Advanced Live Websocket Crypto Data Streams in Python

Shashank Vemuri Jul 7, 2022 8 min read Trading API Python Crypto

An integral part of any market trading strategy is the constant stream of live price data. With the alpaca-trade-api Python package, you can implement these data streams into your algorithms within a couple of minutes!

In this article, we'll go through a few different ways to set up a websocket for live streaming crypto data, and then finish off with a live trading strategy based on cross

Alpaca

Basic Websocket Usage

In this example, we can use the Stream object from the alpaca-trade-api to set up streaming real-time Bitcoin (BTCUSD) price or trade data.

In the main() function, we can use the subscribe_crypto_quotes or subscribe_crypto_trades methods of the Stream object to specify which data we want to receive. As the first parameter for these methods, we can place the function print_quote or print_trade to specify what to do with the data once it is received. For the second parameter, we can enter the symbol.

Next, we can set up an asynchronous function that only executes once the data is received and run the stream!

```
from alpaca_trade_api.stream import Stream

ALPACA_API_KEY = '********************'
ALPACA_SECRET_KEY = '****************************************'

async def print_quote(q):
    print('quote', q)

async def print_trade(t):
    print('trade', t)

def main():
    stream = Stream(ALPACA_API_KEY,ALPACA_SECRET_KEY, raw_data=True)
    stream.subscribe_crypto_quotes(print_quote, 'BTCUSD')
    stream.subscribe_crypto_trades(print_trade, 'BTCUSD')

    @stream.on_bar('BTCUSD')
    async def _(bar):
        print('bar', bar)

    stream.run()
```

Alpaca

Websocket Reconnection

In case of any websocket disconnection, we can wrap the following basic code we used in the previous example with a try/except block so that the websocket can reconnect.

For trading systems that rely on real-time data, it's integral to use a data stream that can run continuously regardless of errors. In this example, the try/except block prevents the program from ending if the websocket connection runs into any errors. Instead, the error will be printed out and after a set period of time, the program will try to re-establish connection to the websocket.

```
import asyncio
import logging
import time
from alpaca_trade_api.stream import Stream
from alpaca_trade_api.common import URL

logging.basicConfig(format='%(asctime)s %(message)s', level=logging.INF

ALPACA_API_KEY = '********************'
ALPACA_SECRET_KEY = '****************************************'

def run_connection(conn):
    try:
        conn.run()
    except KeyboardInterrupt:
        print("Interrupted execution by user")
        asyncio.get_event_loop().run_until_complete(conn.stop_ws())
        exit(0)
    except Exception as e:
        print(f'Exception from websocket connection: {e}')
    finally:
        print("Trying to re-establish connection")
        time.sleep(3)
        run_connection(conn)
```

```
if __name__ == '__main__':
    conn = Stream(ALPACA_API_KEY,
                  ALPACA_SECRET_KEY,
                  base_url=URL('https://paper-api.alpaca.markets'))

    conn.subscribe_crypto_bars(print_quote, 'BTCUSD')

    run_connection(conn)
```

Pause/Resume Data Stream

In order to stop and start the websocket connection at will, we can use the ThreadPoolExecutor method from the concurrent package. This can allow us to shut down the websocket subscription and turn it on again.

```
import logging
import time
from concurrent.futures import ThreadPoolExecutor
from alpaca_trade_api.stream import Stream
from alpaca_trade_api.common import URL

ALPACA_API_KEY = '********************'
ALPACA_SECRET_KEY = '****************************************'

async def print_quote(q):
    print('quote', q)

def consumer_thread():
    global conn
    conn = Stream(ALPACA_API_KEY,
                  ALPACA_SECRET_KEY,
                  base_url=URL('https://paper-api.alpaca.markets'))

    conn.subscribe_crypto_quotes(print_quote, 'BTCUSD')
    conn.run()
```

Alpaca

```
logging.basicConfig(format='%(asctime)s  %(levelname)s %(message)s'
                    level=logging.INFO)

pool = ThreadPoolExecutor(1)

while 1:
    try:
        pool.submit(consumer_thread)
        time.sleep(20)
        conn.stop_ws()
        time.sleep(20)
    except KeyboardInterrupt:
        print("Interrupted execution by user")
        conn.stop_ws()
        exit(0)
    except Exception as e:
        print("You got an exception: {} during execution. continue
              "execution.".format(e))
        # let the execution continue
        pass
```

Dynamic Data Subscription

In addition to subscribing to data for only one cryptocurrency at a time, we can set up the websocket to change the subscription on demand.

