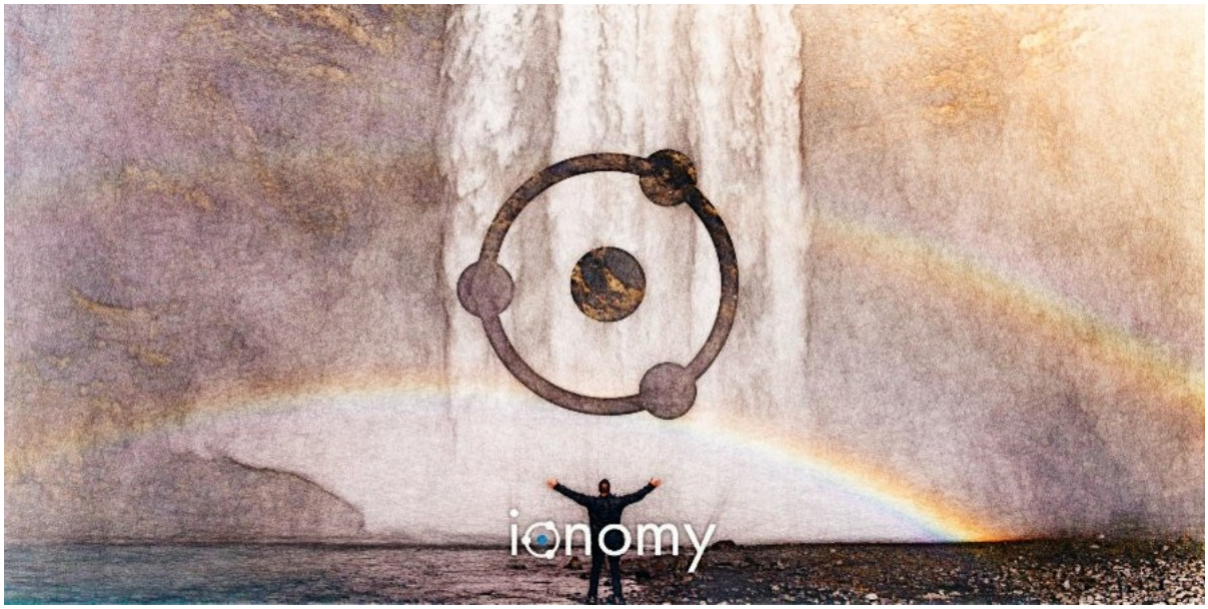
ionomy-python

Apr 24, 2021

Contents


ionomy

CHAPTER 1

Quick Start

1.1 Quick Start

1.1.1 Playground

Start up a jupyter lab instance with all the tutorial notebooks.

```
docker run -p 8888:8888 ionomy-playground:latest
```

Copy and Paste the url with token from terminal output into your browser.

Open the tutorial notebook of interest or create your own to experiment with!

1.1.2 Install

```
pip install ionomy-python
```

1.1.3 Imports and Common Params

The primary classes for use are IonPanda and BitTA.

There are lower level classes if needed: Ionomy, BitTrex, and BitPanda

Ionomy is a raw API wrapper, no extras.

The same with BitTrex, which is a raw api wrapper.

BitPanda is the bittrex equivalent of IonPanda,

but without the TA methods.

```
from Ionomy import IonPanda, BitTA

# Common Params
MARKET = 'btc-hive'
CURRENCY = 'hive'
BASE = 'btc'
TIME = 'day'
```

1.1.4 Instantiate

You must provide your Ionomy or BitTrex API key/secret, respectively.

IMPORTANT: I opted for the package user to manually load and update ohlcv data

This allows for control of the number of API call, since there are limits (See Exchanges Websites).

```
ionpd = IonPanda(IONOMY_KEY, IONOMY_SECRET)
bta = BitTA(BITTREX_KEY, BITTREX_SECRET)

# IMPORTANT - call this method to load/update ohlcv data
bta.update(CURRENCY, BASE, TIME)
```

1.1.5 Common Methods

The "tutorials" provides examples of each method and their returns.

The "modules" provides the detailed code documentation.

Here, I will show one method from each classed being used by the highest order classes.

```
# returns a regular dictionary from the raw json
# I saw no benefit from having a single row dataframe returned
market_summary = ionpd.market_summary(MARKET)

# returns a pandas dataframe
order_book_df = ionpd.order_book(MARKET)

# returns a regular dictionary from the raw json
market_summary = bta.market_summary(MARKET)

# returns a pandas dataframe
order_book_df = bta.order_book(MARKET)

# TA method are returned as DataFrames or Series
rsi_series = rsi_series = bta.rsi(length=1, drift=1, offset=0)
```

The most common TA methods are implemented and available Due to python limitations, such as floating point arthimetic, etc, results will/may differ from the standard talib package and other TA implementations on the same data. This difference should be within a very small margin of error from a c++ implementation.

CHAPTER 2

General

2.1 Installation

The minimal working python version is 3.7.x

2.1.1 Install with pip

```
pip install -U ionomy-python
```

or:

```
python -m pip install ionomy-python
```

2.1.2 Manual installation

You can install beem from this repository if you want the latest but possibly non-compiling version:

```
git clone https://github.com/Distortedlogic/ionomy-python.git
cd ionomy-python
```

then

```
python setup.py build
python setup.py install
```

or:

```
pip install -e .
```

2.2 Tutorials

2.2.1 BitTrex

Imports and Params

[10]:
```
from Ionomy import BitTrex
from decouple import config

MARKET = 'btc-hive'
CURRENCY = 'hive'
BASE = 'btc'
TIME = 'day'
BTC_QUANTITY = 0.001
HIVE_QUANTITY = 100
ADDRESS = 'memehub'
PAYMENTID_OPTIONAL = None
LOW_RATE=0.00003
HIGH_RATE=0.00004
TIMEINFORCE='GTC'
```

Instantiation

[2]:
```
bt = BitTrex(config('TREX_KEY'), config('TREX_SECRET'))
```

Public Endpoint Methods

Markets

[3]:
```
markets = bt.markets()
markets[0]
```

[3]:
```
{'MarketCurrency': 'LTC',
 'BaseCurrency': 'BTC',
 'MarketCurrencyLong': 'Litecoin',
 'BaseCurrencyLong': 'Bitcoin',
 'MinTradeSize': 0.03297187,
 'MarketName': 'BTC-LTC',
 'IsActive': True,
 'IsRestricted': False,
 'Created': '2014-02-13T00:00:00',
 'Notice': None,
 'IsSponsored': None,
 'LogoUrl': 'https://bittrexblobstorage.blob.core.windows.net/public/6defbc41-582d-
↪47a6-bb2e-d0fa88663524.png'}
```

Currencies

[4]:
```
currencies = bt.currencies()
currencies[0]
```

[4]:
```
{'Currency': 'BTC',
 'CurrencyLong': 'Bitcoin',
 'MinConfirmation': 2,
 'TxFee': 0.0005,
 'IsActive': True,
 'IsRestricted': False,
 'CoinType': 'BITCOIN',
 'BaseAddress': '1N52wHoVR79PMDishab2XmRHsbekCdGquK',
 'Notice': None}
```

Ticker

[5]:
```
bt.ticker(MARKET)
```

[5]:
```
{'Bid': 3.313e-05, 'Ask': 3.331e-05, 'Last': 3.345e-05}
```

Market Summaries

[7]:
```
market_summaries = bt.market_summaries()
market_summaries[0]
```

[7]:
```
{'MarketName': 'BTC-STPT',
 'High': 1.1e-06,
 'Low': 1.06e-06,
 'Volume': 89896.60128034,
 'Last': 1.1e-06,
 'BaseVolume': 0.09687313,
 'TimeStamp': '2020-05-16T16:52:42.353',
 'Bid': 1.06e-06,
 'Ask': 1.09e-06,
 'OpenBuyOrders': 42,
 'OpenSellOrders': 273,
 'PrevDay': 1.06e-06,
 'Created': '2019-06-11T18:34:44.627'}
```

Market Summary

[9]:
```
bt.market_summary(MARKET)
```

[9]:
```
{'MarketName': 'BTC-HIVE',
 'High': 3.49e-05,
 'Low': 3.28e-05,
 'Volume': 251042.78009306,
 'Last': 3.381e-05,
 'BaseVolume': 8.52802204,
 'TimeStamp': '2020-05-15T18:05:54.107',
 'Bid': 3.382e-05,
 'Ask': 3.409e-05,
 'OpenBuyOrders': 388,
 'OpenSellOrders': 853,
 'PrevDay': 3.472e-05,
 'Created': '2020-03-21T20:13:46.243'}
```

Market History

[11]:
```
market_history = bt.market_history(MARKET)
market_history[0]
```

[11]:
```
{'Id': 65281835,
 'TimeStamp': '2020-05-15T18:02:15.01',
 'Quantity': 16.80099545,
 'Price': 3.381e-05,
 'Total': 0.0005680416561645,
 'FillType': 'FILL',
 'OrderType': 'BUY',
 'Uuid': '822aacf4-da88-4be8-bb2b-6a723671acf9'}
```

Order Book

[14]:
```
order_book = bt.order_book(MARKET)
order_book["buy"][0]
```

[14]:
```
{'Quantity': 1013.53349185, 'Rate': 3.382e-05}
```

OHLCV

[16]:
```
ohlcv = bt.ohlcv(CURRENCY, BASE, TIME)
ohlcv[0]
```

[16]:
```
{'time': 1586908800,
 'close': 1.637e-05,
 'high': 1.7e-05,
 'low': 1.577e-05,
 'open': 1.633e-05,
 'volumefrom': 270819.04,
 'volumeto': 4.473,
 'conversionType': 'force_direct',
 'conversionSymbol': ''}
```

Market Endpoint Methods

Limit Buy/Sell Orders

[16]:
```
# buy_order_uuid = bt.buy_limit(MARKET, HIVE_QUANTITY, LOW_RATE, TIMEINFORCE)
sell_order_uuid = bt.sell_limit(MARKET, HIVE_QUANTITY, HIGH_RATE, TIMEINFORCE)
```

Cancel Order

[20]:
```
order_uuid = bt.cancel(sell_order_uuid)
order_uuid
```

Order Status

[19]:
```
order = bt.get_order(sell_order_uuid)
order
```

[19]:
```
{'AccountId': None,
 'OrderUuid': 'ffd0af51-6195-442b-aab6-320851bc72cc',
 'Exchange': 'BTC-HIVE',
 'Type': 'LIMIT_SELL',
 'Quantity': 100.0,
 'QuantityRemaining': 100.0,
 'Limit': 4e-05,
 'Reserved': None,
 'ReserveRemaining': None,
 'CommissionReserved': None,
 'CommissionReserveRemaining': None,
 'CommissionPaid': 0.0,
 'Price': 0.0,
 'PricePerUnit': None,
 'Opened': '2020-05-17T20:53:56.25',
 'Closed': None,
 'IsOpen': True,
 'Sentinel': None,
 'CancelInitiated': False,
 'ImmediateOrCancel': False,
 'IsConditional': False,
 'Condition': 'NONE',
 'ConditionTarget': None}
```

Open Orders

[17]:
```
open_orders = bt.open_orders(MARKET)
open_orders[0]
```

[17]:
```
{'Uuid': None,
 'OrderUuid': 'ffd0af51-6195-442b-aab6-320851bc72cc',
 'Exchange': 'BTC-HIVE',
 'OrderType': 'LIMIT_SELL',
 'Quantity': 100.0,
 'QuantityRemaining': 100.0,
 'Limit': 4e-05,
 'CommissionPaid': 0.0,
 'Price': 0.0,
 'PricePerUnit': None,
 'Opened': '2020-05-17T20:53:56.25',
 'Closed': None,
 'CancelInitiated': False,
 'ImmediateOrCancel': False,
 'IsConditional': False,
 'Condition': 'NONE',
 'ConditionTarget': None}
```

Account Endpoint Methods

Balances

```
[18]: balances = bt.balances()
      balances[0]
```

```
[18]: {'Currency': 'BTC',
       'Balance': 1e-08,
       'Available': 1e-08,
       'Pending': 0.0,
       'CryptoAddress': None}
```

Balance

```
[19]: bt.balance(CURRENCY)
```

```
[19]: {'Currency': 'HIVE',
       'Balance': 13728.03479181,
       'Available': 13728.03479181,
       'Pending': 0.0,
       'CryptoAddress': '30c05de7684c4bf1817'}
```

Order History

```
[22]: order_history = bt.order_history()
      order_history[0]
```

```
[22]: {'OrderUuid': '74b7ccde-f6f5-4083-8d0e-e56c080015a1',
       'Exchange': 'BTC-HIVE',
       'TimeStamp': '2020-05-14T21:11:14.44',
       'OrderType': 'LIMIT_BUY',
       'Limit': 3.41e-05,
       'Quantity': 1128.49345297,
       'QuantityRemaining': 0.0,
       'Commission': 7.696e-05,
       'Price': 0.03848162,
       'PricePerUnit': 3.41e-05,
       'IsConditional': False,
       'Condition': '',
       'ConditionTarget': 0.0,
       'ImmediateOrCancel': False,
       'Closed': '2020-05-14T21:11:14.44'}
```

Deposit History

```
[23]: deposit_history = bt.deposit_history(CURRENCY)
      deposit_history[0]
```

```
[23]: {'Id': 102691733,
       'Amount': 7194.41,
       'Currency': 'HIVE',
       'Confirmations': 59,
       'LastUpdated': '2020-05-14T21:06:48.5',
```

(continues on next page)

(continued from previous page)

```
 'TxId': 'da01a679e0c7d7afd6de6e3753ca2beb7dea0caf',
 'CryptoAddress': '30c05de7684c4bf1817'}
```

Deposit Address

[24]:
```
bt.deposit_address(CURRENCY)
```

[24]:
```
{'Currency': 'HIVE', 'Address': '30c05de7684c4bf1817'}
```

Withdrawal History

[28]:
```
withdrawal_history = bt.withdrawal_history(CURRENCY)
withdrawal_history[0]
```

[28]:
```
{'PaymentUuid': '78943ce0-b6d4-4fcb-991a-7516e4323e82',
 'Currency': 'HIVE',
 'Amount': 13728.02479181,
 'Address': 'memehub',
 'Opened': '2020-05-15T18:13:44.513',
 'Authorized': False,
 'PendingPayment': False,
 'TxCost': 0.01,
 'TxId': None,
 'Canceled': True,
 'InvalidAddress': False}
```

Withdraw

[33]:
```
bt.withdraw(CURRENCY, QUANTITY, ADDRESS, PAYMENTID_OPTIONAL)
```

[33]:
```
{'uuid': 'cee4a123-5656-4938-a777-66a05df1b500'}
```

[ ]:
```

```

2.2.2 BitPanda

Imports and Params

[1]:
```
from Ionomy import BitPanda
from decouple import config

MARKET = 'btc-hive'
CURRENCY = 'hive'
BASE = 'btc'
TIME = 'day'
```

Instantiation

[3]:
```
bp = BitPanda(config('TREX_KEY'), config('TREX_SECRET'))
```

Public Endpoint Methods

Markets

[5]:
```
markets_pd = bp.markets()
markets_pd.head()
```

[5]:
```
  MarketCurrency BaseCurrency MarketCurrencyLong BaseCurrencyLong  \
0            LTC          BTC           Litecoin          Bitcoin
1           DOGE          BTC           Dogecoin          Bitcoin
2            VTC          BTC           Vertcoin          Bitcoin
3            PPC          BTC           Peercoin          Bitcoin
4            FTC          BTC        Feathercoin          Bitcoin

   MinTradeSize MarketName  IsActive  IsRestricted    Created Notice  \
0      0.032972    BTC-LTC      True         False 2014-02-13   None
1   1000.000000   BTC-DOGE      True         False 2014-02-13   None
2      8.830802    BTC-VTC      True         False 2014-02-13   None
3      6.025548    BTC-PPC      True         False 2014-02-13   None
4    213.675214    BTC-FTC      True         False 2014-02-13   None

  IsSponsored                                            LogoUrl
0        None  https://bittrexblobstorage.blob.core.windows.n...
1        None  https://bittrexblobstorage.blob.core.windows.n...
2        None  https://bittrexblobstorage.blob.core.windows.n...
3        None  https://bittrexblobstorage.blob.core.windows.n...
4        None  https://bittrexblobstorage.blob.core.windows.n...
```

Currencies

[6]:
```
currencies_pd = bp.currencies()
currencies_pd.head()
```

[6]:
```
  Currency CurrencyLong  MinConfirmation   TxFee  IsActive  IsRestricted  \
0      BTC      Bitcoin                2  0.0005      True         False
1      LTC     Litecoin                6  0.0100      True         False
2     DOGE     Dogecoin                6  2.0000      True         False
3      VTC     Vertcoin              600  0.0200      True         False
4      PPC     Peercoin               12  0.0200      True         False

    CoinType                          BaseAddress Notice
0    BITCOIN  1N52wHoVR79PMDishab2XmRHsbekCdGquK   None
1  BITCOIN16  LhyLNfBkoKshT7R8Pce6vkB9T2cP2o84hx   None
2    BITCOIN  D9GqmkGCpgtnXP7xMD78v9xfqeDkqBZBMT   None
3  BITCOIN16  VfukW89WKT9h3YjHZdSAAuGNVGELY31wyj   None
4  BITCOIN16                                None   None
```

