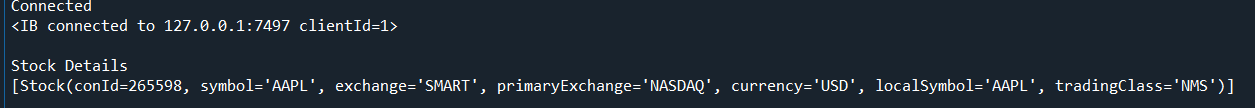
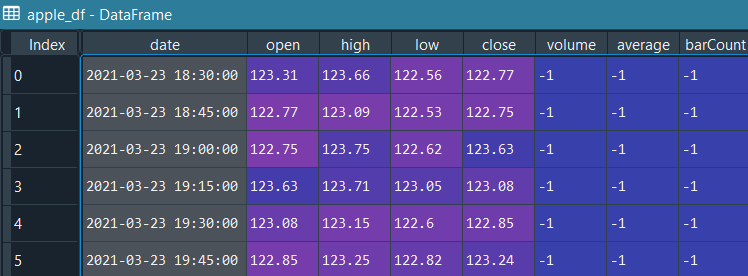
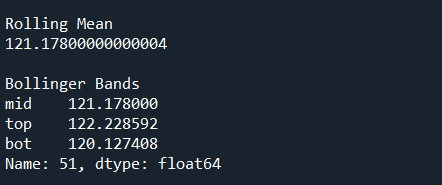
**ib\_insync**

* **Overview:**
  + Provides both a blocking and an asynchronous interface to the IB API, using asyncio networking and event loop.
  + The IB class offers **direct access to the current state**, such as orders, executions, positions, tickers etc. This state is automatically kept in sync with the TWS/IBG application.
  + Blocking: Will block until complete and return the result. The current state will be kept updated while the request is ongoing.
  + Asynchronous: All methods that have the “Async” postfix. Implemented as coroutines or methods that return a Future
  + **The One Rule:** The one rule when working with the IB class is that **user code may not block for too long**.
    - While some of the request methods are blocking, the framework will keep spinning in the background and handle all messages received from TWS/IBG
    - It is important to not block the framework from doing its work. If, for example, the user code spends much time in a calculation, or uses time.sleep() with a long delay, the framework will stop spinning, messages accumulate, and things may go wrong
    - IB request methods are okay to use and do not count towards the user operation time, no matter how long the request takes to finish.
    - An example of too long is if the timestamp of tick data is to remain accurate within a millisecond**, then the user code must not spend longer than a millisecond**. If, on the other extreme, there is very little incoming data and there is no desire for accurate timestamps, **then the user code can block for hours.**
    - If a user operation takes a long time then it can be farmed out to a different process. Alternatively the operation can be made such that it periodically calls IB.sleep(0)
    - For introducing a delay, never use time.sleep() but use [sleep()](https://ib-insync.readthedocs.io/api.html#ib_insync.ib.IB.sleep) instead.
* **Getting Started:**
  + **Installation:** Use pip install ib\_insync, whilst ensuring that the client TWS/IB is installed and the API is enabled
  + **Imports:** We can import the library using from ib\_insync import \*
  + **Connection:** We can instantiate an IB class and use the .connect() method on it**.** The method is a blocking method

**Text, logo

Description automatically generated**

* **Retrieving Current Price of a Stock:**
  + Whenever data is required for an asset, start by creating a contract, we can create a contract using the Stock helper class, we have passed three main positional arguments
    - ticker: string, the symbol for the company
    - exchange: string, the exchange it trades on
    - currency: string, the currency the stock trades in
  + We can use qualiftyContracts(Stock\_object) to obtain additional information about the stock, such as its primary exchange, conId and tradingClass. The following image shows an example for the Apple stock and some information related to it
* **Retrieving Historical Data of a Stock:**
  + We can retrieve historical data using the reqHistoricalData function to get data, we specify the following parameters in this case:
    - contract: same Stock object that we created earlier
    - end time: string, left as an empty string to get most recent candle
    - barSizeSetting: string, the size of the candle, we can go as low as 1 second and as high as 1 month
    - durationStr: string, how far we want to go back for
    - whatToShow: string, we can specify that we want to see the BID, ASK, MIDPOINT, TRADES
    - useRTH: boolean, stands for Regular Trading Hours
    - Return Value: BarDataList
  + The following image shows historical data for the Apple stock, the returned object can be converted into Dataframe, it shows the 15 minutes candles for the bid for the Apple stock, whilst informing the user of the opening, closing, high and low bid price of the stock.
* **Calculating indicators for a Stock:**
  + Since we can convert the returned object from reqHistoricalData to a Dataframe, we can very easily filter and apply operations on the data
  + The following image shows that we can calculate the rolling mean on a rolling window of 20 data points, the last result is shown
  + Alternatively we can set Bollinger bands using the btalib on the Dataframe. The following image shows the Bollinger band for the last data point in our Dataframe



* **Creating a Bracket Order:**
  + To create an order, we need to create a contract first, then an order and then we need to send a request to the IB API
  + Text

    Description automatically generatedThere are four type of orders, orders can be a MarketOrder, LimitOrder, StopOrder or StopLimtOrder
  + In the following image, we are creating three orders:
    - a LimitOrder, for one share in the Apple stock which says to buy the stock if the price is $123.10,
    - a LimitOrder, to sell the share to buy the stock if the price hit $125.50,
    - a StopOrder, to sell the share if the price drops to $122.50
    - We are using placeOrder to submit a request to the IB API. The following image show the TWS

A screenshot of a computer

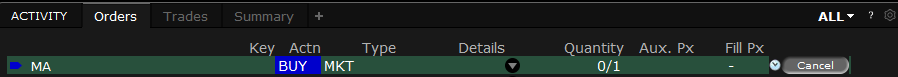
Description automatically generated

* + Similarly, we can create a Market Order as well and we check the status of the resulting trade.
  + We can also create a call back function which will update the trade status to filled once the order is executed
  + The following image shows that the order is Pending Submit since it was placed out of regular trading hours, and since the order has not been filled or executed, that property is shown as None

Text

Description automatically generated

* **Creating Orders based on a Price Condition:**
  + We can create Market Order based on a Price Condition. In the following example, we intend to buy a stock for MasterCard when the stock for Visa hits a certain price
  + We start by creating a contract for Visa, next we create the Price Condition which has the following parameters:
    - price: this is the price Visa needs to hit to trigger our buy
    - conId: this is get automatically populated when we apply qualifyContracts on Visa
    - exch: the exchange on which the stock trades
  + The following image shows TWS after running the code, the clock means that there is a condition



* **Creating Orders based on a Percentage Changes:**
  + The logic in the example is flawed, the code does not enter any of the conditions that are set
  + In the following example we want to create a buy order when a certain stock moves more than 5% in the last 5 minutes.
  + In this case we want to send an order to buy a stock in Mastercard, when Visa moves more than 5% in the last 5 minutes
  + We can start by iterating over incoming data and saving it to a Dataframe, we are going to set the time and date as the index and the last price as the row item