In this case, we can set up a dictionary with all of the the symbols and their corresponding functions we wish to execute. Then, we can run a for loop to iterate through the dictionary items and run the websocket connection for that specified symbol.

```
import logging
import threading
import time
from alpaca_trade_api.stream import Stream
from alpaca_trade_api.common import URL
```

```
async def print_quote(q):
    print('quote', q)

PREVIOUS = None

def consumer_thread():
    global conn
    conn = Stream(ALPACA_API_KEY,
                  ALPACA_SECRET_KEY,
                  base_url=URL('https://paper-api.alpaca.markets'))

    conn.subscribe_crypto_quotes(print_quote, 'BTCUSD')
    global PREVIOUS
    PREVIOUS = "BTCUSD"
    conn.run()

if __name__ == '__main__':
    logging.basicConfig(format='%(asctime)s %(levelname)s %(message)s',
                        level=logging.INFO)
    threading.Thread(target=consumer_thread).start()
    time.sleep(5)  # give the initial connection time to be established
    subscriptions = {"ETHUSD": print_quote,
                     "BTCUSD": print_quote,
                     "DOGEUSD": print_quote,
                     }

    while 1:
        for ticker, handler in subscriptions.items():
            conn.subscribe_crypto_quotes(PREVIOUS)
            conn.subscribe_crypto_quotes(handler, ticker)
            PREVIOUS = ticker
            time.sleep(20)
```

Live Crypto Trading Bot Example: Cross-Sectional Momentum

Typical cross-sectional momentum strategies involve ranking securities based on their recent returns and using that data to go long the best performing assets and go short the worst performing assets, hoping that the prevailing trend in both cases will continue. Since Alpaca does not support shorting cryptocurrency, this tutorial will cover just going long the best performing crypto from the last x period of days.

Import Dependencies

First, we'll need to import all of the required dependencies we'll be using for the crypto bot including pandas for dataframe manipulation, datetime to specify the start and end dates for historical data, and alpaca-trade-api for market data and paper trading account access. If you have not yet used alpaca-trade-api before, you can pip install the package in your terminal.

```
# Import Dependencies
import numpy as np
import pandas as pd
import alpaca_trade_api as tradeapi
import datetime as dt
```

Setup and Define Variables

The next step is to define the variables we'll be needing throughout the program. For the Alpaca API and Secret keys, you can access those within your free paper trading account. After setting up the api, we can use the close_all_positions() method to clear the portfolio of any holdings.

In order to retrieve historical data for the preferred time frame, we can specify the start and end dates using datetime. In this case, we'll be using the past 60 days of historical data.

```
# API Credentials
ALPACA_API_KEY = '********************'
ALPACA_SECRET_KEY = '****************************************'
```

```
# Date Variables
start_date = dt.date.today() - dt.timedelta(days = 60)
end_date = dt.date.today()
```

Create Function to Check Account Positions

Now, we can create a function to check whether the trading account currently holds any of the cryptocurrency that we pass in as the parameter. If it does, we can return the quantity of that cryptocurrency that is being held. Otherwise, then the function will just return 0.

This is important because in the next function which handles the buying and selling, we can focus on buying only if there is currently none of the same crypto in the account.

```
# Check Whether Account Currently Holds Symbol
def check_positions(symbol):
    positions = api.list_positions()
    for p in positions:
        if p.symbol == symbol:
            return float(p.qty)
    return 0
```

Create Cross Sectional Momentum Function

Finally, we can create a function to retrieve the historical data, rank the cryptocurrencies based on their recent returns, and execute the buy/sell orders. The function takes in an input of live bar data which can be used for the current close price of each asset.