Ticker

[7]:
```
bp.ticker(MARKET)
```

[7]:
```
{'Bid': 3.381e-05, 'Ask': 3.409e-05, 'Last': 3.38e-05}
```

Market Summaries

[9]:
```
market_summaries_pd = bp.market_summaries()
market_summaries_pd.head()
```

[9]:

	MarketName	High	Low	Volume	Last	BaseVolume	TimeStamp	Bid	Ask	OpenBuyOrders	OpenSellOrders	PrevDay	Created
0	BTC-STPT	1.090000e-06	1.060000e-06	8.975255e+04	1.090000e-06	0.096687	2020-05-15 18:58:54.720	1.070000e-06	1.100000e-06	40.0	271.0	1.060000e-06	2019-06-11 18:34:44.627
1	ETH-WAXP	1.639700e-04	1.523000e-04	8.819855e+04	1.550100e-04	13.731805	2020-05-15 18:58:54.720	1.540300e-04	1.597800e-04	109.0	109.0	1.569600e-04	2018-02-15 01:09:15.167
2	ETH-FX	2.740000e-04	2.519900e-04	4.506822e+04	2.630000e-04	11.563228	2020-05-15 18:58:54.720	2.610000e-04	2.680000e-04	72.0	135.0	2.590000e-04	2019-04-23 17:00:38.527
3	BTC-PLA	5.000000e-08	4.000000e-08	1.844084e+06	4.000000e-08	0.074589	2020-05-15 18:58:54.720	3.000000e-08	4.000000e-08	45.0	137.0	5.000000e-08	2019-03-28 17:12:11.213
4	EUR-USDT	9.330000e-01	9.230000e-01	2.398502e+04	9.270000e-01	22295.859801	2020-05-15 18:58:54.720	9.230000e-01	9.290000e-01	73.0	72.0	9.300000e-01	2020-03-30 06:19:27.620

Market Summary

[12]:
```
bp.market_summary(MARKET)
```

[12]:
```
{'MarketName': 'BTC-HIVE',
 'High': 3.453e-05,
 'Low': 3.28e-05,
 'Volume': 275969.44525378,
 'Last': 3.365e-05,
 'BaseVolume': 9.35729375,
 'TimeStamp': '2020-05-15T18:59:34.75',
 'Bid': 3.37e-05,
 'Ask': 3.406e-05,
 'OpenBuyOrders': 389,
 'OpenSellOrders': 858,
 'PrevDay': 3.4e-05,
 'Created': '2020-03-21T20:13:46.243'}
```

Order Book

[13]:
```
order_book_pd = bt.order_book(MARKET)
order_book_pd.head()
```

[13]:
```
      Quantity      Rate type
0   833.815866  0.000034  bid
1  1700.000000  0.000034  bid
2    59.127338  0.000034  bid
3  1589.495316  0.000034  bid
4   557.397472  0.000034  bid
```

Market History

[15]:
```
market_history_pd = bp.market_history(MARKET)
market_history_pd.head()
```

[15]:
```
         Id                TimeStamp      Quantity     Price     Total  \
0  65288419 2020-05-15 18:59:18.930  28737.484222  0.000034  0.967016
1  65288418 2020-05-15 18:59:18.930     84.785096  0.000034  0.002854
2  65288417 2020-05-15 18:59:18.930    464.468900  0.000034  0.015639
3  65288416 2020-05-15 18:59:18.930   1700.000000  0.000034  0.057375
4  65288415 2020-05-15 18:59:18.930     35.000000  0.000034  0.001181

  FillType OrderType                                  Uuid
0     FILL      SELL  5bb886f8-7f19-478c-97cf-9f1127f3a698
1     FILL      SELL  17bf5e26-8aa6-4eab-aca0-a87a84e601e0
2     FILL      SELL  4fd0d932-0f83-46f3-b34c-f55864c19199
3     FILL      SELL  6d30c744-537a-4b71-a93a-7194c004f932
4     FILL      SELL  a9a528d8-4d5d-4488-9314-4343a0dca7f8
```

Market Endpoint Methods

Limit Buy/Sell Order

[ ]:
```
order_uuid = bp.buy_limit(MARKET, QUANTITY, RATE, TIMEINFORCE)
order_uuid = bp.sell_limit(MARKET, QUANTITY, RATE, TIMEINFORCE)
```

Cancel Order

[ ]:
```
order_uuid = bp.cancel(UUID)
```

Order Status

[ ]:
```
order = bp.get_order(UUID)
```

Account Endpoint Methods

Balances

[16]:
```
balances_pd = bp.balances()
balances_pd.head()
```

[16]:
```
  Currency       Balance     Available  Pending        CryptoAddress
0      BTC  1.000000e-08  1.000000e-08      0.0                 None
1   BTXCRD  8.489922e+00  8.489922e+00      0.0                 None
2     HIVE  1.372603e+04  1.372603e+04      0.0  30c05de7684c4bf1817
3    STEEM  0.000000e+00  0.000000e+00      0.0  acc190939a9248d981f
```

Balance

[17]:
```
bp.balance(CURRENCY)
```

[17]:
```
{'Currency': 'HIVE',
 'Balance': 13726.03479181,
 'Available': 13726.03479181,
 'Pending': 0.0,
 'CryptoAddress': '30c05de7684c4bf1817'}
```

Order History

[18]:
```
order_history_pd = bp.order_history()
order_history_pd.head()
```

[18]:
```
                              OrderUuid   Exchange                TimeStamp  \
0  74b7ccde-f6f5-4083-8d0e-e56c080015a1   BTC-HIVE 2020-05-14 21:11:14.440
1  b1be09e1-c0b5-40fd-bae1-9ef061521483   BTC-HIVE 2020-05-14 21:09:07.890
2  aef02e01-641f-4a0a-9223-05a846e47e4c  BTC-STEEM 2020-05-14 21:06:01.770
3  aba84609-e842-4681-90b1-42fb24501174   BTC-HIVE 2020-05-13 14:27:35.190
4  bf5b235d-ea76-4bd7-bf37-eebaaaea05dd  BTC-STEEM 2020-05-13 12:55:44.540

     OrderType     Limit     Quantity  QuantityRemaining  Commission  \
0    LIMIT_BUY  0.000034  1128.493453           0.000000    0.000077
1    LIMIT_BUY  0.000034  2828.824311        1128.824311    0.000116
2  MARKET_SELL       NaN  5439.624000           0.000000    0.000193
3    LIMIT_BUY  0.000035   485.000000           0.000000    0.000034
4  MARKET_SELL       NaN   912.726000           0.000000    0.000034

      Price  PricePerUnit  IsConditional Condition  ConditionTarget  \
0  0.038482      0.000034          False                      0.0
1  0.057953      0.000034          False                      0.0
2  0.096666      0.000018          False                      0.0
3  0.016975      0.000035          False                      0.0
4  0.016958      0.000019          False                      0.0

   ImmediateOrCancel                  Closed
0              False 2020-05-14 21:11:14.440
1              False 2020-05-14 21:09:34.990
2               True 2020-05-14 21:06:01.770
```

(continues on next page)

(continued from previous page)

```
3              False 2020-05-13 21:59:27.830
4               True 2020-05-13 12:55:44.540
```

Deposit History

[19]:
```
deposit_history_pd = bp.deposit_history(CURRENCY)
deposit_history_pd.head()
```

[19]:
```
          Id   Amount Currency  Confirmations             LastUpdated  \
0  102691733  7194.41     HIVE             59 2020-05-14 21:06:48.500

                                       TxId        CryptoAddress
0  da01a679e0c7d7afd6de6e3753ca2beb7dea0caf  30c05de7684c4bf1817
```

Deposit History

[20]:
```
bp.deposit_address(CURRENCY)
```

[20]:
```
{'Currency': 'HIVE', 'Address': '30c05de7684c4bf1817'}
```

Withdrawal History

[21]:
```
withdrawal_history_pd = bp.withdrawal_history(CURRENCY)
withdrawal_history_pd.head()
```

[21]:
```
                           PaymentUuid Currency         Amount  Address  \
0  cee4a123-5656-4938-a777-66a05df1b500     HIVE       0.990000  memehub
1  35aca318-89b1-4aa5-9b1a-2ca702b8268f     HIVE       0.990000  memehub
2  78943ce0-b6d4-4fcb-991a-7516e4323e82     HIVE   13728.024792  memehub

                    Opened  Authorized  PendingPayment  TxCost  \
0 2020-05-15 18:18:00.133        True           False    0.01
1 2020-05-15 18:17:31.920        True           False    0.01
2 2020-05-15 18:13:44.513       False           False    0.01

                                       TxId  Canceled  InvalidAddress
0  bd6a480971ab8fdf28bf46375551df9149b241dd     False           False
1  182ce6ce3c7e3b615c2558d046838dd904a85b7e     False           False
2                                      None      True           False
```

Withdraw

[ ]:
```
uuid = bt.withdraw(CURRENCY, QUANTITY, ADDRESS, PAYMENTID_OPTIONAL)
```

2.2.3 BitTA

Imports and Params

[1]:
```
from Ionomy import BitTA
from decouple import config

MARKET = 'btc-hive'
CURRENCY = 'hive'
BASE = 'btc'
TIME = 'day'
```

Instantiation

[2]:
```
bta = BitTA(config('TREX_KEY'), config('TREX_SECRET'))
```

Update the class ohlcv dataframe with the latest data

[3]:
```
bta.update(CURRENCY, BASE, TIME)
```

Momentum

Awesome Oscilator (AO)

[4]:
```
ao_series = bta.ao(fast=5, slow=34, offset=0)
ao_series.tail()
```

[4]:
```
26   -0.000005
27   -0.000005
28   -0.000006
29   -0.000006
30   -0.000007
Name: AO_5_34, dtype: float64
```

Absolute Price Oscillator (APO)

[6]:
```
apo_series = bta.apo(fast=12, slow=26, offset=0)
apo_series.tail()
```

[6]:
```
26   -0.000003
27   -0.000003
28   -0.000003
29   -0.000003
30   -0.000003
Name: APO_12_26, dtype: float64
```

Balance of Power (BOP)

[7]:
```
bop_series = bta.bop(offset=0)
bop_series.tail()
```

[7]:
```
26   -0.802281
27   -0.332075
28   -0.229630
29   -0.594203
30   -0.583333
Name: BOP, dtype: float64
```

Commodity Channel Index (CCI)

[8]:
```
cci_series = bta.cci(length=20, c=0.015, offset=0)
cci_series.tail()
```

[8]:
```
26   -76.387008
27   -77.379598
28   -75.407177
29   -85.992973
30   -97.848760
Name: CCI_20_0.015, dtype: float64
```

Center of Gravity (CG)

[9]:
```
cg_series = bta.cg(length=20, offset=0)
cg_series.tail()
```

[9]:
```
26   -12.041409
27   -12.148211
28   -12.138013
29   -11.565258
30   -11.332511
Name: CG_20, dtype: float64
```

Chande Momentum Oscillator (CMO)

[11]:
```
cmo_series = bta.cmo(length=14, drift=1, offset=0)
cmo_series.tail()
```

[11]:
```
26   -39.251391
27   -38.172458
28   -24.536601
29   -17.513736
30   -67.250355
Name: CMO_14, dtype: float64
```

Coppock Curve (COPC)

[12]:
```
coppock_series = bta.coppock(length=10, fast=11, slow=14, offset=0)
coppock_series.tail()
```

[12]:
```
26   -38.771187
27   -50.955977
```

(continues on next page)

(continued from previous page)

```
28   -55.510796
29   -55.224571
30   -54.643435
Name: COPC_11_14_10, dtype: float64
```

Fisher Transform (FISHT)

[13]:
```
fisher_series = bta.fisher(length=5, offset=0)
fisher_series.tail()
```

[13]:
```
26   -4.306868
27   -4.709653
28   -6.155028
29   -6.877715
30   -7.239059
Name: FISHERT_5, dtype: float64
```

Know Sure Thing' (KST)

[14]:
```
# Need more ohlcv data to get results
default_kwargs = {
    "roc1": 10,
    "roc2": 15,
    "roc3": 20,
    "roc4": 30,
    "sma1": 10,
    "sma1": 10,
    "sma1": 10,
    "sma1": 15,
    "signal": 9,
    "drift": 1,
    "offset": 0
}
kst_series = bta.kst(**default_kwargs)
kst_series.tail()
```

[14]:
```
    KST_10_15_20_30_15_10_10_15  KSTS_9
26                          NaN     NaN
27                          NaN     NaN
28                          NaN     NaN
29                          NaN     NaN
30                          NaN     NaN
```

Moving Average Convergence Divergence (MACD)

[15]:
```
macd_series = bta.macd(fast=12, slow=26, signal=9, offset=0)
macd_series.tail()
```

[15]:
```
    MACD_12_26_9  MACDH_12_26_9  MACDS_12_26_9
26     -0.000003  -1.645118e-07      -0.000002
27     -0.000003  -2.936861e-07      -0.000002
```

(continues on next page)

(continued from previous page)

```
28     -0.000003  -3.568946e-07      -0.000003
29     -0.000003  -4.071771e-07      -0.000003
30     -0.000003  -4.292012e-07      -0.000003
```

Momentum (MOM)

[4]:
```
mom_series = bta.mom(length=12, offset=0)
mom_series.tail()
```

[4]:
```
26   -0.000007
27   -0.000004
28   -0.000013
29   -0.000015
30   -0.000012
Name: MOM_12, dtype: float64
```

Percentage Price Oscillator (PPO)

[5]:
```
ppo_series = bta.ppo(fast=12, slow=26, signal=9, offset=0)
ppo_series.tail()
```

[5]:
```
    PPO_12_26_9  PPOH_12_26_9  PPOS_12_26_9
26    -6.436170    -11.002642      4.566472
27    -8.523310    -10.471825      1.948516
28   -12.216228    -11.331795     -0.884433
29   -16.248230    -12.291037     -3.957193
30   -19.603026    -12.516667     -7.086359
```

Rate of Change (ROC)

[6]:
```
roc_series = bta.roc(length=1, offset=0)
roc_series.tail()
```

[6]:
```
26   -5.739935
27   -2.539683
28   -0.917975
29   -2.450687
30    1.041667
Name: ROC_1, dtype: float64
```

Relative Strength Index (RSI)

[7]:
```
rsi_series = bta.rsi(length=1, drift=1, offset=0)
rsi_series.tail()
```

[7]:
```
26    16.266019
27    10.589262
28     8.499405
29     4.158062
```

(continues on next page)

(continued from previous page)

```
30    32.675188
Name: RSI_1, dtype: float64
```

Relative Vigor Index (RVI)

[8]:
```
rvi_series = bta.rvi(length=14, swma_length=4, offset=0)
rvi_series.tail()
```

[8]:
```
    RVI_14_4  RVIS_14_4
26 -0.300914  -0.282395
27 -0.214800  -0.303975
28 -0.182882  -0.260117
29 -0.148107  -0.205258
30 -0.122529  -0.166287
```

Slope

[11]:
```
slope_series = bta.slope(length=1, as_angle=False, to_degrees=False, offset=0)
slope_series.tail()
```

[11]:
```
26   -2.110000e-06
27   -8.800000e-07
28   -3.100000e-07
29   -8.200000e-07
30    3.400000e-07
Name: SLOPE_1, dtype: float64
```

Stochastic (STOCH)

[12]:
```
stoch_series = bta.stoch(fast_k=14, slow_k=5, slow_d=3, offset=0)
stoch_series.tail()
```

[12]:
```
    STOCHF_14   STOCHF_3    STOCH_5    STOCH_3
26   6.709265  22.935425  15.241707  14.207310
27  13.065977  17.236516  17.021569  15.752323
28   9.055627   9.610290  18.185576  16.816284
29   3.249097   8.456901  12.802855  16.003333
30  23.015873  11.773533  11.019168  14.002533
```

Trix (TRIX)

[13]:
```
trix_series = bta.trix(length=18, drift=1, offset=0)
trix_series.tail()
```

[13]:
```
26   -0.242958
27   -0.294361
28   -0.347659
29   -0.402758
30   -0.456325
Name: TRIX_18, dtype: float64
```

True Strength Index (TSI)

[14]:
```
tsi_series = bta.tsi(fast=13, slow=25, drift=1, offset=0)
tsi_series.tail()
```

[14]:
```
26    8.408619
27    7.977190
28    7.571647
29    7.105209
30    6.754438
Name: TSI_13_25, dtype: float64
```

Ultimate Oscillator (UO)

