Documentação do Projeto: Integração com a API de Bancos e Análise de Dados

Autor: Gabrielly Simone Miranda da Silva

Data: 16/12/2024

1. Introdução

Objetivo:

O principal objetivo desse projeto é demonstrar como integrar a API BrasilAPI para coletar dados sobre bancos no Brasil, tratar essas informações e gerar tabelas úteis para análise. A ideia é explorar como é possível consumir dados de APIs públicas, realizar alguns tratamentos simples com Python e pandas, e apresentar resultados de forma organizada.

Público-alvo:

Código desenvolvido para apresentação do projeto final de conclusão do curso de Python da Coder House, turma 63730, professor Gabriel Rodrigues.

Saída:

O resultado do código será uma série de tabelas filtradas com as informações dos bancos, como:

- Todos os Bancos.
- Bancos com código ímpar.
- Bancos cujos nomes são menores que 10 caracteres.

Essas tabelas serão geradas como DataFrames e podem ser facilmente exportadas para CSV ou usadas diretamente para análise.

1. Importação das Bibliotecas

As bibliotecas necessárias para o funcionamento do código são importadas logo no início:

Importação de Bibliotecas

```python
# Importação de bibliotecas
import requests  # Para acessar a API
import pandas as pd  # Para manipular os dados
import sqlite3  # Para armazenar os dados em SQLite
```

[23] ✓ 0.0s

- **requests**: Utilizada para fazer a requisição à API e obter os dados.
- **pandas**: Usada para manipular e tratar os dados de forma eficiente.
- **sqlite3**: Usada para armazenar os dados extraídos em um banco de dados SQLite.

---

2. Acesso à API e Tratamento de Dados

O código acessa a API **BrasilAPI** para obter informações sobre os bancos no Brasil. O endpoint utilizado é:

Definição da URL da API e função para consultar dados

```python
# Definir a URL da API
url_api = "https://brasilapi.com.br/api/banks/v1"

# Função para acessar a API e retornar os dados
def acessar_api(url):
    try:
        response = requests.get(url)  # Fazendo a requisição GET
        response.raise_for_status()  # Verificando se houve erro na requisição
        return response.json()  # Retorna os dados no formato JSON
    except requests.exceptions.RequestException as e:
        print(f"Erro ao acessar a API: {e}")
        return None
```

24] ✓ 0.0s

Obter dados da API

```python
# Chamar a função para acessar a API
dados = acessar_api(url_api)

# Verificar se os dados foram obtidos com sucesso
if dados:
    print("Dados extraídos com sucesso!")
else:
    print("Falha ao obter dados da API.")
```

5] ✓ 0.0s

```
Dados extraídos com sucesso!
```

Se a requisição for bem-sucedida (código de status 200), os dados são extraídos em formato JSON. Caso contrário, é exibida uma mensagem de erro com o status da requisição.

Após a requisição bem-sucedida, os dados são carregados em um DataFrame pandas:

Criar o DataFrame e tratar os dados

```
# Se os dados foram obtidos, criamos o DataFrame
if dados:
    df_principal = pd.DataFrame(dados)
```

---

3. Tratamento dos Dados

O tratamento de dados inclui:

- **Remoção de duplicatas**: Para garantir que os dados não contenham registros duplicados.
- **Substituição de valores nulos**: Todos os valores nulos nas colunas são substituídos por 'Desconhecido'.

```
# Remover duplicatas e substituir valores nulos
df_principal = df_principal.drop_duplicates()
df_principal = df_principal.fillna('Desconhecido')
```

- **Padronização dos nomes das colunas**: Os nomes das colunas são ajustados para tornar o entendimento mais fácil.
- **Normalização dos nomes dos bancos**: Os nomes dos bancos são convertidos para letras maiúsculas para uniformizar a apresentação dos dados.

```
# Padronizar os nomes das colunas
df_principal.columns = ['ISPB', 'Nome_Banco', 'Codigo_Banco', 'Nome_Completo']

# Normalizar o texto nas colunas de nome
df_principal['Nome_Banco'] = df_principal['Nome_Banco'].str.upper()
df_principal['Nome_Completo'] = df_principal['Nome_Completo'].str.upper()

print("Dados tratados com sucesso!")
display(df_principal.head())  # Exibindo as primeiras linhas para conferirmos
```

---

4. Tabelas Filtradas

Tabela 1: Todos os Dados Tratados

A primeira tabela contém todos os dados tratados, incluindo as colunas normalizadas e as duplicatas removidas.

Tabela 1: Todos os Dados (Dados Brutos Tratados)

```
# Tabela 1: Todos os dados tratados
df_tabela1 = df_principal.copy()
print("Tabela 1: Todos os Dados Tratados")
display(df_tabela1.head())
```

✓ 0.0s

Tabela 1: Todos os Dados Tratados

	ISPB	Nome_Banco	Codigo_Banco	Nome_Completo
0	00000000	BCO DO BRASIL S.A.	1.0	BANCO DO BRASIL S.A.
1	00000208	BRB - BCO DE BRASILIA S.A.	70.0	BRB - BANCO DE BRASILIA S.A.
2	00038121	SELIC	Desconhecido	BANCO CENTRAL DO BRASIL - SELIC
3	00038166	BACEN	Desconhecido	BANCO CENTRAL DO BRASIL
4	00122327	SANTINVEST S.A. - CFI	539.0	SANTINVEST S.A. - CREDITO, FINANCIAMENTO E INV...