In order to calculate the momentum signal, first we can create a dataframe of historical price data for our specified dates. Next, we can apply a function to this dataframe to convert price data into daily percentage returns. Using these returns, we can rank the best performing crypto over the past 7 days and give a buy signal to that one crypto.

We can check if our paper trading account currently holds any of that particular cryptocurrency. If not, it can use the remaining non-marginable buying power to buy it.

Alpaca

Wrapping the entire function in a try/except block ensures that the program will not break due to errors, and will simply print out the error message. Since this is a trading bot and is intended to run throughout market hours, it's best if the program is continuously running.

```
# Cross Sectional Momentum Bot Function
def cross_sectional_momentum(bar):
    try:
        # Get the Latest Data
        dataframe = pd.DataFrame()
        symbols = ['BTCUSD','ETHUSD','DOGEUSD','SHIBUSD','MATICUSD','AL
        for symbol in symbols:
            data = api.get_crypto_bars(symbol, tradeapi.TimeFrame(1, tr
            data = pd.DataFrame(data).rename(columns={"close": str(symb
            dataframe = pd.concat([dataframe,data], axis=1, sort=False)

        returns_data = dataframe.apply(func = lambda x: x.shift(-1)/x -

        # Calculate Momentum Dataframe
        momentum_df = returns_data.apply(func = lambda x: x.shift(1)/x.
        momentum_df = momentum_df.rank(axis = 1)
        for col in momentum_df.columns:
            momentum_df[col] = np.where(momentum_df[col] > 8, 1, 0)

        # Get Symbol with Highest Momentum
        momentum_df['Buy'] = momentum_df.astype(bool).dot(momentum_df.c
        buy_symbol = momentum_df['Buy'].iloc[-1]
        old_symbol = momentum_df['Buy'].iloc[-2]

        # Account Details
        current_position = check_positions(symbol=buy_symbol)
        old_position = check_positions(symbol=old_symbol)

        # No Current Positions
        if current_position == 0 and old_position == 0:
            cash_balance = api.get_account().non_marginable_buying_powe
            api.submit_order(buy_symbol, notional=cash_balance, side='b
            message = f'Symbol: {buy_symbol} | Side: Buy | Notional: {c
```

Alpaca

```
        if current_position == 0 and old_position == 1:
            api.close_position(old_position)
            message = f'Symbol: {old_symbol} | Side: Sell'
            print(message)

            cash_balance = api.get_account().non_marginable_buying_powe
            api.submit_order(buy_symbol, notional=cash_balance, side='b
            message = f'Symbol: {buy_symbol} | Side: Buy | Notional: {c
            print(message)

        print("-"*20)

    except Exception as e:
        print (e)
```

Set Up Alpaca Live Crypto Data

The last step of building the Python bot is to start streaming live market data for all the cryptocurrencies from Alpaca. Fortunately, Alpaca makes this process extremely easy.

First, we have to create an instance of the data streaming API by calling the Stream method in which we pass the API keys. We can also specify that we want raw data only from the FTX exchange. Then, we can create an asynchronous function to receive the live bar data and within this function, we can call the previous cross sectional momentum bot function. Lastly, we can subscribe to the daily bars of each of the 9 cryptos and then start the streaming of data. That's it!

```
# Create instance of Alpaca data streaming API
alpaca_stream = tradeapi.Stream(ALPACA_API_KEY, ALPACA_SECRET_KEY, raw_

# Create handler for receiving live bar data
async def on_crypto_bar(bar):
    print(bar)
    cross_sectional_momentum(bar)
```

Alpaca

```
# Start streaming of data
alpaca_stream.run()
```

Conclusion

In this tutorial, we started from setting up a simple websocket for Bitcoin live data to a fully fledged algorithmic trading bot that simultaneously utilizes the live data of 9 cryptocurrencies. With the examples shown in this article, you can hopefully tackle whatever needs you may have for live data use with websockets!

bots with the Alpaca Crypto API!

