualquer rota de dados, o usuário precisa autenticar-se e obter um token JWT.

```
from flask_jwt_extended import JWTManager, jwt_required, create_access_token

# Inicializando o JWT Manager
app.config['JWT_SECRET_KEY'] = 'your_secret_key'
jwt = JWTManager(app)

@app.route('/login', methods=['POST'])
def login():
    if request.json.get("username") == "admin" and request.json.get("password") == "
    admin":
        access_token = create_access_token(identity="admin")
        return jsonify(access_token=access_token)
    return jsonify({"msg": "Bad username or password"}), 401
```

Listing 1: Exemplo de uso de JWT

2.3 Função Principal `get_data(page: str)`

Essa função recupera dados de uma URL específica, processa-os e retorna um JSON com o conteúdo, conforme descrito anteriormente.

2.4 Rotas Protegidas

As rotas da API são protegidas usando o decorador `jwt_required()`, que exige a presença de um token JWT válido.

```
@app.route('/producaoCSV')
@jwt_required()
def producao_csv():
    return get_data("Producao")
```

Listing 2: Exemplo de rota protegida

3 Tabela de Rotas da API

Rota	Verbo	Descrição	Estrutura de Retorno
/login	POST	Gera um token de autenticação JWT.	Retorna um token JWT.
/producaoCSV	GET	Retorna dados de produção.	JSON com dados de produção de uvas.
/processamentoCSV/tipo/{tipo}	GET	Retorna dados de processamento.	JSON com dados de processamento segmentados por tipo.
/comercializacaoCSV	GET	Retorna dados de comercialização.	JSON com dados de comercialização.
/importacaoCSV/tipo/{tipo}	GET	Retorna dados de importação.	JSON com dados de importação segmentados por tipo.
/exportacaoCSV/tipo/{tipo}	GET	Retorna dados de exportação.	JSON com dados de exportação segmentados por tipo.

4 Como Rodar o Código

4.1 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 -r requirements.txt
```

3. Execute o servidor com:

```
python app.py
```

4.2 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 -r requirements.txt
```

4. Execute o servidor:

```
python app.py
```

4.3 Parando o Ambiente Virtual

Para desativar o ambiente virtual, execute:

```
deactivate
```

5 Testes Unitários

Os testes utilizam a biblioteca `unittest` para verificar se as rotas estão funcionando corretamente.

5.1 Teste de Login

Testa se o login retorna um token JWT válido para credenciais corretas.

```
def test_login(self):
    tester = app.test_client(self)
    response = tester.post('/login', json={'username': 'admin', 'password': 'admin'})
    self.assertEqual(response.status_code, 200)
    self.assertIn('access_token', response.json)
```

Listing 3: Exemplo de teste de login

5.2 Teste de Rotas Protegidas

Verifica se as rotas protegidas retornam os dados corretos quando o token JWT é fornecido.

```
def test_producao_csv(self):
    tester = app.test_client(self)
    login_response = tester.post('/login', json={'username': 'admin', 'password': 'admin
    '})
    token = login_response.json['access_token']

    response = tester.get('/producaoCSV', headers={'Authorization': f'Bearer {token}'})
    self.assertEqual(response.status_code, 200)
```

Listing 4: Exemplo de teste de rota protegida

5.3 Execução dos Testes

Para rodar os testes, execute o seguinte comando:

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

```
pip install -r requirements.txt

```

3. Execute o servidor com:

```
python -m unittest test_app

```

6 Cenários de Uso

A API pode ser utilizada para prever a demanda futura, otimizar a produção e alavancar as exportações. Alimentando o modelo com dados históricos de comércio, produção e exportação, é possível prever os volumes de demanda em diferentes mercados com mais precisão..

6.1 Modelos de Machine Learning

Integrar a API com modelos de Machine Learning ajuda a fazer previsões sobre a demanda futura e otimizar a produção. É possível utilizar ferramentas como o AWS SageMaker e criar um pipeline eficiente. Técnicas como regressão linear, árvores de decisão ou modelos preditivos são boas opções para prever a demanda.

6.2 Benefícios

Previsão de Demanda: Os produtores podem ajustar a produção com base em previsões mais precisas de demanda. Otimização de Estoque: Melhor planejamento de estoques para diferentes períodos do ano. Planejamento de Exportações: Identificação de oportunidades de exportação, com base em tendências globais de consumo.