[15]:
```
uo_series = bta.uo(
        fast=7,
        medium=14,
        slow=28,
        fast_w=4.0,
        medium_w=2.0,
        slow_w=1.0,
        drift=1,
        offset=0,
    )
uo_series.tail()
```

[15]:
```
26          NaN
27    32.055171
28    32.909266
29    34.102532
30    36.042116
Name: UO_7_14_28, dtype: float64
```

William's Percent R (WILLR)

[17]:
```
willr_series = bta.willr(length=20, offset=0)
willr_series.tail()
```

[17]:
```
26   -90.027701
27   -98.949449
28   -99.271895
29   -99.804645
30   -97.614151
Name: WILLR_20, dtype: float64
```

Overlap

Double Exponential Moving Average (DEMA)

[18]:
```
dema_series = bta.dema(length=10, offset=0)
dema_series.tail()
```

[18]:
```
26    0.000036
27    0.000035
28    0.000034
29    0.000033
30    0.000033
Name: DEMA_10, dtype: float64
```

Exponential Moving Average (EMA)

[20]:
```
ema_series = bta.ema(length=10, offset=0)
ema_series.tail()
```

[20]:
```
26    0.000038
27    0.000038
28    0.000037
29    0.000036
30    0.000035
Name: EMA_10, dtype: float64
```

Fibonacci's Weighted Moving Average (FWMA)

[21]:
```
fwma_series = bta.fwma(length=10, asc=True, offset=0)
fwma_series.tail()
```

[21]:
```
26    0.000036
27    0.000035
28    0.000035
29    0.000034
30    0.000033
Name: FWMA_10, dtype: float64
```

Hull Moving Average (HMA)

[22]:
```
hma_series = bta.hma(length=10, offset=0)
hma_series.tail()
```

[22]:
```
26    0.000035
27    0.000034
28    0.000034
29    0.000033
30    0.000032
Name: HMA_10, dtype: float64
```

Ichimoku Kinkō Hyō (ichimoku)

[24]:
```
ichimoku_pd_1, ichimoku_pd_2 = bta.ichimoku(tenkan=9, kijun=26, senkou=52, offset=0)
```

[25]:
```
ichimoku_pd_1.tail()
```

[25]:
```
    ISA_9  ISB_26     ITS_9    IKS_26  ICS_26
26    NaN     NaN  0.000042  0.000072     NaN
27    NaN     NaN  0.000039  0.000072     NaN
28    NaN     NaN  0.000038  0.000072     NaN
29    NaN     NaN  0.000036  0.000072     NaN
30    NaN     NaN  0.000036  0.000072     NaN
```

[26]:
```
ichimoku_pd_2.tail()
```

[26]:
```
       ISA_9  ISB_26
52  0.000057     NaN
53  0.000055     NaN
54  0.000055     NaN
55  0.000054     NaN
56  0.000054     NaN
```

Kaufman's Adaptive Moving Average (KAMA)

[4]:
```
kama_series = bta.kama(length=10, fast=2, slow=30, drift=1, offset=0)
kama_series.tail()
```

[4]:
```
26    0.000030
27    0.000031
28    0.000032
29    0.000032
30    0.000032
Name: KAMA_10_2_30, dtype: float64
```

Linear Regression Moving Average (linreg)

[5]:
```
linreg_series = bta.linreg(length=10, offset=0)
linreg_series.tail()
```

[5]:
```
26    0.000035
27    0.000034
28    0.000034
29    0.000034
30    0.000033
Name: LR_10, dtype: float64
```

Pascal's Weighted Moving Average (PWMA)

[6]:
```
pwma_series = bta.pwma(length=10, asc=True, offset=0)
pwma_series.tail()
```

[6]:
```
26    0.000037
27    0.000037
28    0.000036
29    0.000036
30    0.000035
Name: PWMA_10, dtype: float64
```

wildeR's Moving Average (RMA)

[7]:
```
rma_series = bta.rma(length=10, offset=0)
rma_series.tail()
```

[7]:
```
26    0.000041
27    0.000040
28    0.000040
29    0.000039
30    0.000038
Name: RMA_10, dtype: float64
```

Sine Weighted Moving Average (SINWMA)

[10]:
```
sinwma_series = bta.sinwma(length=10, offset=0)
sinwma_series.tail()
```

[10]:
```
26    0.000038
27    0.000037
28    0.000036
29    0.000036
30    0.000035
Name: SINWMA_10, dtype: float64
```

Simple Moving Average (SMA)

[9]:
```
sma_series = bta.sma(length=10, offset=0)
sma_series.tail()
```

[9]:
```
26    0.000038
27    0.000038
28    0.000037
29    0.000036
30    0.000035
Name: EMA_10, dtype: float64
```

Symmetric Weighted Moving Average (SWMA)

[11]:
```
swma_series = bta.swma(length=10, offset=0)
swma_series.tail()
```

[11]:
```
26    0.000037
27    0.000037
28    0.000036
29    0.000036
30    0.000035
Name: SWMA_10, dtype: float64
```

Tim Tillson's T3 Moving Average (T3)

[12]:
```
t3_series = bta.t3(length=10, a=0.7, offset=0)
t3_series.tail()
```

[12]:
```
26    0.000041
27    0.000040
28    0.000039
29    0.000038
30    0.000037
Name: T3_10_0.7, dtype: float64
```

Triple Exponential Moving Average (TEMA)

[13]:
```
tema_series = bta.tema(length=10, offset=0)
tema_series.tail()
```

[13]:
```
26    0.000034
27    0.000033
28    0.000033
29    0.000032
30    0.000031
Name: TEMA_10, dtype: float64
```

Triangular Moving Average (TRIMA)

[14]:
```
trima_series = bta.trima(length=10, offset=0)
trima_series.tail()
```

[14]:
```
26    0.000039
27    0.000038
28    0.000037
29    0.000036
30    0.000035
Name: TRIMA_10, dtype: float64
```

Volume Weighted Average Price (VWAP)

[15]:
```
vwap_series = bta.vwap(offset=0)
vwap_series.tail()
```

[15]:
```
26    0.000068
27    0.000068
28    0.000068
29    0.000068
30    0.000068
Name: VWAP, dtype: float64
```

Volume Weighted Moving Average (VWMA)

[16]:
```
vwma_series = bta.vwma(length=10, offset=0)
vwma_series.tail()
```

[16]:
```
26    0.000040
27    0.000038
28    0.000037
29    0.000036
30    0.000035
Name: VWMA_10, dtype: float64
```

Weighted Moving Average (WMA)

[17]:
```
wma_series = bta.wma(length=10, asc=True, offset=0)
wma_series.tail()
```

[17]:
```
26    0.000037
27    0.000036
28    0.000035
29    0.000035
30    0.000034
Name: WMA_10, dtype: float64
```

Zero Lag Moving Average (ZLMA)

[19]:
```
zlma_series = bta.zlma(length=10, offset=0, mamode='ema')
zlma_series.tail()
```

[19]:
```
26    0.000035
27    0.000035
28    0.000034
29    0.000033
30    0.000032
Name: ZLEMA_10, dtype: float64
```

Log Return

[20]:
```
log_return_series = bta.log_return(length=10, cumulative=False, percent=False,␣
↪offset=0)
log_return_series.tail()
```

[20]:
```
26   -0.289628
27   -0.344520
28   -0.294087
29   -0.258776
30   -0.175491
Name: LOGRET_10, dtype: float64
```

[21]:
```
percent_return_series = bta.percent_return(length=10, cumulative=False, percent=False,
↪ offset=0)
percent_return_series.tail()
```

[21]:
```
26   -0.251458
27   -0.291439
28   -0.254788
29   -0.228004
30   -0.160955
Name: PCTRET_10, dtype: float64
```

Trend Return

[29]:
```
# trend_return_series = bta.trend_return(trend="close", log=True, cumulative=False,␣
↪offset=0, trend_reset=0)
# trend_return_series.tail()
```

Rolling Kurtosis

[30]:
```
kurtosis_series = bta.kurtosis(length=10, offset=0, mamode='ema')
kurtosis_series.tail()
```

[30]:
```
26    0.065587
27    1.024344
28    1.768071
29   -1.197006
30   -1.618680
Name: KURT_10, dtype: float64
```

Rolling Mean Absolute Deviation

[31]:
```
mad_series = bta.mad(length=10, offset=0)
mad_series.tail()
```

[31]:
```
26    0.000004
27    0.000003
28    0.000002
29    0.000002
30    0.000002
Name: MAD_10, dtype: float64
```

Rolling Median

[32]:
```
median_series = bta.median(length=10, offset=0)
median_series.tail()
```

[32]:
```
26    0.000037
27    0.000037
28    0.000037
29    0.000036
30    0.000035
Name: MEDIAN_10, dtype: float64
```

Rolling Quantile

[33]:
```
quantile_series = bta.quantile(length=30, q=0.5, offset=0)
quantile_series.tail()
```

[33]:
```
26         NaN
27         NaN
28         NaN
29    0.000037
30    0.000037
Name: QTL_30_0.5, dtype: float64
```

Rolling Skew

[34]:
```
skew_series = bta.skew(length=30, offset=0)
skew_series.tail()
```

[34]:
```
26         NaN
27         NaN
28         NaN
29    1.928790
30    2.001403
Name: SKEW_30, dtype: float64
```

Rolling Standard Deviation

[35]:
```
stdev_series = bta.stdev(length=30, offset=0)
stdev_series.tail()
```

[35]:
```
26         NaN
27         NaN
28         NaN
29    0.000021
30    0.000020
Name: STDEV_30, dtype: float64
```

Rolling Variance

[36]:
```
variance_series = bta.variance(length=30, offset=0)
variance_series.tail()
```

[36]:
```
26            NaN
27            NaN
28            NaN
29    4.263450e-10
30    4.121269e-10
Name: VAR_30, dtype: float64
```

Rolling Z Score

[37]:
```
zscore_series = bta.zscore(length=30, std=1, offset=0)
zscore_series.tail()
```

[37]:
```
26         NaN
27         NaN
28         NaN
29   -0.386298
30   -0.430835
Name: Z_30, dtype: float64
```

Trend

Average Directional Movement (ADX)

[38]:
```
adx_series = bta.adx(length=14, drift=1, offset=0)
adx_series.tail()
```

[38]:
```
       ADX_14     DMP_14     DMN_14
26        NaN  43.174417  24.456502
27  33.907743  43.244219  26.401392
28  32.872561  44.458695  27.142852
29  31.783427  45.526168  28.934007
30  30.660034  46.291902  30.529761
```

Aroon (AROON)

[40]:
```
aroon_series = bta.aroon(length=1, offset=0)
aroon_series.tail()
```

[40]:
```
    AROOND_1  AROONU_1
26     100.0     100.0
27     100.0     100.0
28     100.0     100.0
29     100.0     100.0
30     100.0     100.0
```

Decreasing

[44]:
```
decreasing_series = bta.decreasing(length=1, asint=True, offset=0)
decreasing_series.tail()
```

[44]:
```
26    1
27    1
28    1
29    1
30    1
Name: DEC_1, dtype: int64
```

Detrend Price Oscillator (DPO)

[45]:
```
dpo_series = bta.dpo(length=1, centered=True, offset=0)
dpo_series.tail()
```

[45]:
```
26    8.800000e-07
27    3.100000e-07
28    8.200000e-07
29    3.200000e-07
30             NaN
Name: DPO_1, dtype: float64
```

Increasing

[46]:
```
increasing_series = bta.increasing(length=1, asint=True, offset=0)
increasing_series.tail()
```

[46]:
```
26    0
27    0
28    0
29    0
30    0
Name: INC_1, dtype: int64
```

Linear Decay

[47]:
```
linear_decay_series = bta.linear_decay(length=1, offset=0)
linear_decay_series.tail()
```

[47]:
```
26    0.000035
27    0.000034
28    0.000033
29    0.000033
30    0.000032
Name: LDECAY_1, dtype: float64
```

Q Stick

[48]:
```
qstick_series = bta.qstick(length=1, offset=0)
qstick_series.tail()
```

[48]:
```
26   -2.110000e-06
27   -8.800000e-07
28   -3.100000e-07
29   -8.200000e-07
30   -3.200000e-07
Name: QS_1, dtype: float64
```

Vortex

[49]:
```
vortex_series = bta.vortex(length=1, offset=0)
vortex_series.tail()
```

[49]:
```
      VTXP_1    VTXM_1
26  0.311787  1.433460
27  0.445283  1.547170
28  1.311111  1.651852
29  0.478261  1.500000
30  0.573333  1.346667
```

Acceleration Bands (ACCBANDS)

[52]:
```
vortex_series = bta.accbands(
    length=10,
    c=4,
    drift=1,
    mamode='ema',
    offset=0
)
vortex_series.tail()
```

[52]:
```
    ACCBL_10  ACCBM_10  ACCBU_10
26  0.000027  0.000039  0.000055
27  0.000027  0.000038  0.000052
28  0.000028  0.000037  0.000049
29  0.000028  0.000036  0.000047
30  0.000028  0.000036  0.000045
```

Average True Range (ATR)

[53]:
```
atr_series = bta.atr(length=1, mamode='ema', offset=0)
atr_series.tail()
```

[53]:
```
26    0.000003
27    0.000003
28    0.000001
29    0.000001
30    0.000002
Name: ATR_1, dtype: float64
```

Bollinger Bands (BBANDS)

[54]:
```
bbands_series = bta.bbands(length=20, std=2, mamode='ema', offset=0)
bbands_series.tail()
```

[54]:
```
          BBL_20    BBM_20    BBU_20
26  6.995546e-07  0.000040  0.000079
27 -2.899943e-07  0.000039  0.000079
28 -2.216349e-07  0.000039  0.000077
29  1.852932e-05  0.000038  0.000058
30  2.409578e-05  0.000038  0.000051
```

Donchian Channels (DC)

[55]:
```
donchian_series = bta.donchian(lower_length=10, upper_length=20, offset=0)
donchian_series.tail()
```

[55]:
```
    DCL_10_20  DCM_10_20  DCU_10_20
26   0.000035   0.000077   0.000118
27   0.000034   0.000076   0.000118
28   0.000033   0.000076   0.000118
29   0.000033   0.000053   0.000073
30   0.000032   0.000044   0.000056
```

Keltner Channels (KC)

[56]:
```
kc_series = bta.kc(length=20, scalar=2, mamode='ema', offset=0)
kc_series.tail()
```

[56]:
```
      KCL_20    KCB_20    KCU_20
26  0.000027  0.000041  0.000055
27  0.000028  0.000040  0.000053
28  0.000028  0.000040  0.000051
29  0.000029  0.000039  0.000049
30  0.000029  0.000038  0.000048
```

Mass Index (MASSI)

[60]:
```
massi_series = bta.massi(fast=9, slow=25, offset=0)
massi_series.tail()
```

[60]:
```
26   NaN
27   NaN
28   NaN
29   NaN
30   NaN
Name: MASSI_9_25, dtype: float64
```

Normalized Average True Range (NATR)

[61]:
```
natr_series = bta.natr(length=20, mamode='ema', drift=1, offset=0)
natr_series.tail()
```

[61]:
```
26    24.382287
27    23.276758
28    21.531668
29    20.284296
30    18.905182
Name: NATR_20, dtype: float64
```

True Range

[62]:
```
true_range_series = bta.true_range(drift=1, offset=0)
true_range_series.tail()
```

[62]:
```
26    0.000003
27    0.000003
28    0.000001
29    0.000001
30    0.000002
Name: TRUERANGE_1, dtype: float64
```

Volume

Accumulation/Distribution (AD)

[63]:
```
ad_series = bta.ad(offset=0)
ad_series.tail()
```

[63]:
```
26   -10.025277
27   -13.443919
28   -17.399771
29   -21.928423
30   -23.634089
Name: AD, dtype: float64
```

Accumulation/Distribution Oscillator or Chaikin Oscillator

[64]:
```
adosc_series = bta.adosc(fast=12, slow=26, offset=0)
adosc_series.tail()
```

[64]:
```
26   -53.875972
27   -53.644655
28   -53.167654
29   -52.549297
30   -51.602041
Name: ADOSC_12_26, dtype: float64
```

Chaikin Money Flow (CMF)

[65]:
```
cmf_series = bta.cmf(length=20, offset=0)
cmf_series.tail()
```

[65]:
```
26   -0.024778
27   -0.164204
28   -0.215310
29   -0.625480
30   -0.483050
Name: CMF_20, dtype: float64
```

Elder's Force Index (EFI)

[66]:
```
efi_series = bta.efi(length=13, drift=1, mamode='ema', offset=0)
efi_series.tail()
```

[66]:
```
26    0.000105
27    0.000088
28    0.000075
29    0.000064
30    0.000054
Name: EFI_13, dtype: float64
```

Ease of Movement (EOM)

[67]:
```
eom_series = bta.eom(length=14, drift=1, offset=0)
eom_series.tail()
```