Please note that this article is for informational purposes only. The example above is for illustrative purposes only. Actual crypto prices may vary depending on the market price at that particular time. Alpaca Crypto LLC does not recommend any specific cryptocurrencies.

Cryptocurrency is highly speculative in nature, involves a high degree of risks, such as volatile market price swings, market manipulation, flash crashes, and cybersecurity risks. Cryptocurrency is not regulated or is lightly regulated in most countries. Cryptocurrency trading can lead to large, immediate and permanent loss of financial value. You should have appropriate knowledge and experience before engaging in cryptocurrency trading. For additional information please click here.

Cryptocurrency services are made available by Alpaca Crypto LLC ("Alpaca Crypto"), a FinCEN registered money services business (NMLS # 2160858), and a wholly-owned subsidiary of AlpacaDB, Inc. Alpaca Crypto is not a member of SIPC or FINRA. Cryptocurrencies are not stocks and your cryptocurrency investments are not protected by either FDIC or SIPC. Please see the Disclosure Library for more information.

This is not an offer, solicitation of an offer, or advice to buy or sell cryptocurrencies, or open a cryptocurrency account in any jurisdiction where Alpaca Crypto is not registered or licensed, as applicable.

Alpaca

Related Tags: Python Crypto

Related Articles:

Triangular Arbitrage with Alpaca's Coin Pair Trading (with Python Examples)

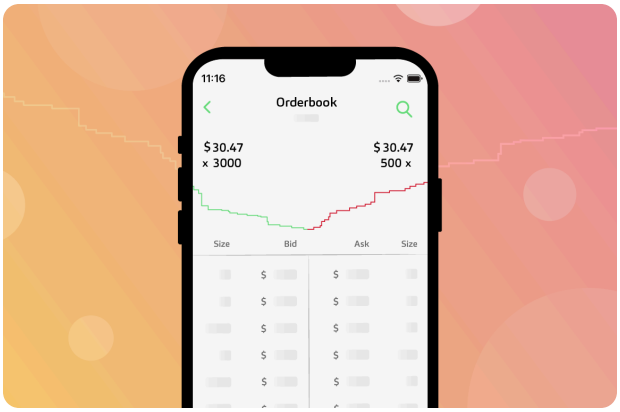
Jul 25, 2022
8 min read

Trading API

Alpaca

Jun 8, 2022

Trading API



Visualizing Orderbook Data with Alpaca Crypto API

May 26, 2022

7 min read

Trading API

Explore all of our Trading API content

Developers

Trading API Docs

Broker API Docs

Market Data API Docs

OAuth Docs

Alpaca

Learn
SDKs & Libraries
GitHub
API Status
About Us
Story
Blog
Contact
We're hiring

Disclosures
Disclosure Library
Form CRS
Form CRS Responses to Conversation Starters
Privacy Policy
Cookie Policy
Terms & Conditions

Securities brokerage services are provided by Alpaca Securities LLC ("Alpaca Securities"), member FINRA/SIPC, a wholly-owned subsidiary of AlpacaDB, Inc. Technology and services are offered by AlpacaDB, Inc.

Options trading is not suitable for all investors due to its inherent high risk, which can potentially result in significant losses. Please read Characteristics and Risks of Standardized Options before investing in options.

Cryptocurrency services are made available by Alpaca Crypto LLC ("Alpaca Crypto"), a FinCEN registered money services business (NMLS # 2160858), and a wholly-owned subsidiary of AlpacaDB, Inc. Alpaca Crypto is not a member of SIPC or FINRA. Cryptocurrencies are not stocks and your cryptocurrency investments are not protected by either FDIC or SIPC.

Cryptocurrency is highly speculative in nature, involves a high degree of risks, such as volatile market price swings, market manipulation, flash crashes, and cybersecurity risks. Cryptocurrency regulations are continuously evolving, and it is your responsibility to understand and abide by them. Cryptocurrency trading can lead to large, immediate and permanent loss of financial value. You should have appropriate knowledge and experience before engaging in cryptocurrency trading. For additional information, please click here.