Dokumentation TradingBot

Das Projekt besteht darin, durch den Verlauf von Aktienpreise und mithilfe einer Strategie Gewinne zu erzielen. Zuerst werden die Aktienpreise im 1 Minütigen Abständen der letzten 7 Tage in eine Datenbank gepackt und anschließend wird in echtzeit jede Minute der aktuelle Preis von 15:30 - 22:00 zur Datenbank hinzugefügt. Außerhalb des zeitlichen Rahmens von 15:30 - 22:00 passiert nichts, da die Preise von Yahoo Finance abgefragt werden und nur in diesem Zeitraum die Preise aktualisieren. Kauf und Verkaufsignale sollen durch eine Simple Moving Average (SMA) Strategie generiert werden. Es gibt hier 2 Werte, einen "kleinen/short" SMA von 10-50 meist und einen "großen/long" SMA zwischen meist 100-251. Basierend auf dieser Zahl wird dann der Durchschnittliche Preis der letzten x Tage generiert. x ist am Ende dann jeweils der Wert zwischen 10-50 und 100-251. Jeder Aktienkurs hat dann am Ende einen short SMA und einen long SMA. Geht der short SMA über den long SMA wird gekauft und fällt der short SMA unter den long SMA wird verkauft. Um jetzt die optimale Kombination von short SMA und long SMA zu finden wird ein simpler Bruteforce algorithmus angewandt, der basierend auf den Aktienkursen der letzten 7 Tage ein optimales SMA Parameter Paar generiert. Das ergebnis, sowie der absolute return of invest und die Anzahl erforderlicher trades wird dann mit dem Aktienkürzel in einer Datenbank Tabelle gespeichert.

Installation der Packages und starten des Programmes

Einfach den Befehl pip install -r requirements.txt ausführen und alle nötigen packages werden installiert. Anschließend dann die Main.py Datei ausführen und es werden zuerst die besten Parameter für die SMA Strategie "gebacktested", was einen Moment dauert. Anschließend startet dann der minütliche "Test" ob gekauft oder verkauft werden soll. Bei Kauf- / Verkaufsignalen wird in der Konsole der aktuelle Preis mit dem jeweiligen Signal ausgegeben

Erklärung der Klassen und Funktionen

SmaStrategyBacktester-Klasse

Die SmaStrategyBacktester-Klasse enthält die Backtesting-Strategie auf der Basis vom Simple Moving Average (SMA).

- SmaStrategyBacktester(historical_data: pandas.DataFrame):
    - Diese Funktion initialisiert die SmaStrategyBacktester-Klasse mit einer pandas.DataFrame-Instanz, die die historischen Preisdaten des Aktiensymbols enthält, auf das die Backtesting-Strategie angewendet werden soll.
- get_optimal_sma_parameter(sma_s_range: Tuple[int, int, int], sma_l_range: Tuple[int, int, int]) -> Tuple[int, int]:
    - Diese Funktion wird verwendet, um die optimalen SMA-Parameter für die Backtesting-Strategie zu berechnen. Die Funktion akzeptiert als Argumente "sma_s_range" und "sma_l_range", die jeweils einen Tupel darstellen, das den Startwert, Endwert und Schrittweite des SMA-Parameters angibt. Die Funktion gibt ein Tupel zurück, das die optimalen SMA-Parameter enthält.

- run_sma_strategy(sma_parameters: Tuple[int, int]):
    - Diese Funktion wird verwendet, um die Backtesting-Strategie auf der Basis von Simple Moving Average (SMA) mit den gegebenen SMA-Parametern auszuführen. Das Argument "sma_parameters" ist ein Tupel, das die SMA-Parameter enthält.
- get_absolute_return_of_sma_strategy() -> float:
    - Diese Funktion gibt die absolute Rendite der Backtesting-Strategie zurück.
- get_amount_of_trades() -> int:
    - Diese Funktion gibt die Anzahl der Trades zurück, die von der Backtesting-Strategie durchgeführt wurden.

GenerateHistoricalDataframeForStock-Klasse

Die Klasse GenerateHistoricalDataframeForStock dient dazu, ein DataFrame mit historischen Preisen für eine bestimmte Aktie zu erstellen. Dazu werden die numpy und yfinance Bibliotheken importiert.

- GenerateHistoricalDataframeForStock(self, symbol: str):
    - Der Konstruktor der Klasse nimmt ein Aktienkürzel als Eingabe, erstellt ein DataFrame mit historischen Preisen für das gegebene Symbol und fügt dem DataFrame eine returns-Spalte hinzu, die die logarithmischen Renditen des Aktienkurses berechnet.
- get_dataframe(self) -> pandas.DataFrame:
    - Gibt das DataFrame mit historischen Preisen zurück.

UpdateStockPricesDB.py

Helferdatei mit Funktionen, welche aus der Datenbank mit den historischen und aktuellen Preisen für eine Aktie ein Pandas Dataframe erstellt und ein Pandas Dataframe auch in die Datenbank speichern kann

ProvideOptimalSMAParameter.py

Helferdatei mit Funktionen, die eine SQLite-Datenbank mit dem Namen "trading_bot.db" verwalten. Die Datenbank speichert optimale Parameter für einen Simple Moving Average (SMA).

- store_optimal_sma_parameter_to_db(symbol: str, optimal_parameter: Tuple[int, int], absolute_return: float, amount_of_trades: int):
    - speichert die optimalen Parameter für einen bestimmten Aktiensymbol in der Datenbank. Wenn das Symbol bereits vorhanden ist, werden die aktualisierten Parameter gespeichert.
- get_optimal_sma_parameter_from_db(symbol: str) -> Tuple[int, int]:
    - gibt die optimalen SMA-Parameter für ein bestimmtes Symbol aus der Datenbank zurück.
- set_last_signal(symbol: str, new_last_signal: str):
    - aktualisiert das letzte Signal für ein bestimmtes Symbol in der Datenbank.
- get_last_signal(symbol: str) -> str:

    - gibt das letzte Signal für ein bestimmtes Symbol aus der Datenbank zurück.

- clear_table():
    - löscht alle Einträge in der Datenbanktabelle.
- drop_table():
    - löscht die gesamte Tabelle aus der Datenbank.