[67]:
```
26   -0.000038
27   -0.000036
28   -0.000031
29   -0.000025
30   -0.000029
Name: EOM_14_100000000, dtype: float64
```

Money Flow Index (MFI)

[68]:
```
mfi_series = bta.mfi(length=14, drift=1, offset=0)
mfi_series.tail()
```

[68]:
```
26    45.056603
27    48.168097
28    54.635662
29    58.310922
30    46.966315
Name: MFI_14, dtype: float64
```

Negative Volume Index (NVI)

[69]:
```
nvi_series = bta.nvi(length=13, initial=1_000, offset=0)
nvi_series.tail()
```

[69]:
```
26    1447.481869
27    1429.308862
28    1417.968110
29    1388.480099
30    1356.293779
Name: NVI_13, dtype: float64
```

On Balance Volume (OBV)

[70]:
```
obv_series = bta.obv(offset=0)
obv_series.tail()
```

[70]:
```
26    400.875
27    386.495
28    379.735
29    373.608
30    368.491
Name: OBV, dtype: float64
```

Positive Volume Index (PVI)

[71]:
```
pvi_series = bta.pvi(length=13, initial=1_000, offset=0)
pvi_series.tail()
```

[71]:
```
26    1247.036814
27    1247.036814
28    1247.036814
29    1247.036814
30    1247.036814
Name: PVI_13, dtype: float64
```

Price-Volume (PVOL)

[72]:
```
pvol_series = bta.pvol(offset=0)
pvol_series.tail()
```

[72]:
```
26    0.000650
27    0.000486
28    0.000226
29    0.000200
30    0.000165
Name: PVOL, dtype: float64
```

Price-Volume Trend (PVT)

[73]:
```
pvt_series = bta.pvt(drift=1, offset=0)
pvt_series.tail()
```

[73]:
```
26    60957.106450
27    60920.585815
28    60914.380307
29    60899.364945
30    60894.348279
Name: PVT, dtype: float64
```

Volume Profile (VP)

[74]:
```
vp_series = bta.vp(width=10)
vp_series.tail()
```

[74]:

	low_close	mean_close	high_close	pos_volume	neg_volume	total_volume
5	0.000045	0.000046	0.000048	174.200	44.120	218.320
6	0.000037	0.000039	0.000042	8.082	27.090	35.172
7	0.000035	0.000036	0.000037	63.750	0.000	63.750
8	0.000034	0.000035	0.000037	18.770	22.503	41.273
9	0.000032	0.000033	0.000033	0.000	18.004	18.004

[ ]:
```

```

2.2.4 Ionomy

Imports and Params

[1]:
```
from Ionomy import Ionomy
from decouple import config

MARKET = 'btc-hive'
CURRENCY = 'hive'
BASE = 'btc'
TIME = 'day'
BTC_QUANTITY = 0.001
HIVE_QUANTITY = 100
LOW_RATE=0.00003
HIGH_RATE=0.00004
ADDRESS = 'memehub'
```

Instantiation

[2]:
```
ion = Ionomy(config('IONOMY_KEY'), config('IONOMY_SECRET'))
```

Public Endpoint Methods

Markets

[3]:
```
markets = ion.markets()
markets[0]
```

[3]:
```
{'market': 'btc-eth',
 'title': 'Bitcoin:Ethereum',
 'currencyBase': 'btc',
 'currencyMarket': 'eth',
 'orderMinSize': '0.00001000',
 'buyFee': '0.20000000',
 'sellFee': '0.20000000',
 'inMaintenance': False}
```

Currencies

[6]:
```
currencies = ion.currencies()
currencies[0]
```

[6]:
```
{'currency': 'dash',
 'title': 'Dash',
 'withdrawMinSize': '0.00100000',
 'withdrawFee': '0.00200000',
 'inMaintenance': False,
 'canDeposit': 1,
 'canWithdraw': 1}
```

Order Book

[11]:
```
order_book = ion.order_book(MARKET)
order_book["bids"][0]
```

[11]:
```
{'size': '207.00000000', 'price': '0.00003301'}
```

Market Summaries

[12]:
```
market_summaries = ion.market_summaries()
market_summaries[0]
```

[12]:
```
{'market': 'btc-eth',
 'high': '0.02150000',
 'low': '0.01900001',
 'volume': '0.31467562',
 'price': '0.01900001',
 'change': '-11.63',
 'baseVolume': '0.00597884',
 'bidsOpenOrders': '10',
 'bidsLastPrice': '0.01900003',
 'highestBid': '0.01900003',
 'asksOpenOrders': '14',
 'asksLastPrice': '0.02250000',
 'lowestAsk': '0.02250000'}
```

Market Summary

[14]:
```
market_summary = ion.market_summary(MARKET)
market_summary
```

[14]:
```
{'market': 'btc-hive',
 'high': '0.00003538',
 'low': '0.00003300',
 'volume': '3856.61483680',
 'price': '0.00003370',
 'change': '2.06',
 'baseVolume': '0.12996792',
 'bidsOpenOrders': '39',
```

(continues on next page)

(continued from previous page)

```
 'bidsLastPrice': '0.00003301',
 'highestBid': '0.00003301',
 'asksOpenOrders': '47',
 'asksLastPrice': '0.00003447',
 'lowestAsk': '0.00003370'}
```

Market History

[15]:
```
market_history = ion.market_history(MARKET)
market_history[0]
```

[15]:
```
{'type': 'MARKET_BUY',
 'amount': '140.93003798',
 'price': '0.00003370',
 'total': '0.00474934',
 'createdAt': '2020-05-16T09:28:58Z'}
```

Market Endpoint Methods

Limit Buy/Sell Order

[3]:
```
order = ion.limit_buy(HIVE_QUANTITY, LOW_RATE, MARKET)
# order = ion.limit_sell(HIVE_QUANTITY, HIGH_RATE, MARKET)
```

```
---------------------------------------------------------------------------
Exception                                 Traceback (most recent call last)
<ipython-input-3-8453aa1ba66a> in <module>
      1 order = ion.limit_buy(HIVE_QUANTITY, LOW_RATE, MARKET)
----> 2 order = ion.limit_sell(HIVE_QUANTITY, HIGH_RATE, MARKET)

/notebooks/packages/Ionomy/ionomy.py in limit_sell(self, amount, price, market)
     92             'price': f'{price:.8f}'
     93         }
---> 94         return self._request('market/sell-limit', params)
     95
     96     def cancel_order(self, orderId: str) -> bool:

/notebooks/packages/Ionomy/ionomy.py in _request(self, endpoint, params)
     47         data = json.loads(response.content)
     48         if not data['success']:
---> 49             raise Exception(data['message'])
     50         return data['data']
     51

Exception: 40017
```

Cancel Order

[9]:
```
success = ion.cancel_order(order['orderId'])
success
```

[9]:
```
True
```

Order Status

[8]:
```
order_status = ion.order_status(order['orderId'])
order_status
```

[8]:
```
{'orderId': '5ec1a9d18b196d3e336cf6d2',
 'status': 'OPEN',
 'market': 'btc-hive',
 'type': 'LIMIT_BUY',
 'amount': '100.00000000',
 'price': '0.00003000',
 'filled': '0.00000000',
 'opened': '2020-05-17T21:17:05Z',
 'closed': None}
```

Account Endpoint Methods

Open Orders

[4]:
```
open_orders = ion.open_orders(MARKET)
open_orders[0]
```

[4]:
```
[{'orderId': '5ec1a9d18b196d3e336cf6d2',
  'market': 'btc-hive',
  'type': 'LIMIT_BUY',
  'amount': '100.00000000',
  'price': '0.00003000',
  'filled': '0.00000000',
  'createdAt': '2020-05-17T21:17:05Z'}]
```

Balances

[18]:
```
balances = ion.balances()
balances[0]
```

[18]:
```
{'currency': 'btc', 'available': '0.11574671', 'reserved': '0.00000001'}
```

Balance

[19]:
```
balance = ion.balance(CURRENCY)
balance
```

[19]:
```
{'currency': 'hive', 'available': '0.54814539', 'reserved': '0.00000000'}
```

Deposit Address

[20]:
```
deposit_address = ion.deposit_address(CURRENCY)
deposit_address
```

[20]:
```
{'currency': 'hive', 'address': '5e8f6cd1a2a3e2080524eb42'}
```

Deposit History

[29]:
```
deposit_history = ion.deposit_history("Steem")
deposit_history["deposits"][0]
```

[29]:
```
{'transactionId': '5e8f6db5149be70cd35ab8e2',
 'state': 'PROCESSED',
 'currency': 'steem',
 'amount': '1.00000000',
 'createdAt': '2020-04-09T18:47:17Z'}
```

Withdrawal History

[25]:
```
withdrawal_history = ion.withdrawal_history(CURRENCY)
withdrawal_history[0]
```

[25]:
```
{'transactionId': '5e8f726da2a3e220e6566b72',
 'state': 'PROCESSED',
 'currency': 'hive',
 'amount': '7186.01000000',
 'feeAmount': '0.01000000',
 'createdAt': '2020-04-09T19:07:25Z'}
```

2.2.5 IonPanda

Imports and Params

[1]:
```
from Ionomy import IonPanda
from decouple import config

MARKET = 'btc-hive'
CURRENCY = 'hive'
BASE = 'btc'
TIME = 'day'
AMOUNT = 1
PRICE = 0.0000001
```

Instantiation

[2]:
```
ionpd = IonPanda(config('IONOMY_KEY'), config('IONOMY_SECRET'))
```

Public Endpoint Methods

Markets

[3]:
```
markets_pd = ionpd.markets()
markets_pd.head()
```

[3]:
```
      market             title currencyBase currencyMarket  orderMinSize  \
0    btc-eth  Bitcoin:Ethereum          btc            eth       0.00001
1    btc-ion       Bitcoin:Ion          btc            ion       0.00001
2   btc-dash      Bitcoin:Dash          btc           dash       0.00001
3   btc-pivx      Bitcoin:Pivx          btc           pivx       0.00001
4  btc-atoms     Bitcoin:Atoms          btc          atoms       0.00001

   buyFee  sellFee  inMaintenance
0     0.2      0.2          False
1     0.2      0.2          False
2     0.2      0.2          False
3     0.2      0.2          False
4     0.0      0.4          False
```

Currencies

[4]:
```
currencies_pd = ionpd.currencies()
currencies_pd.head()
```

[4]:
```
  currency     title  withdrawMinSize  withdrawFee  inMaintenance  canDeposit  \
0     dash      Dash            0.001       0.0020          False        True
1     pivx      Pivx            0.001       0.1000          False        True
2      ion       Ion            0.250       0.5000          False        True
3      btc   Bitcoin            0.001       0.0005          False        True
4      eth  Ethereum            0.001       0.0060          False        True

   canWithdraw
0         True
1         True
2         True
3         True
4         True
```

Order Book

[6]:
```
order_book_pd = ionpd.order_book(MARKET)
order_book_pd.head()
```

[6]:
```
    size     price type
0  207.0  0.000033  bid
1    3.2  0.000033  bid
2  160.0  0.000032  bid
3  143.0  0.000032  bid
4  100.0  0.000032  bid
```

Market Summaries

[7]:
```
market_summaries_pd = ionpd.market_summaries()
market_summaries_pd.head()
```

[7]:
```
     market      high       low       volume     price  change  baseVolume  \
0   btc-ion  0.000003  0.000003  3488.067194  0.000003  -17.05    0.008825
1   btc-eth  0.021500  0.019000     0.314676  0.019000  -11.63    0.005979
2   btc-ltc  0.004750  0.000560    11.346268  0.000560  -13.85    0.006354
3  btc-dash  0.007992  0.007949     0.130268  0.007949   -0.54    0.001035
4  btc-pivx  0.000043  0.000028   345.795984  0.000028  -21.96    0.009644

   bidsOpenOrders  bidsLastPrice  highestBid  asksOpenOrders  asksLastPrice  \
0               9       0.000003    0.000003             163       0.000003
1              10       0.019000    0.019000              14       0.022500
2              10       0.004500    0.004500              43       0.005300
3              10       0.008000    0.007464              42       0.007949
4               7       0.000029    0.000028              19       0.000036

   lowestAsk
0   0.000003
1   0.022500
2   0.005300
3   0.007949
4   0.000043
```

Market Summary

[8]:
```
market_summary = ionpd.market_summary(MARKET)
market_summary
```

[8]:
```
{'market': 'btc-hive',
 'high': '0.00003538',
 'low': '0.00003300',
 'volume': '3856.61483680',
 'price': '0.00003370',
 'change': '2.06',
 'baseVolume': '0.12996792',
 'bidsOpenOrders': '39',
 'bidsLastPrice': '0.00003301',
 'highestBid': '0.00003301',
 'asksOpenOrders': '47',
 'asksLastPrice': '0.00003447',
 'lowestAsk': '0.00003370'}
```

Market History

[9]:
```
market_history = ionpd.market_history(MARKET)
market_history.head()
```

[9]:
```
         type      amount     price     total           createdAt
0  MARKET_BUY  140.930038  0.000034  0.004749 2020-05-16 09:28:58
1  MARKET_BUY   71.402354  0.000034  0.002463 2020-05-16 06:14:42
2  MARKET_BUY   88.000000  0.000034  0.002979 2020-05-16 06:14:42
```

(continues on next page)

(continued from previous page)

```
3   LIMIT_BUY   50.000000  0.000034  0.001692 2020-05-16 06:01:16
4  MARKET_BUY   74.982622  0.000034  0.002579 2020-05-16 03:27:41
```

Market Endpoint Methods

Limit Buy/Sell Orders

[ ]:
```
order = ionpd.limit_buy(AMOUNT, PRICE, MARKET)
order = ionpd.limit_sell(AMOUNT, PRICE, MARKET)
```

Success

[ ]:
```
success = ionpd.cancel_order(ORDERID)
```

Order Status

[ ]:
```
order_status = ionpd.order_status(ORDERID)
```

Account Endpoint Methods

Open Orders

[10]:
```
open_orders_pd = ionpd.open_orders(MARKET)
open_orders_pd
```

[10]:
```
Empty DataFrame
Columns: [orderId, market, type, amount, price, filled, createdAt]
Index: []
```

Balances

[11]:
```
balances_pd = ionpd.balances()
balances_pd.head()
```

[11]:
```
       currency  available      reserved
0           btc   0.115747  1.000000e-08
1           ion   0.000000  0.000000e+00
2         steem   0.560000  0.000000e+00
3          hive   0.548145  0.000000e+00
4  gravity-live   0.000000  0.000000e+00
```

Balance

[12]:
```
balance = ionpd.balance(CURRENCY)
balance
```

[12]:
```
{'currency': 'hive', 'available': '0.54814539', 'reserved': '0.00000000'}
```

Deposit Address

[13]:
```
deposit_address = ionpd.deposit_address(CURRENCY)
deposit_address
```

[13]:
```
{'currency': 'hive', 'address': '5e8f6cd1a2a3e2080524eb42'}
```

Deposit History

[14]:
```
deposit_history_pd = ionpd.deposit_history(CURRENCY)
deposit_history_pd.head()
```

[14]:
```
Empty DataFrame
Columns: [currency, deposits]
Index: []
```

Withdrawal History

[15]:
```
withdrawal_history_pd = ionpd.withdrawal_history(CURRENCY)
withdrawal_history_pd.head()
```

[15]:
```
              transactionId      state currency       amount  feeAmount  \
0  5e8f726da2a3e220e6566b72  PROCESSED     hive  7186.010000       0.01
1  5e8f9934a2a3e24bda6b9962  PROCESSED     hive   579.563903       0.01
2  5e9a6fe4a2a3e2016530b782  PROCESSED     hive  6194.010000       0.01
3  5ea24752a2a3e2245f19e352  PROCESSED     hive  4308.010000       0.01
4  5eacd1792249e5764b7323d2  PROCESSED     hive   124.050000       0.05

            createdAt
0 2020-04-09 19:07:25
1 2020-04-09 21:52:52
2 2020-04-18 03:11:32
3 2020-04-24 01:56:34
4 2020-05-02 01:48:41
```

[ ]:
```

```

2.3 Modules

2.3.1 BitTrex

class Ionomy.**BitTrex**(*api_key: str*, *secret_key: str*)

Bases: object

BitTrex API Wrapper

Parameters

- **{str} -- BitTrex API Key** (*api_key*) –
- **{str} -- BitTrex API Secret** (*secret_key*) –

balance (*currency: str*) → List[Dict[str, Union[str, float, None]]]

balances () → List[Dict[str, Union[str, float, None]]]

buy_limit (*market: str*, *quantity: float*, *rate: float*, *timeInForce: str*) → str

cancel (*uuid: str*) → bool

currencies () → List[Dict[str, Union[bool, str, float, None]]]

deposit_address (*currency: str*)

deposit_history (*currency: str*) → List[Dict[str, Union[str, int, float, None]]]

get_order (*uuid: str*)

market_history (*market: str*) → List[Dict[str, Any]]

market_summaries () → List[Dict[str, Union[bool, int, float, str, None]]]

market_summary (*market: str*) → Dict[str, Union[str, int, float]]

markets () → List[Dict[str, Union[bool, str, float]]]

ohlcv (*currency: str*, *base: str*, *time: str*, *limit: int*) → List[Dict[str, Union[str, int, float]]]

open_orders (*market: str*) → List[Dict[str, Union[bool, str, float, None]]]

order_book (*market: str*) → Dict[str, List[Dict[str, Any]]]

order_history () → List[Dict[str, Any]]

sell_limit (*market: str, quantity: float, rate: float, timeInForce: Optional[str]*) → str

ticker (*market: str*) → Dict[str, float]

withdraw (*currency: str, quantity: float, address: str, paymentid: Optional[str]*)

withdrawal_history (*currency: str*) → List[Dict[str, Union[bool, str, float, None]]]

2.3.2 BitPanda

class Ionomy.**BitPanda** (*api_key: str*, *secret_key: str*)

Bases: Ionomy.bittrex.BitTrex

Pandas DataFrame Wrapper for BitTrex Base Class

Parameters

- **{str} -- BitTrex API Key** (*api_key*) –
- **{str} -- BitTrex API Secret** (*secret_key*) –

balances () → pandas.core.frame.DataFrame

currencies () → pandas.core.frame.DataFrame

deposit_history (*currency: str*) → pandas.core.frame.DataFrame

market_history (*market: str*) → pandas.core.frame.DataFrame

market_summaries () → pandas.core.frame.DataFrame

markets() → pandas.core.frame.DataFrame

ohlcv(*currency: str*, *base: str*, *time: str*, *limit: int*) → pandas.core.frame.DataFrame

order_book(*market: str*) → pandas.core.frame.DataFrame

order_history() → pandas.core.frame.DataFrame

withdrawal_history(*currency: str*) → pandas.core.frame.DataFrame

2.3.3 BitTA

class Ionomy.**BitTA**(*api_key: str*, *secret_key: str*)
Bases: Ionomy.bit_panda.BitPanda

Technical Analysis Wrapper for BitPanda

Parameters

- **{str} -- BitTrex API Key** (*api_key*) –
- **{str} -- BitTrex API Secret** (*secret_key*) –

accbands(*length=None*, *c=None*, *drift=None*, *mamode=None*, *offset=None*, ***kwargs*)
Acceleration Bands (ACCBANDS)

Acceleration Bands created by Price Headley plots upper and lower envelope bands around a simple moving average.

