Alpha Vantage API
Importação e persistência de preços de ativos

Eduardo Machado Martins
www.eduardommartins.wordpress.com

ESTRUTURA DO PROJETO

A chave foi adquirida no site da Alpha Vantage.
https://www.alphavantage.co/

Documentação da biblioteca disponiubilizada pela Alpha Vantage
https://www.alphavantage.co/documentation/

Bibliotecas instaladas via linha de comando e impotadas no início do código.

```
$ pip3 install alpha_vantage
$ pip3 install sqlite3
$ pip3 install pandas
```

Sistema operacional utilizado: Linux (Ubuntu 20.04 LTS)
Editor de código: VS Code
GitHub: https://github.com/edmartins-br/ConsumingAlphaVantageAPI

1. Conexão com a Alpha Vantage API

- Para acessar a chave que permite consumir a API da Alpha Vantage, foi realizado o seguinte procedimento:

  A chave foi salva num arquivo .txt para não ficar exposta no código principal.

```
ALPHAVANTAGE_API_KEY = open('alphaVantageKey.txt').read()
```

  A função **TimeSeries** foi declarada dentro da variável “ts”, com o parâmetro output_size no formato “**pandas**” para que fosse possível converter os dados num **DataFrame**:

```
ts = TimeSeries(key = ALPHAVANTAGE_API_KEY, output_format='pandas')
```

2. Aquisição dos dados utilizando as funções .get_daily e .get_symbol_search.

- Aqui são declarados os atributos da classe para utilização nas funcionalidades da aplicação. A função **.get_Daily** traz os dados: Date, Open, High, Low, Close e Volume. Já a função **.get_symbol_search** retorna uma linha com 9 colunas com os dados do ativo como Symbol, Nome, Region, etc. O parâmetro de saída utilizado foi o “Compact”, que retorna os 100 últimos registros. Caso fosse usado “full”, retornaria os registros dos últimos 20 anos.

```
class Stock:
  def __init__(self, stock_name, stock_info):
    self.__stock_name = stock_name
    self.__stock_info = stock_info
    self.__data, self.__metaData = ts.get_daily(symbol = (f'{ stock_name }.SAO'), outputsize = 'compact')
    self.__symbol, self.__SymbolMetaData = ts.get_symbol_search(f'{ stock_name }.SAO')
```

3. Banco de dados SQLite

- Declaração da conexão com o nome do banco a ser criado. Após criação, foi utilizada a função **.to_sql** da biblioteca **pandas** para criação das tabelas e armazenamento dos dados recebidos. A instrução possui o parâmetro **“if_exists”** que determina que, caso já existam dados no banco, eles são atualizados, caso não existam, são inseridos de acordo com sua respectiva tabela.

```
self.conn = sqlite3.connect('stock.db')

self.__data.to_sql(f'{stock_name}', self.conn, if_exists='replace')
self.__symbol.to_sql(f'{stock_info}', self.conn, if_exists='replace')
#print(self.__symbol)
# print(type(self.__symbol))
```

4. Busca dos dados armazenados no SQLite

- Na função Busca, é solicitado o parâmetro **_days**, para que se possa calcular quantos dias para trás devem ser trazidos do banco, utilizando a função **datetime.timedelta**.

- Nesta seção do código são executadas as intruções responsáveis por selecionar as informações armazenadas no banco de dados, e converte-las para um **dataFrame** e em seguida printadas no console:

5. Plotagem de gráficos individuais para cada ativo.

```
query = pd.read_sql(f'SELECT * FROM { self.__stock_name } WHERE date >= "{ date_string }"', self.conn)
df = pd.DataFrame(query, columns=['date','4. close'])

querySymbol = pd.read_sql(f'SELECT * FROM { self.__stock_info }', self.conn)
dfs = pd.DataFrame(querySymbol, columns=['1. symbol', '2. name'])

print(f'DADOS CARREGADOS DE { self.__stock_name }:')
print('_'*50)

print('='*50)
print(dfs)

print('='*50)
print(df)
print('='*50)
```

6. Plotagem de gráficos individuais para cada ativo.

- Aqui são executadas as instruções utilizando a biblioteca **matplotlib** para realizar o plot indvidual dos valores de fechamento dos ativos. É realizada uam verificação para identificar qual o ativo buscado.

```
if (self.__stock_name == 'B3SA3' or self.__stock_name == 'PETR4'):

  plt.title('Daily Time Series for the {self.__stock_name} stock (close)')

  plt.plot(df['4. close'])

  plt.xlabel("YEAR")
  plt.ylabel("POINTS")

  plt.plot([], label = self.__stock_name)
  plt.legend()

  plt.show()

return df, dfs
```

7. Instanciando a classe e chamando a função Busca.

- Instanciando a classe Stock e chamando a função de busca posteriormente, passando o parâmetro _days, que neste caso é 7, referente aos valores da última semana.

```
def busca(self, _days):
  date = datetime.datetime.now() - datetime.timedelta(days=_days)
  date_string = date.strftime('%Y-%m-%d')
```

8. Resultados

- Para os últimos 7 dias a aplicação retorna os seguintes resultados:

```
CARREGANDO DADOS...

DADOS CARREGADOS DE B3SA3:

================================================
   1. symbol          2. name
0  B3SA3.SAO  B3 S.A. - Brasil
================================================
                  date  4. close
0   2020-12-10 00:00:00    174.63
1   2020-12-11 00:00:00    176.52
2   2020-12-14 00:00:00    174.84
3   2020-12-15 00:00:00    176.64
4   2020-12-16 00:00:00    181.65
..                  ...       ...
95  2021-05-05 00:00:00     51.50
96  2021-05-06 00:00:00     50.80
97  2021-05-07 00:00:00     53.34
98  2021-05-10 00:00:00     53.30
99  2021-05-11 00:00:00     52.85

[100 rows x 2 columns]
================================================
DADOS CARREGADOS DE PETR4:

================================================
   1. symbol                              2. name
0  PETR4.SAO  Petróleo Brasileiro S.A. - Petrobras
================================================
                  date  4. close
0   2020-12-10 00:00:00     27.82
1   2020-12-11 00:00:00     27.57
2   2020-12-14 00:00:00     27.62
3   2020-12-15 00:00:00     27.85
4   2020-12-16 00:00:00     28.19
..                  ...       ...
95  2021-05-05 00:00:00     23.83
96  2021-05-06 00:00:00     23.50
97  2021-05-07 00:00:00     24.38
98  2021-05-10 00:00:00     24.70
99  2021-05-11 00:00:00     25.15

[100 rows x 2 columns]
================================================
```