Tabela 2: Bancos com Código Ímpar

Nesta etapa, o código filtra os bancos com códigos ímpares. Para isso:

- A coluna Codigo_Banco é convertida para numérico, tratando possíveis erros na conversão.
- Linhas com valores nulos em Codigo_Banco são removidas.

Tabela 2: Bancos com Código Ímpar

```
# Convertendo 'Codigo_Banco' para numérico
df_principal['Codigo_Banco'] = pd.to_numeric(df_principal['Codigo_Banco'], errors='coerce')

# Remover linhas onde 'Codigo_Banco' seja nulo
df_principal = df_principal.dropna(subset=['Codigo_Banco'])

# Garantir que 'Codigo_Banco' seja inteiro
df_principal['Codigo_Banco'] = df_principal['Codigo_Banco'].astype(int)

# Filtrar bancos com código ímpar
df_tabela2 = df_principal[df_principal['Codigo_Banco'] % 2 != 0]
print("Tabela 2: Bancos com Código Ímpar")
display(df_tabela2.head())
```

✓ 0.0s

Tabela 2: Bancos com Código Ímpar

	ISPB	Nome_Banco	Codigo_Banco	Nome_Completo
0	00000000	BCO DO BRASIL S.A.	1	BANCO DO BRASIL S.A.
4	00122327	SANTINVEST S.A. - CFI	539	SANTINVEST S.A. - CREDITO, FINANCIAMENTO E INV...
8	00329598	ÍNDIGO INVESTIMENTOS DTVM LTDA.	407	ÍNDIGO INVESTIMENTOS DISTRIBUIDORA DE TÍTULOS ...
11	00416968	BANCO INTER	77	BANCO INTER S.A.
12	00460065	COLUNA S.A. DTVM	423	COLUNA S/A DISTRIBUIDORA DE TITULOS E VALORES ...

Tabela 3: Bancos com Nome Reduzido

A terceira tabela é composta pelos bancos que têm o nome do banco com menos de 10 caracteres. A filtragem é feita utilizando a função str.len() para medir o comprimento do nome:

Tabela 3: Bancos com Nome Reduzido

```
# Filtrar bancos com nomes menores que 10 caracteres
df_tabela3 = df_principal[df_principal['Nome_Banco'].str.len() < 10]
print("Tabela 3: Bancos com Nome Reduzido")
display(df_tabela3.head())
```

✓ 0.0s

Tabela 3: Bancos com Nome Reduzido

	ISPB	Nome_Banco	Codigo_Banco	Nome_Completo
5	00204963	CCR SEARA	430	COOPERATIVA DE CREDITO RURAL SEARA - CREDISEARA
52	03973814	SERVICOOP	190	SERVICOOP - COOPERATIVA DE CRÉDITO DOS SERVIDO...
147	16695922	ID CTVM	439	ID CORRETORA DE TÍTULOS E VALORES MOBILIÁRIOS ...
170	22896431	PICPAY	380	PICPAY INSTITUIÇÃO DE PAGAMENTO S.A.
177	27302181	CRED-UFES	427	COOPERATIVA DE CREDITO DOS SERVIDORES DA UNIVE...

---

5. Armazenamento dos Dados no Banco de Dados SQLite

O código conecta a um banco de dados SQLite e armazena as três tabelas filtradas nele:

Conectar ao banco SQLite e salvar as tabelas

```
# Conectar ao banco SQLite
conexao = sqlite3.connect("bancos_brasilapi.db")

# Salvar as tabelas no banco de dados
df_tabela1.to_sql("todos_bancos", conexao, if_exists="replace", index=False)
df_tabela2.to_sql("bancos_codigo_impar", conexao, if_exists="replace", index=False)
df_tabela3.to_sql("bancos_nome_reduzido", conexao, if_exists="replace", index=False)
```

Após o armazenamento, a conexão com o banco é fechada:

```
# Fechar conexão
conexao.close()
print("Tabelas armazenadas com sucesso no banco SQLite 'bancos_brasilapi.db'.")
```

---

6. Verificação de Bancos com Código Zero

Por fim, o código verifica se há bancos com código igual a zero na tabela de dados tratados e exibe um alerta se houver algum banco com esse código:

Verificar bancos com Codigo_Banco igual a zero

```
# Verificar bancos com Codigo_Banco igual a 0
bancos_codigo_zero = df_tabela1[df_tabela1['Codigo_Banco'] == 0]

if not bancos_codigo_zero.empty:
    print("⚠️ Alerta: Existem bancos com 'Codigo_Banco' igual a 0:")
    print(bancos_codigo_zero)
else:
    print("Nenhum banco com 'Codigo_Banco' igual a 0 encontrado.")
```

✓ 0.0s

```
Nenhum banco com 'Codigo_Banco' igual a 0 encontrado.
```

Resumo das Tabelas Criadas

1. **Tabela 1 - Todos os Dados Tratados**: Contém os dados brutos após tratamento (remoção de duplicatas, substituição de valores nulos, padronização de colunas e nomes).
2. **Tabela 2 - Bancos com Código Ímpar**: Contém apenas os bancos cujo código de banco é ímpar.
3. **Tabela 3 - Bancos com Nome Reduzido**: Contém os bancos com nomes de menos de 10 caracteres.

Conclusão

Este código fornece uma maneira eficiente de acessar dados de bancos, tratá-los e armazená-los de forma organizada em um banco de dados SQLite. Ele também oferece funcionalidades específicas de filtragem e verificação, como a identificação de bancos com códigos ímpares e nomes reduzidos, além de verificar a presença de bancos com código zero.