Sources: https://www.tradingtechnologies.com/help/x-study/technical-indicator-definitions/acceleration-bands-abands/

Calculation:

Default Inputs: length=10, c=4

EMA = Exponential Moving Average

SMA = Simple Moving Average

HL_RATIO = c * (high - low) / (high + low)

LOW = low * (1 - HL_RATIO)

HIGH = high * (1 + HL_RATIO)

if 'ema': LOWER = EMA(LOW, length)

MID = EMA(close, length)

UPPER = EMA(HIGH, length)

else: LOWER = SMA(LOW, length)

MID = SMA(close, length)

UPPER = SMA(HIGH, length)

Parameters

- **high** (*pd.Series*) – Series of 'high's
- **low** (*pd.Series*) – Series of 'low's
- **close** (*pd.Series*) – Series of 'close's
- **length** (*int*) – It's period. Default: 10

- **c** (*int*) – Multiplier. Default: 4
- **mamode** (*str*) – Two options: None or 'ema'. Default: 'ema'
- **drift** (*int*) – The difference period. Default: 1
- **offset** (*int*) – How many periods to offset the result. Default: 0

Kwargs: fillna (value, optional): pd.DataFrame.fillna(value)

fill_method (value, optional): Type of fill method

Returns lower, mid, upper columns.

Return type pd.DataFrame

ad(*offset=None*, ***kwargs*)
Accumulation/Distribution (AD)

Accumulation/Distribution indicator utilizes the relative position of the close to it's High-Low range with volume. Then it is cumulated.

Sources: https://www.tradingtechnologies.com/help/x-study/technical-indicator-definitions/accumulationdistribution-ad/

Calculation: CUM = Cumulative Sum if 'open':

AD = close - open

else: AD = 2 * close - high - low

hl_range = high - low AD = AD * volume / hl_range AD = CUM(AD)

Parameters

- **high** (*pd.Series*) – Series of 'high's
- **low** (*pd.Series*) – Series of 'low's
- **close** (*pd.Series*) – Series of 'close's
- **volume** (*pd.Series*) – Series of 'volume's
- **open** (*pd.Series*) – Series of 'open's
- **offset** (*int*) – How many periods to offset the result. Default: 0

Kwargs: fillna (value, optional): pd.DataFrame.fillna(value) fill_method (value, optional): Type of fill method

Returns New feature generated.

Return type pd.Series

adosc(*fast=None*, *slow=None*, *offset=None*, ***kwargs*)
Accumulation/Distribution Oscillator or Chaikin Oscillator

Accumulation/Distribution Oscillator indicator utilizes Accumulation/Distribution and treats it similarily to MACD or APO.

Sources: https://www.investopedia.com/articles/active-trading/031914/understanding-chaikin-oscillator.asp

Calculation:

Default Inputs: fast=12, slow=26

AD = Accum/Dist

ad = AD(high, low, close, open)

fast_ad = EMA(ad, fast)

slow_ad = EMA(ad, slow)

ADOSC = fast_ad - slow_ad

Parameters

- **high** (*pd.Series*) – Series of 'high's
- **low** (*pd.Series*) – Series of 'low's
- **close** (*pd.Series*) – Series of 'close's
- **open** (*pd.Series*) – Series of 'open's
- **volume** (*pd.Series*) – Series of 'volume's
- **fast** (*int*) – The short period. Default: 12
- **slow** (*int*) – The long period. Default: 26
- **offset** (*int*) – How many periods to offset the result. Default: 0

Kwargs: fillna (value, optional): pd.DataFrame.fillna(value)

fill_method (value, optional): Type of fill method

Returns New feature generated.

Return type pd.Series

adx(*length=None*, *drift=None*, *offset=None*, ***kwargs*)
Average Directional Movement (ADX)

Average Directional Movement is meant to quantify trend strength by measuring the amount of movement in a single direction.

Sources: https://www.tradingtechnologies.com/help/x-study/technical-indicator-definitions/average-directional-movement-adx/

Calculation:

DMI ADX TREND 2.0 by @TraderR0BERT, NETWORTHIE.COM //Created by @TraderR0BERT, NETWORTHIE.COM, last updated 01/26/2016 //DMI Indicator //Resolution input option for higher/lower time frames study(title="DMI ADX TREND 2.0", shorttitle="ADX TREND 2.0")

adxlen = input(14, title="ADX Smoothing")

dilen = input(14, title="DI Length")

thold = input(20, title="Threshold")

threshold = thold

//Script for Indicator dirmov(len) =>

```
    up = change(high)
    down = -change(low)
    truerange = rma(tr, len)
    plus = fixnan(100 * rma(up > down and up > 0 ? up : 0, len) / truerange)
    minus = fixnan(100 * rma(down > up and down > 0 ? down : 0, len) / truerange)
    [plus, minus]
adx(dilen, adxlen) =>
    [plus, minus] = dirmov(dilen)
    sum = plus + minus
    adx = 100 * rma(abs(plus - minus) / (sum == 0 ? 1 : sum), adxlen)
    [adx, plus, minus]
[sig, up, down] = adx(dilen, adxlen)
osob=input(40,title="Exhaustion Level for ADX, default = 40")
col = sig >= sig[1] ? green : sig <= sig[1] ? red : gray
//Plot Definitions Current Timeframe p1 = plot(sig, color=col, linewidth = 3, title="ADX")
p2 = plot(sig, color=col, style=circles, linewidth=3, title="ADX")
p3 = plot(up, color=blue, linewidth = 3, title="+DI")
p4 = plot(up, color=blue, style=circles, linewidth=3, title="+DI")
p5 = plot(down, color=fuchsia, linewidth = 3, title="-DI")
p6 = plot(down, color=fuchsia, style=circles, linewidth=3, title="-DI")
h1 = plot(threshold, color=black, linewidth =3, title="Threshold")
trender = (sig >= up or sig >= down) ? 1 : 0
bgcolor(trender>0?black:gray, transp=85)
//Alert Function for ADX crossing Threshold
Up_Cross = crossover(up, threshold)
alertcondition(Up_Cross, title="DMI+ cross", message="DMI+ Crossing Threshold")
Down_Cross = crossover(down, threshold)
alertcondition(Down_Cross, title="DMI- cross", message="DMI- Crossing Threshold")
```

Parameters

- **high** (*pd.Series*) – Series of 'high's
- **low** (*pd.Series*) – Series of 'low's
- **close** (*pd.Series*) – Series of 'close's
- **length** (*int*) – It's period. Default: 14
- **drift** (*int*) – The difference period. Default: 1
- **offset** (*int*) – How many periods to offset the result. Default: 0

Kwargs: fillna (value, optional): pd.DataFrame.fillna(value)

fill_method (value, optional): Type of fill method

Returns adx, dmp, dmn columns.

Return type pd.DataFrame

amat (*fast=None*, *slow=None*, *mamode=None*, *lookback=None*, *offset=None*, ***kwargs*)
Indicator: Archer Moving Averages Trends (AMAT)

ao (*fast: int = None*, *slow: int = None*, *offset: int = None*, ***kwargs*) → pandas.core.series.Series
Awesome Oscillator (AO)

The Awesome Oscillator is an indicator used to measure a security's momentum. AO is generally used to affirm trends or to anticipate possible reversals.

Sources: https://www.tradingview.com/wiki/Awesome_Oscillator_(AO) https://www.ifcm.co.uk/ntx-indicators/awesome-oscillator

Calculation:

Default Inputs: fast=5, slow=34

SMA = Simple Moving Average

median = (high + low) / 2

AO = SMA(median, fast) - SMA(median, slow)

Parameters

- **high** (*pd.Series*) – Series of 'high's
- **low** (*pd.Series*) – Series of 'low's
- **fast** (*int*) – The short period. Default: 5
- **slow** (*int*) – The long period. Default: 34
- **offset** (*int*) – How many periods to offset the result. Default: 0

Kwargs: fillna (value, optional): pd.DataFrame.fillna(value)

fill_method (value, optional): Type of fill method

Returns New feature generated.

Return type pd.Series

aobv (*fast=None*, *slow=None*, *mamode=None*, *max_lookback=None*, *min_lookback=None*, *offset=None*, ***kwargs*)
Indicator: Archer On Balance Volume (AOBV)

apo (*fast=None*, *slow=None*, *offset=None*, ***kwargs*) → pandas.core.series.Series
Absolute Price Oscillator (APO)

The Absolute Price Oscillator is an indicator used to measure a security's momentum. It is simply the difference of two Exponential Moving Averages (EMA) of two different periods. Note: APO and MACD lines are equivalent.

Sources: https://www.investopedia.com/terms/p/ppo.asp

Calculation:

Default Inputs: fast=12, slow=26

EMA = Exponential Moving Average

APO = EMA(close, fast) - EMA(close, slow)

Parameters

- **close** (*pd.Series*) – Series of 'close's
- **fast** (*int*) – The short period. Default: 12
- **slow** (*int*) – The long period. Default: 26
- **offset** (*int*) – How many periods to offset the result. Default: 0

Kwargs: fillna (value, optional): pd.DataFrame.fillna(value)

fill_method (value, optional): Type of fill method

Returns New feature generated.

Return type pd.Series

aroon (*length=None*, *offset=None*, ***kwargs*)

Aroon (AROON)

Aroon attempts to identify if a security is trending and how strong.

Sources: https://www.tradingview.com/wiki/Aroon https://www.tradingtechnologies.com/help/x-study/technical-indicator-definitions/aroon-ar/

Calculation:

Default Inputs: length=1

def maxidx(x): return 100 * (int(np.argmax(x)) + 1) / length

def minidx(x): return 100 * (int(np.argmin(x)) + 1) / length

_close = close.rolling(length, min_periods=min_periods)

aroon_up = _close.apply(maxidx, raw=True)

aroon_down = _close.apply(minidx, raw=True)

Parameters

- **close** (*pd.Series*) – Series of 'close's
- **length** (*int*) – It's period. Default: 1
- **offset** (*int*) – How many periods to offset the result. Default: 0

Kwargs: fillna (value, optional): pd.DataFrame.fillna(value)

fill_method (value, optional): Type of fill method

Returns aroon_up, aroon_down columns.

Return type pd.DataFrame

atr (*length=None*, *mamode=None*, *offset=None*, ***kwargs*) → pandas.core.series.Series
Average True Range (ATR)

Averge True Range is used to measure volatility, especially volatility caused by gaps or limit moves.

Sources: https://www.tradingview.com/wiki/Average_True_Range_(ATR)

Calculation:

Default Inputs: length=14, drift=1

SMA = Simple Moving Average

EMA = Exponential Moving Average

TR = True Range

tr = TR(high, low, close, drift)

if 'ema': ATR = EMA(tr, length)

else: ATR = SMA(tr, length)

Parameters

- **high** (*pd.Series*) – Series of 'high's
- **low** (*pd.Series*) – Series of 'low's
- **close** (*pd.Series*) – Series of 'close's
- **length** (*int*) – It's period. Default: 14
- **mamode** (*str*) – Two options: None or 'ema'. Default: 'ema'
- **drift** (*int*) – The difference period. Default: 1
- **offset** (*int*) – How many periods to offset the result. Default: 0

Kwargs: fillna (value, optional): pd.DataFrame.fillna(value)

fill_method (value, optional): Type of fill method

Returns New feature generated.

Return type pd.Series

bbands (*length=None*, *std=None*, *mamode=None*, *offset=None*, ***kwargs*)
Bollinger Bands (BBANDS)

A popular volatility indicator.

Sources: https://www.tradingview.com/wiki/Bollinger_Bands_(BB)

Calculation:

Default Inputs: length=20, std=2

EMA = Exponential Moving Average

SMA = Simple Moving Average

STDEV = Standard Deviation

stdev = STDEV(close, length)

if 'ema': MID = EMA(close, length)

else: MID = SMA(close, length)

LOWER = MID - std * stdev

UPPER = MID + std * stdev

Parameters

- **close** (*pd.Series*) – Series of 'close's
- **length** (*int*) – The short period. Default: 20
- **std** (*int*) – The long period. Default: 2
- **mamode** (*str*) – Two options: None or 'ema'. Default: 'ema'
- **offset** (*int*) – How many periods to offset the result. Default: 0

Kwargs: fillna (value, optional): pd.DataFrame.fillna(value)

fill_method (value, optional): Type of fill method

Returns lower, mid, upper columns.

Return type pd.DataFrame

bop (*offset=None*, ***kwargs*) → pandas.core.series.Series
Balance of Power (BOP)

Balance of Power measure the market strength of buyers against sellers.

Sources: http://www.worden.com/TeleChartHelp/Content/Indicators/Balance_of_Power.htm

Calculation: BOP = (close - open) / (high - low)

Parameters

- **open** (*pd.Series*) – Series of 'open's
- **high** (*pd.Series*) – Series of 'high's
- **low** (*pd.Series*) – Series of 'low's
- **close** (*pd.Series*) – Series of 'close's
- **offset** (*int*) – How many periods to offset the result. Default: 0

Kwargs: fillna (value, optional): pd.DataFrame.fillna(value) fill_method (value, optional): Type of fill method

Returns New feature generated.

Return type pd.Series

cci (*length=None*, *c=None*, *offset=None*, ***kwargs*) → pandas.core.series.Series
Commodity Channel Index (CCI)

Commodity Channel Index is a momentum oscillator used to primarily identify overbought and oversold levels relative to a mean.

Sources: https://www.tradingview.com/wiki/Commodity_Channel_Index_(CCI)

Calculation:

Default Inputs: length=20, c=0.015

SMA = Simple Moving Average

MAD = Mean Absolute Deviation

tp = typical_price = hlc3 = (high + low + close) / 3

mean_tp = SMA(tp, length)

mad_tp = MAD(tp, length)

CCI = (tp - mean_tp) / (c * mad_tp)

Parameters

- **high** (*pd.Series*) – Series of 'high's
- **low** (*pd.Series*) – Series of 'low's
- **close** (*pd.Series*) – Series of 'close's
- **length** (*int*) – It's period. Default: 20
- **c** (*float*) – Scaling Constant. Default: 0.015
- **offset** (*int*) – How many periods to offset the result. Default: 0

Kwargs: fillna (value, optional): pd.DataFrame.fillna(value)

fill_method (value, optional): Type of fill method

Returns New feature generated.

Return type pd.Series

cg(*length=None*, *offset=None*, ***kwargs*) → pandas.core.series.Series
Center of Gravity (CG)

The Center of Gravity Indicator by John Ehlers attempts to identify turning points while exhibiting zero lag and smoothing.

