**data\_mt5.py**

Este arquivo contém funções para conectar-se ao MetaTrader 5 (MT5), obter dados históricos de mercado, calcular métricas e indicadores financeiros, e salvar os resultados em arquivos CSV. As principais funções incluem a coleta de dados, cálculo de retornos, adição de indicadores financeiros, e a exportação dos dados processados.

**Importações**

* **MetaTrader5** (mt5): Interface para interagir com o MetaTrader 5 e acessar dados financeiros.
* **Pandas** (pd): Utilizado para manipulação de dados em formato DataFrame.
* **Logging** (logging): Para registrar informações e erros durante a execução.
* **NumPy** (np): Biblioteca para cálculos numéricos.
* **os**: Utilizado para manipulação de arquivos e diretórios no sistema de arquivos.

**Funções no arquivo:**

**1. initialize\_mt5()**

python

Copiar

def initialize\_mt5():

"""Inicializa a conexão com o MetaTrader 5."""

* **Objetivo**: Tenta estabelecer uma conexão com o MetaTrader 5. Caso a conexão falhe, retorna **False** e registra um erro no log.
* **Uso**: Chamado no início de outras funções que requerem dados do MT5.

**2. get\_data(symbol, timeframe, num\_candles)**

python

Copiar

def get\_data(symbol, timeframe, num\_candles):

"""Obtém dados do MetaTrader 5 para o símbolo e timeframe fornecidos."""

* **Objetivo**: Obtém dados históricos de **candlesticks** para um símbolo financeiro específico no **timeframe** e quantidade de candles solicitados.
* **Processo**:
  + Verifica se o símbolo é válido.
  + Obtém os dados usando **mt5.copy\_rates\_from\_pos()**.
  + Converte os timestamps para o formato **datetime**.
* **Retorno**: Retorna um **DataFrame** contendo os dados históricos com colunas como time, open, close, high, low, e volume.

**3. add\_tick\_info(df, symbol)**

python

Copiar

def add\_tick\_info(df, symbol):

"""Adiciona informações de tick (bid, ask e last\_price) ao DataFrame."""

* **Objetivo**: Adiciona informações de **tick** (preços de **bid**, **ask**, e **last\_price**) para um símbolo financeiro.
* **Processo**:
  + Valida o parâmetro **symbol**.
  + Obtém os preços de **bid** e **ask** do MetaTrader 5.
  + Calcula o **last\_price** como a média entre **bid** e **ask**.
* **Retorno**: Retorna o DataFrame com as colunas **bid**, **ask**, e **last\_price** adicionadas.

**4. calculate\_market\_returns(symbol, timeframe, num\_candles)**

python

Copiar

def calculate\_market\_returns(symbol, timeframe, num\_candles):

"""Calcula os retornos logarítmicos do mercado."""

* **Objetivo**: Calcula os **retornos logarítmicos** do mercado com base nos preços de fechamento.
* **Processo**:
  + Usa **get\_data()** para obter os dados históricos do mercado.
  + Calcula o retorno logarítmico entre preços de fechamento consecutivos.
  + Remove valores **NaN** que surgem devido ao cálculo do primeiro retorno.
* **Retorno**: Retorna uma série de **retornos logarítmicos**.

**5. get\_last\_period\_data(symbol, timeframe, num\_candles, frame\_period)**

python

Copiar

def get\_last\_period\_data(symbol, timeframe, num\_candles, frame\_period):

"""Obtém os dados do último período e adiciona as métricas calculadas."""

* **Objetivo**: Obtém os dados do último período especificado e adiciona diversas métricas e indicadores financeiros.
* **Processo**:
  + Obtém os dados históricos com **get\_data()**.
  + Adiciona informações de **tick** com **add\_tick\_info()**.
  + Calcula os **retornos do mercado** com **calculate\_market\_returns()**.
  + Aplica a função **feature\_engineering\_pipeline()** para adicionar indicadores técnicos e métricas financeiras.
* **Retorno**: Retorna o DataFrame com as métricas calculadas.

**6. save\_csv(df, symbol, timeframe, base\_path)**

python

Copiar

def save\_csv(df, symbol, timeframe, base\_path):

"""Salva os dados calculados em um arquivo CSV dentro da pasta definida em config.py."""

* **Objetivo**: Salva o DataFrame com os dados calculados em um arquivo CSV na pasta **input\_data**.
* **Processo**:
  + Cria o diretório **input\_data** (se não existir) usando **os.makedirs()**.
  + Define o nome do arquivo baseado no **símbolo** e **timeframe**.
  + Salva o **DataFrame** como um arquivo CSV com codificação **UTF-8**.
  + Registra no log se o arquivo foi salvo com sucesso ou se ocorreu algum erro.

**7. fetch\_and\_process\_data(symbol, timeframe, num\_candles, frame\_period, base\_path)**

python

Copiar

def fetch\_and\_process\_data(symbol, timeframe, num\_candles, frame\_period, base\_path):

"""Função principal que obtém e processa os dados com base nos parâmetros fornecidos."""

* **Objetivo**: Função principal que integra o processo de conexão com o MT5, coleta de dados, cálculo de métricas e salvamento dos resultados em CSV.
* **Processo**:
  + Tenta inicializar a conexão com o **MetaTrader 5**.
  + Obtém os dados do **último período** com **get\_last\_period\_data()**.
  + Verifica se os dados foram obtidos com sucesso.
  + Salva os dados em um arquivo CSV com **save\_csv()**.
  + Registra mensagens de log para cada etapa (erro, sucesso).
  + Fecha a conexão com o MT5 independentemente de qualquer erro.