Sources: http://www.mesasoftware.com/papers/TheCGOscillator.pdf

Calculation:

Default Inputs: length=10

Parameters

- **close** (*pd.Series*) – Series of 'close's
- **length** (*int*) – The length of the period. Default: 10
- **offset** (*int*) – How many periods to offset the result. Default: 0

Kwargs: fillna (value, optional): pd.DataFrame.fillna(value)

fill_method (value, optional): Type of fill method

Returns New feature generated.

Return type pd.Series

cmf (*length=None*, *offset=None*, ***kwargs*)
Chaikin Money Flow (CMF)

Chailin Money Flow measures the amount of money flow volume over a specific period in conjunction with Accumulation/Distribution.

Sources: https://www.tradingview.com/wiki/Chaikin_Money_Flow_(CMF) https://stockcharts.com/school/doku.php?id=chart_school:technical_indicators:chaikin_money_flow_cmf

Calculation:

Default Inputs: length=20

if 'open': ad = close - open

else: ad = 2 * close - high - low

hl_range = high - low

ad = ad * volume / hl_range

CMF = SUM(ad, length) / SUM(volume, length)

Parameters

- **high** (*pd.Series*) – Series of 'high's
- **low** (*pd.Series*) – Series of 'low's
- **close** (*pd.Series*) – Series of 'close's
- **open** (*pd.Series*) – Series of 'open's
- **volume** (*pd.Series*) – Series of 'volume's
- **length** (*int*) – The short period. Default: 20
- **offset** (*int*) – How many periods to offset the result. Default: 0

Kwargs: fillna (value, optional): pd.DataFrame.fillna(value)

fill_method (value, optional): Type of fill method

Returns New feature generated.

Return type pd.Series

cmo (*length=None*, *drift=None*, *offset=None*, ***kwargs*) → pandas.core.series.Series
Chande Momentum Oscillator (CMO)

Attempts to capture the momentum of an asset with overbought at 50 and oversold at -50.

Sources: https://www.tradingtechnologies.com/help/x-study/technical-indicator-definitions/chande-momentum-oscillator-cmo/

Calculation:

Default Inputs: drift=1

if close.diff(drift) > 0: PSUM = SUM(close - prev_close)

else: NSUM = ABS(SUM(close - prev_close))

CMO = 100 * (PSUM - NSUM) / (PSUM + NSUM)

Parameters

- **close** (*pd.Series*) – Series of 'close's
- **length** (*int*) – The length of the period. Default: 14
- **drift** (*int*) – The short period. Default: 1
- **offset** (*int*) – How many periods to offset the result. Default: 0

Kwargs: fillna (value, optional): pd.DataFrame.fillna(value)

fill_method (value, optional): Type of fill method

Returns New feature generated.

Return type pd.Series

coppock(*length=None*, *fast=None*, *slow=None*, *offset=None*, ***kwargs*) → pandas.core.series.Series
Coppock Curve (COPC)

Coppock Curve (originally called the "Trendex Model") is a momentum indicator is designed for use on a monthly time scale. Although designed for monthly use, a daily calculation over the same period can be made, converting the periods to 294-day and 231-day rate of changes, and a 210-day weighted moving average.

Sources: https://en.wikipedia.org/wiki/Coppock_curve

Calculation:

Default Inputs: length=10, fast=11, slow=14

SMA = Simple Moving Average

MAD = Mean Absolute Deviation

tp = typical_price = hlc3 = (high + low + close) / 3

mean_tp = SMA(tp, length)

mad_tp = MAD(tp, length)

CCI = (tp - mean_tp) / (c * mad_tp)

Parameters

- **close** (*pd.Series*) – Series of 'close's
- **length** (*int*) – WMA period. Default: 10
- **fast** (*int*) – Fast ROC period. Default: 11
- **slow** (*int*) – Slow ROC period. Default: 14
- **offset** (*int*) – How many periods to offset the result. Default: 0

Kwargs: fillna (value, optional): pd.DataFrame.fillna(value)

fill_method (value, optional): Type of fill method

Returns New feature generated.

Return type pd.Series

decreasing(*length=None*, *asint=True*, *offset=None*, ***kwargs*)
Decreasing

Returns True or False if the series is decreasing over a periods. By default, it returns True and False as 1 and 0 respectively with kwarg 'asint'.

Sources:

Calculation: decreasing = close.diff(length) < 0

if asint: decreasing = decreasing.astype(int)

Parameters

- **close** (*pd.Series*) – Series of 'close's
- **length** (*int*) – It's period. Default: 1
- **asint** (*bool*) – Returns as binary. Default: True
- **offset** (*int*) – How many periods to offset the result. Default: 0

Kwargs: fillna (value, optional): pd.DataFrame.fillna(value)

fill_method (value, optional): Type of fill method

Returns New feature generated.

Return type pd.Series

dema(*length=None*, *offset=None*, ***kwargs*)
Double Exponential Moving Average (DEMA)

The Double Exponential Moving Average attempts to a smoother average with less lag than the normal Exponential Moving Average (EMA).

Sources: https://www.tradingtechnologies.com/help/x-study/technical-indicator-definitions/double-exponential-moving-average-dema/

Calculation:

Default Inputs: length=10

EMA = Exponential Moving Average

ema1 = EMA(close, length)

ema2 = EMA(ema1, length)

DEMA = 2 * ema1 - ema2

Parameters

- **close** (*pd.Series*) – Series of 'close's
- **length** (*int*) – It's period. Default: 10
- **offset** (*int*) – How many periods to offset the result. Default: 0

Kwargs: fillna (value, optional): pd.DataFrame.fillna(value)

fill_method (value, optional): Type of fill method

Returns New feature generated.

Return type pd.Series

donchian(*lower_length=None*, *upper_length=None*, *offset=None*, ***kwargs*)
Donchian Channels (DC)

Donchian Channels are used to measure volatility, similar to Bollinger Bands and Keltner Channels.

Sources: https://www.tradingview.com/wiki/Donchian_Channels_(DC)

Calculation:

Default Inputs: length=20

LOWER = close.rolling(length).min()

UPPER = close.rolling(length).max()

MID = 0.5 * (LOWER + UPPER)

Parameters

- **close** (*pd.Series*) – Series of 'close's
- **lower_length** (*int*) – The short period. Default: 10
- **upper_length** (*int*) – The long period. Default: 20
- **offset** (*int*) – How many periods to offset the result. Default: 0

Kwargs: fillna (value, optional): pd.DataFrame.fillna(value)

fill_method (value, optional): Type of fill method

Returns lower, mid, upper columns.

Return type pd.DataFrame

dpo(*length=None*, *centered=True*, *offset=None*, ***kwargs*)
Detrend Price Oscillator (DPO)

Is an indicator designed to remove trend from price and make it easier to identify cycles.

Sources: http://stockcharts.com/school/doku.php?id=chart_school:technical_indicators:detrended_price_osci

Calculation:

Default Inputs: length=1, centered=True

SMA = Simple Moving Average

drift = int(0.5 * length) + 1

DPO = close.shift(drift) - SMA(close, length)

if centered: DPO = DPO.shift(-drift)

Parameters

- **close** (*pd.Series*) – Series of 'close's
- **length** (*int*) – It's period. Default: 1
- **centered** (*bool*) – Shift the dpo back by int(0.5 * length) + 1. Default: True
- **offset** (*int*) – How many periods to offset the result. Default: 0

Kwargs: fillna (value, optional): pd.DataFrame.fillna(value)

fill_method (value, optional): Type of fill method

Returns New feature generated.

Return type pd.Series

efi (*length=None*, *drift=None*, *mamode=None*, *offset=None*, ***kwargs*)
Elder's Force Index (EFI)

Elder's Force Index measures the power behind a price movement using price and volume as well as potential reversals and price corrections.

Sources: https://www.tradingview.com/wiki/Elder%27s_Force_Index_(EFI) https://www.motivewave.com/studies/elders_force_index.htm

Calculation:

Default Inputs: length=20, drift=1, mamode=None

EMA = Exponential Moving Average

SMA = Simple Moving Average

pv_diff = close.diff(drift) * volume if mamode == 'sma':

EFI = SMA(pv_diff, length)

else: EFI = EMA(pv_diff, length)

Parameters

- **close** (*pd.Series*) – Series of 'close's
- **volume** (*pd.Series*) – Series of 'volume's
- **length** (*int*) – The short period. Default: 13
- **drift** (*int*) – The diff period. Default: 1
- **mamode** (*str*) – Two options: None or 'sma'. Default: None
- **offset** (*int*) – How many periods to offset the result. Default: 0

Kwargs: fillna (value, optional): pd.DataFrame.fillna(value)

fill_method (value, optional): Type of fill method

Returns New feature generated.

Return type pd.Series

ema (*length=None*, *offset=None*, ***kwargs*) → pandas.core.series.Series
Exponential Moving Average (EMA)

The Exponential Moving Average is more responsive moving average compared to the Simple Moving Average (SMA). The weights are determined by alpha which is proportional to it's length. There are several different methods of calculating EMA. One method uses just the standard definition of EMA and another uses the SMA to generate the initial value for the rest of the calculation.

Sources: https://stockcharts.com/school/doku.php?id=chart_school:technical_indicators:moving_averages https://www.investopedia.com/ask/answers/122314/what-exponential-moving-average-ema-formula-and-how-ema-calculated.asp

Calculation:

Default Inputs: length=10

SMA = Simple Moving Average if kwargs['presma']:

initial = SMA(close, length) rest = close[length:] close = initial + rest

EMA = close.ewm(span=length, adjust=adjust).mean()

Parameters

- **close** (*pd.Series*) – Series of 'close's
- **length** (*int*) – It's period. Default: 10
- **offset** (*int*) – How many periods to offset the result. Default: 0

Kwargs: adjust (bool, optional): Default: True sma (bool, optional): If True, uses SMA for initial value. fillna (value, optional): pd.DataFrame.fillna(value) fill_method (value, optional): Type of fill method

Returns New feature generated.

Return type pd.Series

eom(*length=None*, *divisor=None*, *drift=None*, *offset=None*, ***kwargs*)
Ease of Movement (EOM)

Ease of Movement is a volume based oscillator that is designed to measure the relationship between price and volume flucuating across a zero line.

Sources: https://www.tradingview.com/wiki/Ease_of_Movement_(EOM) https://www.motivewave.com/studies/ease_of_movement.htm https://stockcharts.com/school/doku.php?id=chart_school:technical_indicators:ease_of_movement_emv

Calculation:

Default Inputs: length=14, divisor=100000000, drift=1

SMA = Simple Moving Average

hl_range = high - low

distance = 0.5 * (high - high.shift(drift) + low - low.shift(drift))

box_ratio = (volume / divisor) / hl_range

eom = distance / box_ratio

EOM = SMA(eom, length)

Parameters

- **high** (*pd.Series*) – Series of 'high's
- **low** (*pd.Series*) – Series of 'low's
- **close** (*pd.Series*) – Series of 'close's
- **volume** (*pd.Series*) – Series of 'volume's
- **length** (*int*) – The short period. Default: 14
- **drift** (*int*) – The diff period. Default: 1
- **offset** (*int*) – How many periods to offset the result. Default: 0

Kwargs: fillna (value, optional): pd.DataFrame.fillna(value)

fill_method (value, optional): Type of fill method

Returns New feature generated.

Return type pd.Series

fisher (*length=None*, *offset=None*, ***kwargs*) → pandas.core.series.Series
Fisher Transform (FISHT)

Attempts to identify trend reversals.

Sources: https://tulipindicators.org/fisher

Calculation:

Default Inputs: length=5

Parameters

- **close** (*pd.Series*) – Series of 'close's
- **length** (*int*) – WMA period. Default: 5
- **offset** (*int*) – How many periods to offset the result. Default: 0

Kwargs: fillna (value, optional): pd.DataFrame.fillna(value)

fill_method (value, optional): Type of fill method

Returns New feature generated.

Return type pd.Series

fwma (*length=None*, *asc=None*, *offset=None*, ***kwargs*)
Fibonacci's Weighted Moving Average (FWMA)

Fibonacci's Weighted Moving Average is similar to a Weighted Moving Average (WMA) where the weights are based on the Fibonacci Sequence.

Source: Kevin Johnson

Calculation:

Default Inputs: length=10,

def weights(w):

def _compute(x): return np.dot(w * x)

return _compute

fibs = utils.fibonacci(length - 1)

FWMA = close.rolling(length)_.apply(weights(fibs), raw=True)

Parameters

- **close** (*pd.Series*) – Series of 'close's
- **length** (*int*) – It's period. Default: 10
- **asc** (*bool*) – Recent values weigh more. Default: True
- **offset** (*int*) – How many periods to offset the result. Default: 0

Kwargs: fillna (value, optional): pd.DataFrame.fillna(value)

fill_method (value, optional): Type of fill method

Returns New feature generated.

Return type pd.Series

hl2 (*offset=None*, ***kwargs*)
Indicator: HL2

hlc3 (*offset=None*, ***kwargs*)
Indicator: HLC3

hma (*length=None*, *offset=None*, ***kwargs*)
Hull Moving Average (HMA)

The Hull Exponential Moving Average attempts to reduce or remove lag in moving averages.

Sources: https://alanhull.com/hull-moving-average

Calculation:

Default Inputs: length=10

WMA = Weighted Moving Average

half_length = int(0.5 * length)

sqrt_length = int(math.sqrt(length))

wmaf = WMA(close, half_length)

wmas = WMA(close, length)

HMA = WMA(2 * wmaf - wmas, sqrt_length)

Parameters

- **close** (*pd.Series*) – Series of 'close's
- **length** (*int*) – It's period. Default: 10
- **offset** (*int*) – How many periods to offset the result. Default: 0

Kwargs: fillna (value, optional): pd.DataFrame.fillna(value)

fill_method (value, optional): Type of fill method

Returns New feature generated.

Return type pd.Series

ichimoku (*tenkan=None*, *kijun=None*, *senkou=None*, *offset=None*, ***kwargs*)
Ichimoku Kinkō Hyō (ichimoku)

Developed Pre WWII as a forecasting model for financial markets.

Sources: https://www.tradingtechnologies.com/help/x-study/technical-indicator-definitions/ichimoku-ich/

Calculation:

Default Inputs: tenkan=9, kijun=26, senkou=52

MIDPRICE = Midprice

TENKAN_SEN = MIDPRICE(high, low, close, length=tenkan)

KIJUN_SEN = MIDPRICE(high, low, close, length=kijun)

CHIKOU_SPAN = close.shift(-kijun)

SPAN_A = 0.5 * (TENKAN_SEN + KIJUN_SEN)

SPAN_A = SPAN_A.shift(kijun)

SPAN_B = MIDPRICE(high, low, close, length=senkou)

SPAN_B = SPAN_B.shift(kijun)

Parameters

- **high** (*pd.Series*) – Series of 'high's
- **low** (*pd.Series*) – Series of 'low's
- **close** (*pd.Series*) – Series of 'close's
- **tenkan** (*int*) – Tenkan period. Default: 9
- **kijun** (*int*) – Kijun period. Default: 26
- **senkou** (*int*) – Senkou period. Default: 52
- **offset** (*int*) – How many periods to offset the result. Default: 0

Kwargs: fillna (value, optional): pd.DataFrame.fillna(value)

fill_method (value, optional): Type of fill method

Returns

Two DataFrames.

For the visible period: spanA, spanB, tenkan_sen, kijun_sen, and chikou_span columns

For the forward looking period: spanA and spanB columns

Return type pd.DataFrame

increasing(*length=None*, *asint=True*, *offset=None*, ***kwargs*)

Increasing

Returns True or False if the series is increasing over a periods. By default, it returns True and False as 1 and 0 respectively with kwarg 'asint'.

Sources:

Calculation: increasing = close.diff(length) > 0 if asint:

increasing = increasing.astype(int)

Parameters

- **close** (*pd.Series*) – Series of 'close's
- **length** (*int*) – It's period. Default: 1
- **asint** (*bool*) – Returns as binary. Default: True

- **offset** (*int*) – How many periods to offset the result. Default: 0

Kwargs: fillna (value, optional): pd.DataFrame.fillna(value) fill_method (value, optional): Type of fill method

Returns New feature generated.

Return type pd.Series

kama(*length=None*, *fast=None*, *slow=None*, *drift=None*, *offset=None*, ***kwargs*)
Kaufman's Adaptive Moving Average (KAMA)

Developed by Perry Kaufman, Kaufman's Adaptive Moving Average (KAMA) is a moving average designed to account for market noise or volatility. KAMA will closely follow prices when the price swings are relatively small and the noise is low. KAMA will adjust when the price swings widen and follow prices from a greater distance. This trend-following indicator can be used to identify the overall trend, time turning points and filter price movements.

Sources: https://stockcharts.com/school/doku.php?id=chart_school:technical_indicators:kaufman_s_adaptive_moving_average

Calculation:

Default Inputs: length=10

Parameters

- **close** (*pd.Series*) – Series of 'close's
- **length** (*int*) – It's period. Default: 10
- **fast** (*int*) – The short period. Default: 2
- **slow** (*int*) – The long period. Default: 30
- **drift** (*int*) – The difference period. Default: 1
- **offset** (*int*) – How many periods to offset the result. Default: 0

Kwargs: fillna (value, optional): pd.DataFrame.fillna(value)

fill_method (value, optional): Type of fill method

Returns New feature generated.

Return type pd.Series

kc(*length=None*, *scalar=None*, *mamode=None*, *offset=None*, ***kwargs*)
Keltner Channels (KC)

A popular volatility indicator similar to Bollinger Bands and Donchian Channels.

Sources: https://www.tradingview.com/wiki/Keltner_Channels_(KC)

Calculation:

Default Inputs: length=20, scalar=2

ATR = Average True Range

EMA = Exponential Moving Average

SMA = Simple Moving Average

if 'ema': BASIS = EMA(close, length)

BAND = ATR(high, low, close)

else: hl_range = high - low

tp = typical_price = hlc3(high, low, close)

BASIS = SMA(tp, length)

BAND = SMA(hl_range, length)

LOWER = BASIS - scalar * BAND

UPPER = BASIS + scalar * BAND

Parameters

- **high** (*pd.Series*) – Series of 'high's
- **low** (*pd.Series*) – Series of 'low's
- **close** (*pd.Series*) – Series of 'close's
- **length** (*int*) – The short period. Default: 20
- **scalar** (*float*) – A positive float to scale the bands. Default: 2
- **mamode** (*str*) – Two options: None or 'ema'. Default: 'ema'
- **offset** (*int*) – How many periods to offset the result. Default: 0

Kwargs: fillna (value, optional): pd.DataFrame.fillna(value)

fill_method (value, optional): Type of fill method

Returns lower, basis, upper columns.

Return type pd.DataFrame

kst(*roc1=None*, *roc2=None*, *roc3=None*, *roc4=None*, *sma1=None*, *sma2=None*, *sma3=None*, *sma4=None*, *signal=None*, *drift=None*, *offset=None*, ***kwargs*) → pandas.core.series.Series
'Know Sure Thing' (KST)

The 'Know Sure Thing' is a momentum based oscillator and based on ROC.

Sources: https://www.tradingview.com/wiki/Know_Sure_Thing_(KST) https://www.incrediblecharts.com/indicators/kst.php

Calculation:

Default Inputs: roc1=10, roc2=15, roc3=20, roc4=30, sma1=10, sma2=10, sma3=10, sma4=15, signal=9, drift=1

ROC = Rate of Change

SMA = Simple Moving Average

rocsma1 = SMA(ROC(close, roc1), sma1)

rocsma2 = SMA(ROC(close, roc2), sma2)

rocsma3 = SMA(ROC(close, roc3), sma3)

rocsma4 = SMA(ROC(close, roc4), sma4)

KST = 100 * (rocsma1 + 2 * rocsma2 + 3 * rocsma3 + 4 * rocsma4)

KST_Signal = SMA(KST, signal)

Parameters

- **close** (*pd.Series*) – Series of 'close's
- **roc1** (*int*) – ROC 1 period. Default: 10
- **roc2** (*int*) – ROC 2 period. Default: 15
- **roc3** (*int*) – ROC 3 period. Default: 20
- **roc4** (*int*) – ROC 4 period. Default: 30
- **sma1** (*int*) – SMA 1 period. Default: 10
- **sma2** (*int*) – SMA 2 period. Default: 10
- **sma3** (*int*) – SMA 3 period. Default: 10
- **sma4** (*int*) – SMA 4 period. Default: 15
- **signal** (*int*) – It's period. Default: 9
- **drift** (*int*) – The difference period. Default: 1
- **offset** (*int*) – How many periods to offset the result. Default: 0

Kwargs: fillna (value, optional): pd.DataFrame.fillna(value)

fill_method (value, optional): Type of fill method

Returns kst and kst_signal columns

Return type pd.DataFrame

kurtosis(*length=None*, *offset=None*, ***kwargs*)

Rolling Kurtosis

Sources:

Calculation:

Default Inputs: length=30

KURTOSIS = close.rolling(length).kurt()

Parameters

- **close** (*pd.Series*) – Series of 'close's
- **length** (*int*) – It's period. Default: 30
- **offset** (*int*) – How many periods to offset the result. Default: 0

Kwargs: fillna (value, optional): pd.DataFrame.fillna(value)

fill_method (value, optional): Type of fill method

Returns New feature generated.

Return type pd.Series

linear_decay(*length=None*, *offset=None*, ***kwargs*)
Linear Decay

Adds a linear decay moving forward from prior signals like crosses.

Sources: https://tulipindicators.org/decay

Calculation:

Default Inputs: length=5

max(close, close[-1] - (1 / length), 0)

Parameters

- **close** (*pd.Series*) – Series of 'close's
- **length** (*int*) – It's period. Default: 1
- **offset** (*int*) – How many periods to offset the result. Default: 0

Kwargs: fillna (value, optional): pd.DataFrame.fillna(value)

fill_method (value, optional): Type of fill method

Returns New feature generated.

Return type pd.Series

linreg(*length=None*, *offset=None*, ***kwargs*)
Linear Regression Moving Average (linreg)

Linear Regression Moving Average

Source: TA Lib

Calculation:

Default Inputs: length=14

x = [1, 2, . . . , n]

x_sum = 0.5 * length * (length + 1)

x2_sum = length * (length + 1) * (2 * length + 1) / 6

divisor = length * x2_sum - x_sum * x_sum

lr(series): y_sum = series.sum()

y2_sum = (series* series).sum()

xy_sum = (x * series).sum()

m = (length * xy_sum - x_sum * y_sum) / divisor

b = (y_sum * x2_sum - x_sum * xy_sum) / divisor

return m * (length - 1) + b

linreg = close.rolling(length).apply(lr)

Parameters

- **close** (*pd.Series*) – Series of 'close's
- **length** (*int*) – It's period. Default: 10

- **offset** (*int*) – How many periods to offset the result. Default: 0

Kwargs: fillna (value, optional): pd.DataFrame.fillna(value)

fill_method (value, optional): Type of fill method

angle (bool, optional): Default: False. If True, returns the angle of the slope in radians

degrees (bool, optional): Default: False. If True, returns the angle of the slope in degrees

intercept (bool, optional): Default: False. If True, returns the angle of the slope in radians

r (bool, optional): Default: False. If True, returns it's correlation 'r'

slope (bool, optional): Default: False. If True, returns the slope

tsf (bool, optional): Default: False. If True, returns the Time Series Forecast value.

Returns New feature generated.

Return type pd.Series

log_return(*length=None*, *cumulative=False*, *percent=False*, *offset=None*, ***kwargs*) → pandas.core.series.Series

Log Return

Calculates the logarithmic return of a Series. See also: help(df.ta.log_return) for additional **kwargs a valid 'df'.

Sources: https://stackoverflow.com/questions/31287552/logarithmic-returns-in-pandas-dataframe

Calculation:

Default Inputs: length=1, cumulative=False

LOGRET = log(close.diff(periods=length))

CUMLOGRET = LOGRET.cumsum() if cumulative

Parameters

- **close** (*pd.Series*) – Series of 'close's
- **length** (*int*) – It's period. Default: 20
- **cumulative** (*bool*) – If True, returns the cumulative returns. Default: False
- **offset** (*int*) – How many periods to offset the result. Default: 0

Kwargs: fillna (value, optional): pd.DataFrame.fillna(value)

fill_method (value, optional): Type of fill method

Returns New feature generated.

Return type pd.Series

long_run(*fast: pandas.core.series.Series*, *slow: pandas.core.series.Series*, *length=None*, *offset=None*, ***kwargs*)

Indicator: Long Run

macd(*fast=None*, *slow=None*, *signal=None*, *offset=None*, ***kwargs*) → pandas.core.series.Series

Moving Average Convergence Divergence (MACD)

The MACD is a popular indicator to that is used to identify a security's trend. While APO and MACD are the same calculation, MACD also returns two more series called Signal and Histogram. The Signal is an EMA of MACD and the Histogram is the difference of MACD and Signal.

Sources: https://www.tradingview.com/wiki/MACD_(Moving_Average_Convergence/Divergence)

Calculation:

Default Inputs: fast=12, slow=26, signal=9

EMA = Exponential Moving Average

MACD = EMA(close, fast) - EMA(close, slow)

Signal = EMA(MACD, signal)

Histogram = MACD - Signal

Parameters

- **close** (*pd.Series*) – Series of 'close's
- **fast** (*int*) – The short period. Default: 12
- **slow** (*int*) – The long period. Default: 26
- **signal** (*int*) – The signal period. Default: 9
- **offset** (*int*) – How many periods to offset the result. Default: 0

Kwargs: fillna (value, optional): pd.DataFrame.fillna(value)

fill_method (value, optional): Type of fill method

Returns macd, histogram, signal columns.

Return type pd.DataFrame

mad(*length=None*, *offset=None*, ***kwargs*)
Rolling Mean Absolute Deviation

Sources:

Calculation:

Default Inputs: length=30

mad = close.rolling(length).mad()

Parameters

- **close** (*pd.Series*) – Series of 'close's
- **length** (*int*) – It's period. Default: 30
- **offset** (*int*) – How many periods to offset the result. Default: 0

Kwargs: fillna (value, optional): pd.DataFrame.fillna(value)

fill_method (value, optional): Type of fill method

Returns New feature generated.

Return type pd.Series

massi (*fast=None*, *slow=None*, *offset=None*, ***kwargs*)
Mass Index (MASSI)

The Mass Index is a non-directional volatility indicator that utilitizes the High-Low Range to identify trend reversals based on range expansions.

Sources: https://stockcharts.com/school/doku.php?id=chart_school:technical_indicators:mass_index mi = sum(ema(high - low, 9) / ema(ema(high - low, 9), 9), length)

Calculation:

Default Inputs: fast: 9, slow: 25

EMA = Exponential Moving Average

hl = high - low

hl_ema1 = EMA(hl, fast)

hl_ema2 = EMA(hl_ema1, fast)

hl_ratio = hl_ema1 / hl_ema2

MASSI = SUM(hl_ratio, slow)

Parameters

- **high** (*pd.Series*) – Series of 'high's
- **low** (*pd.Series*) – Series of 'low's
- **fast** (*int*) – The short period. Default: 9
- **slow** (*int*) – The long period. Default: 25
- **offset** (*int*) – How many periods to offset the result. Default: 0

Kwargs: fillna (value, optional): pd.DataFrame.fillna(value)

fill_method (value, optional): Type of fill method

Returns New feature generated.

Return type pd.Series

median (*length=None*, *offset=None*, ***kwargs*)
Rolling Median

Rolling Median of over 'n' periods. Sibling of a Simple Moving Average.

Sources: https://www.incrediblecharts.com/indicators/median_price.php

Calculation:

Default Inputs: length=30

MEDIAN = close.rolling(length).median()

Parameters

- **close** (*pd.Series*) – Series of 'close's
- **length** (*int*) – It's period. Default: 30
- **offset** (*int*) – How many periods to offset the result. Default: 0

Kwargs: fillna (value, optional): pd.DataFrame.fillna(value)

fill_method (value, optional): Type of fill method

Returns New feature generated.

Return type pd.Series

mfi (*length=None*, *drift=None*, *offset=None*, ***kwargs*)
Money Flow Index (MFI)

Money Flow Index is an oscillator indicator that is used to measure buying and selling pressure by utilizing both price and volume.

Sources: https://www.tradingview.com/wiki/Money_Flow_(MFI)

Calculation:

Default Inputs: length=14, drift=1

tp = typical_price = hlc3 = (high + low + close) / 3

rmf = raw_money_flow = tp * volume

pmf = pos_money_flow = SUM(rmf, length) if tp.diff(drift) > 0 else 0

nmf = neg_money_flow = SUM(rmf, length) if tp.diff(drift) < 0 else 0

MFR = money_flow_ratio = pmf / nmf

MFI = money_flow_index = 100 * pmf / (pmf + nmf)

Parameters

- **high** (*pd.Series*) – Series of 'high's
- **low** (*pd.Series*) – Series of 'low's
- **close** (*pd.Series*) – Series of 'close's
- **volume** (*pd.Series*) – Series of 'volume's
- **length** (*int*) – The sum period. Default: 14
- **drift** (*int*) – The difference period. Default: 1
- **offset** (*int*) – How many periods to offset the result. Default: 0

Kwargs: fillna (value, optional): pd.DataFrame.fillna(value)

fill_method (value, optional): Type of fill method

Returns New feature generated.

Return type pd.Series

midpoint (*length=None*, *offset=None*, ***kwargs*)
Indicator: Midpoint

midprice (*length=None*, *offset=None*, ***kwargs*)
Indicator: Midprice

mom(*length=None*, *offset=None*, ***kwargs*) → pandas.core.series.Series
Momentum (MOM)

Momentum is an indicator used to measure a security's speed (or strength) of movement. Or simply the change in price.

Sources: http://www.onlinetradingconcepts.com/TechnicalAnalysis/Momentum.html

Calculation:

Default Inputs: length=1

MOM = close.diff(length)

Parameters

- **close** (*pd.Series*) – Series of 'close's
- **length** (*int*) – It's period. Default: 1
- **offset** (*int*) – How many periods to offset the result. Default: 0

Kwargs: fillna (value, optional): pd.DataFrame.fillna(value)

fill_method (value, optional): Type of fill method

Returns New feature generated.

Return type pd.Series

natr(*length=None*, *mamode=None*, *drift=None*, *offset=None*, ***kwargs*)
Normalized Average True Range (NATR)

Normalized Average True Range attempt to normalize the average true range.

Sources: https://www.tradingtechnologies.com/help/x-study/technical-indicator-definitions/normalized-average-true-range-natr/

Calculation:

Default Inputs: length=20

ATR = Average True Range

NATR = (100 / close) * ATR(high, low, close)

Parameters

- **high** (*pd.Series*) – Series of 'high's
- **low** (*pd.Series*) – Series of 'low's
- **close** (*pd.Series*) – Series of 'close's
- **length** (*int*) – The short period. Default: 20
- **offset** (*int*) – How many periods to offset the result. Default: 0

Kwargs: fillna (value, optional): pd.DataFrame.fillna(value)

fill_method (value, optional): Type of fill method

Returns New feature

Return type pd.Series

nvi (*length=None*, *initial=None*, *offset=None*, ***kwargs*)
Negative Volume Index (NVI)

The Negative Volume Index is a cumulative indicator that uses volume change in an attempt to identify where smart money is active.

Sources: https://stockcharts.com/school/doku.php?id=chart_school:technical_indicators:negative_volume_inde https://www.motivewave.com/studies/negative_volume_index.htm

Calculation:

Default Inputs: length=1, initial=1000

ROC = Rate of Change

roc = ROC(close, length)

signed_volume = signed_series(volume, initial=1)

nvi = signed_volume[signed_volume < 0].abs() * roc_

nvi.fillna(0, inplace=True)

nvi.iloc[0]= initial

nvi = nvi.cumsum()

Parameters

- **close** (*pd.Series*) – Series of 'close's
- **volume** (*pd.Series*) – Series of 'volume's
- **length** (*int*) – The short period. Default: 13
- **initial** (*int*) – The short period. Default: 1000
- **offset** (*int*) – How many periods to offset the result. Default: 0

Kwargs: fillna (value, optional): pd.DataFrame.fillna(value)

fill_method (value, optional): Type of fill method

Returns New feature generated.

Return type pd.Series

obv (*offset=None*, ***kwargs*)
On Balance Volume (OBV)

On Balance Volume is a cumulative indicator to measure buying and selling pressure.

Sources: https://www.tradingview.com/wiki/On_Balance_Volume_(OBV) https://www.tradingtechnologies.com/help/x-study/technical-indicator-definitions/on-balance-volume-obv/ https://www.motivewave.com/studies/on_balance_volume.htm

Calculation: signed_volume = signed_series(close, initial=1) * volume

obv = signed_volume.cumsum()

Parameters

- **close** (*pd.Series*) – Series of 'close's
- **volume** (*pd.Series*) – Series of 'volume's

- **offset** (*int*) – How many periods to offset the result. Default: 0

Kwargs: fillna (value, optional): pd.DataFrame.fillna(value)

fill_method (value, optional): Type of fill method

Returns New feature generated.

Return type pd.Series

ohlc4 (*offset=None*, ***kwargs*)
Indicator: OHLC4

percent_return (*length=None*, *cumulative=False*, *percent=False*, *offset=None*, ***kwargs*) → pandas.core.series.Series
Percent Return

Calculates the percent return of a Series. See also: help(df.ta.percent_return) for additional **kwargs a valid 'df'.

Sources: https://stackoverflow.com/questions/31287552/logarithmic-returns-in-pandas-dataframe

Calculation:

Default Inputs: length=1, cumulative=False

PCTRET = close.pct_change(length)

CUMPCTRET = PCTRET.cumsum() if cumulative

Parameters

- **close** (*pd.Series*) – Series of 'close's
- **length** (*int*) – It's period. Default: 20
- **cumulative** (*bool*) – If True, returns the cumulative returns. Default: False
- **offset** (*int*) – How many periods to offset the result. Default: 0

Kwargs: fillna (value, optional): pd.DataFrame.fillna(value)

fill_method (value, optional): Type of fill method

Returns New feature generated.

Return type pd.Series

ppo (*fast=None*, *slow=None*, *signal=None*, *offset=None*, ***kwargs*)
Percentage Price Oscillator (PPO)

The Percentage Price Oscillator is similar to MACD in measuring momentum.

Sources: https://www.tradingview.com/wiki/MACD_(Moving_Average_Convergence/Divergence)

Calculation:

Default Inputs: fast=12, slow=26

SMA = Simple Moving Average

EMA = Exponential Moving Average

fast_sma = SMA(close, fast)

slow_sma = SMA(close, slow)

PPO = 100 * (fast_sma - slow_sma) / slow_sma

Signal = EMA(PPO, signal)

Histogram = PPO - Signal

Parameters

- **close** (*pandas.Series*) – Series of 'close's
- **fast** (*int*) – The short period. Default: 12
- **slow** (*int*) – The long period. Default: 26
- **signal** (*int*) – The signal period. Default: 9
- **offset** (*int*) – How many periods to offset the result. Default: 0

Kwargs: fillna (value, optional): pd.DataFrame.fillna(value)

fill_method (value, optional): Type of fill method

Returns ppo, histogram, signal columns

Return type pd.DataFrame

pvi (*length=None*, *initial=None*, *offset=None*, ***kwargs*)
Positive Volume Index (PVI)

The Positive Volume Index is a cumulative indicator that uses volume change in an attempt to identify where smart money is active. Used in conjunction with NVI.

Sources: https://www.investopedia.com/terms/p/pvi.asp

Calculation:

Default Inputs: length=1, initial=1000

ROC = Rate of Change

roc = ROC(close, length)

signed_volume = signed_series(volume, initial=1)

pvi = signed_volume[signed_volume > 0].abs() * **roc_**

pvi.fillna(0, inplace=True)

pvi.iloc[0]= initial

pvi = pvi.cumsum()

Parameters

- **close** (*pd.Series*) – Series of 'close's
- **volume** (*pd.Series*) – Series of 'volume's
- **length** (*int*) – The short period. Default: 13
- **initial** (*int*) – The short period. Default: 1000
- **offset** (*int*) – How many periods to offset the result. Default: 0

Kwargs: fillna (value, optional): pd.DataFrame.fillna(value)

fill_method (value, optional): Type of fill method

Returns New feature generated.

Return type pd.Series

pvol(*offset=None*, ***kwargs*)
Price-Volume (PVOL)

Returns a series of the product of price and volume.

Calculation:

if signed: pvol = signed_series(close, 1) * close * volume

else: pvol = close * volume

Parameters

- **close** (*pd.Series*) – Series of 'close's
- **volume** (*pd.Series*) – Series of 'volume's
- **signed** (*bool*) – Keeps the sign of the difference in 'close's. Default: True
- **offset** (*int*) – How many periods to offset the result. Default: 0

Kwargs: fillna (value, optional): pd.DataFrame.fillna(value)

fill_method (value, optional): Type of fill method

Returns New feature generated.

Return type pd.Series

pvt(*drift=None*, *offset=None*, ***kwargs*)
Price-Volume Trend (PVT)

The Price-Volume Trend utilizes the Rate of Change with volume to and it's cumulative values to determine money flow.

Sources: https://www.tradingview.com/wiki/Price_Volume_Trend_(PVT)

Calculation:

Default Inputs: drift=1

ROC = Rate of Change

pv = ROC(close, drift) * volume

PVT = pv.cumsum()

Parameters

- **close** (*pd.Series*) – Series of 'close's
- **volume** (*pd.Series*) – Series of 'volume's
- **drift** (*int*) – The diff period. Default: 1
- **offset** (*int*) – How many periods to offset the result. Default: 0

Kwargs: fillna (value, optional): pd.DataFrame.fillna(value)

fill_method (value, optional): Type of fill method

Returns New feature generated.

Return type pd.Series

pwma (*length=None*, *asc=None*, *offset=None*, ***kwargs*)
Pascal's Weighted Moving Average (PWMA)

Pascal's Weighted Moving Average is similar to a symmetric triangular window except PWMA's weights are based on Pascal's Triangle.

Source: Kevin Johnson

Calculation:

Default Inputs: length=10

def weights(w):

def _compute(x): return np.dot(w * x)

return _compute

triangle = utils.pascals_triangle(length + 1)

PWMA = close.rolling(length)_.apply(weights(triangle), raw=True)

Parameters

- **close** (*pd.Series*) – Series of 'close's
- **length** (*int*) – It's period. Default: 10
- **asc** (*bool*) – Recent values weigh more. Default: True
- **offset** (*int*) – How many periods to offset the result. Default: 0

Kwargs: fillna (value, optional): pd.DataFrame.fillna(value)

fill_method (value, optional): Type of fill method

Returns New feature generated.

Return type pd.Series

qstick (*length=None*, *offset=None*, ***kwargs*)
Q Stick

The Q Stick indicator, developed by Tushar Chande, attempts to quantify and identify trends in candlestick charts.

Sources: https://library.tradingtechnologies.com/trade/chrt-ti-qstick.html

Calculation:

Default Inputs: length=10

xMA is one of: sma (default), dema, ema, hma, rma

qstick = xMA(close - open, length)

Parameters

- **open** (*pd.Series*) – Series of 'open's
- **close** (*pd.Series*) – Series of 'close's

- **length** (*int*) – It's period. Default: 1
- **ma** (*str*) – The type of moving average to use. Default: None, which is 'sma'
- **offset** (*int*) – How many periods to offset the result. Default: 0

Kwargs: fillna (value, optional): pd.DataFrame.fillna(value)

fill_method (value, optional): Type of fill method

Returns New feature generated.

Return type pd.Series

quantile(*length=None*, *q=None*, *offset=None*, ***kwargs*)
Rolling Quantile

Sources:

Calculation:

Default Inputs: length=30, q=0.5

QUANTILE = close.rolling(length).quantile(q)

Parameters

- **close** (*pd.Series*) – Series of 'close's
- **length** (*int*) – It's period. Default: 30
- **q** (*float*) – The quantile. Default: 0.5
- **offset** (*int*) – How many periods to offset the result. Default: 0

Kwargs: fillna (value, optional): pd.DataFrame.fillna(value)

fill_method (value, optional): Type of fill method

Returns New feature generated.

Return type pd.Series

rma(*length=None*, *offset=None*, ***kwargs*)
wildeR's Moving Average (RMA)

The WildeR's Moving Average is simply an Exponential Moving Average (EMA) with a modified alpha = 1 / length.

Sources: https://alanhull.com/hull-moving-average

Calculation:

Default Inputs: length=10

EMA = Exponential Moving Average

alpha = 1 / length

RMA = EMA(close, alpha=alpha)

Parameters

- **close** (*pd.Series*) – Series of 'close's

- **length** (*int*) – It's period. Default: 10
- **offset** (*int*) – How many periods to offset the result. Default: 0

Kwargs: fillna (value, optional): pd.DataFrame.fillna(value)

fill_method (value, optional): Type of fill method

Returns New feature generated.

Return type pd.Series

roc(*length=None*, *offset=None*, ***kwargs*) → pandas.core.series.Series
Rate of Change (ROC)

Rate of Change is an indicator is also referred to as Momentum (yeah, confusingly). It is a pure momentum oscillator that measures the percent change in price with the previous price 'n' (or length) periods ago.

Sources: https://www.tradingview.com/wiki/Rate_of_Change_(ROC)

Calculation:

Default Inputs: length=1

MOM = Momentum

ROC = 100 * MOM(close, length) / close.shift(length)

Parameters

- **close** (*pd.Series*) – Series of 'close's
- **length** (*int*) – It's period. Default: 1
- **offset** (*int*) – How many periods to offset the result. Default: 0

Kwargs: fillna (value, optional): pd.DataFrame.fillna(value)

fill_method (value, optional): Type of fill method

Returns New feature generated.

Return type pd.Series

rsi(*length=None*, *drift=None*, *offset=None*, ***kwargs*) → pandas.core.series.Series
Relative Strength Index (RSI)

The Relative Strength Index is popular momentum oscillator used to measure the velocity as well as the magnitude of directional price movements.

Sources: https://www.tradingview.com/wiki/Relative_Strength_Index_(RSI)

Calculation:

Default Inputs: length=14, drift=1

ABS = Absolute Value

EMA = Exponential Moving Average

positive = close if close.diff(drift) > 0 else 0

negative = close if close.diff(drift) < 0 else 0

pos_avg = EMA(positive, length)

neg_avg = ABS(EMA(negative, length))

RSI = 100 * pos_avg / (pos_avg + neg_avg)

Parameters

- **close** (*pd.Series*) – Series of 'close's
- **length** (*int*) – It's period. Default: 1
- **drift** (*int*) – The difference period. Default: 1
- **offset** (*int*) – How many periods to offset the result. Default: 0

Kwargs: fillna (value, optional): pd.DataFrame.fillna(value)

fill_method (value, optional): Type of fill method

Returns New feature generated.

Return type pd.Series

rvi (*length=None*, *swma_length=None*, *offset=None*, ***kwargs*)
Relative Vigor Index (RVI)

The Relative Vigor Index attempts to measure the strength of a trend relative to its closing price to its trading range. It is based on the belief that it tends to close higher than they open in uptrends or close lower than they open in downtrends.

Sources: https://www.investopedia.com/terms/r/relative_vigor_index.asp

Calculation:

Default Inputs: length=14, swma_length=4

SWMA = Symmetrically Weighted Moving Average

numerator = SUM(SWMA(close - open, swma_length), length)

denominator = SUM(SWMA(high - low, swma_length), length)

RVI = numerator / denominator

Parameters

- **open** (*pd.Series*) – Series of 'open's
- **high** (*pd.Series*) – Series of 'high's
- **low** (*pd.Series*) – Series of 'low's
- **close** (*pd.Series*) – Series of 'close's
- **length** (*int*) – It's period. Default: 14
- **swma_length** (*int*) – It's period. Default: 4
- **offset** (*int*) – How many periods to offset the result. Default: 0

Kwargs: fillna (value, optional): pd.DataFrame.fillna(value)

fill_method (value, optional): Type of fill method

Returns New feature generated.

Return type pd.Series

short_run (*fast: pandas.core.series.Series*, *slow: pandas.core.series.Series*, *length=None*, *offset=None*, ***kwargs*)
Indicator: Short Run

sinwma (*length=None*, *offset=None*, ***kwargs*)
Sine Weighted Moving Average (SWMA)

A weighted average using sine cycles. The middle term(s) of the average have the highest weight(s).

Source: https://www.tradingview.com/script/6MWFvnPO-Sine-Weighted-Moving-Average/ Author: Everget (https://www.tradingview.com/u/everget/)

Calculation:

Default Inputs: length=10

def weights(w):

def _compute(x): return np.dot(w * x)

return _compute

sines = Series([sin((i + 1) * pi / (length + 1)) for i in range(0, length)])

w = sines / sines.sum()

SINWMA = close.rolling(length, min_periods=length).apply(weights(w), raw=True)

Parameters

- **close** (*pd.Series*) – Series of 'close's
- **length** (*int*) – It's period. Default: 10
- **offset** (*int*) – How many periods to offset the result. Default: 0

Kwargs: fillna (value, optional): pd.DataFrame.fillna(value)

fill_method (value, optional): Type of fill method

Returns New feature generated.

Return type pd.Series

skew (*length=None*, *offset=None*, ***kwargs*)
Rolling Skew

Sources:

Calculation:

Default Inputs: length=30

SKEW = close.rolling(length).skew()

Parameters

- **close** (*pd.Series*) – Series of 'close's
- **length** (*int*) – It's period. Default: 30
- **offset** (*int*) – How many periods to offset the result. Default: 0

Kwargs: fillna (value, optional): pd.DataFrame.fillna(value)

fill_method (value, optional): Type of fill method

Returns New feature generated.

Return type pd.Series

slope(*length=None*, *as_angle=None*, *to_degrees=None*, *offset=None*, ***kwargs*)
Slope

Returns the slope of a series of length n. Can convert the slope to angle, default as radians or degrees.

Sources: Algebra I

Calculation:

Default Inputs: length=1

slope = close.diff(length) / length

if as_angle: slope = slope.apply(atan) if to_degrees:

slope *= 180 / PI

Parameters

- **close** (*pd.Series*) – Series of 'close's
- **length** (*int*) – It's period. Default: 1
- **offset** (*int*) – How many periods to offset the result. Default: 0

Kwargs: as_angle (value, optional): Converts slope to an angle. Default: False

to_degrees (value, optional): Converts slope angle to degrees. Default: False

fillna (value, optional): pd.DataFrame.fillna(value)

fill_method (value, optional): Type of fill method

Returns New feature generated.

Return type pd.Series

sma(*length=None*, *offset=None*, ***kwargs*) → pandas.core.series.Series
Simple Moving Average (SMA)

The Simple Moving Average is the classic moving average that is the equally weighted average over n periods.

Sources: https://www.tradingtechnologies.com/help/x-study/technical-indicator-definitions/simple-moving-average-sma/

Calculation:

Default Inputs: length=10

SMA = SUM(close, length) / length

Parameters

- **close** (*pd.Series*) – Series of 'close's
- **length** (*int*) – It's period. Default: 10
- **offset** (*int*) – How many periods to offset the result. Default: 0

Kwargs: adjust (bool): Default: True

presma (bool, optional): If True, uses SMA for initial value.

fillna (value, optional): pd.DataFrame.fillna(value)

fill_method (value, optional): Type of fill method

Returns New feature generated.

Return type pd.Series

stdev (*length=None*, *offset=None*, ***kwargs*)
Rolling Standard Deviation

Sources:

Calculation:

Default Inputs: length=30

VAR = Variance

STDEV = variance(close, length).apply(np.sqrt)

Parameters

- **close** (*pd.Series*) – Series of 'close's
- **length** (*int*) – It's period. Default: 30
- **offset** (*int*) – How many periods to offset the result. Default: 0

Kwargs: fillna (value, optional): pd.DataFrame.fillna(value) fill_method (value, optional): Type of fill method

Returns New feature generated.

Return type pd.Series

stoch (*fast_k=None*, *slow_k=None*, *slow_d=None*, *offset=None*, ***kwargs*)
Stochastic (STOCH)

Stochastic Oscillator is a range bound momentum indicator. It displays the location of the close relative to the high-low range over a period.

Sources: https://www.tradingview.com/wiki/Stochastic_(STOCH)

Calculation:

Default Inputs: fast_k=14, slow_k=5, slow_d=3

SMA = Simple Moving Average

lowest_low = low for last fast_k periods

highest_high = high for last fast_k periods

FASTK = 100 * (close - lowest_low) / (highest_high - lowest_low)

FASTD = SMA(FASTK, slow_d)

SLOWK = SMA(FASTK, slow_k)

SLOWD = SMA(SLOWK, slow_d)

Parameters

- **high** (*pd.Series*) – Series of 'high's
- **low** (*pd.Series*) – Series of 'low's
- **close** (*pd.Series*) – Series of 'close's
- **fast_k** (*int*) – The Fast %K period. Default: 14
- **slow_k** (*int*) – The Slow %K period. Default: 5
- **slow_d** (*int*) – The Slow %D period. Default: 3
- **offset** (*int*) – How many periods to offset the result. Default: 0

Kwargs: fillna (value, optional): pd.DataFrame.fillna(value)

fill_method (value, optional): Type of fill method

Returns fastk, fastd, slowk, slowd columns.

Return type pd.DataFrame

swma (*length=None*, *asc=None*, *offset=None*, ***kwargs*)
Symmetric Weighted Moving Average (SWMA)

Symmetric Weighted Moving Average where weights are based on a symmetric triangle. For example: n=3 -> [1, 2, 1], n=4 -> [1, 2, 2, 1], etc... This moving average has variable length in contrast to TradingView's fixed length of 4.

Source: https://www.tradingview.com/study-script-reference/#fun_swma

Calculation:

Default Inputs: length=10

def weights(w):

def _compute(x): return np.dot(w * x)

return _compute

triangle = utils.symmetric_triangle(length - 1) SWMA = close.rolling(length)_.apply(weights(triangle), raw=True)

Parameters

- **close** (*pd.Series*) – Series of 'close's
- **length** (*int*) – It's period. Default: 10
- **asc** (*bool*) – Recent values weigh more. Default: True
- **offset** (*int*) – How many periods to offset the result. Default: 0

Kwargs: fillna (value, optional): pd.DataFrame.fillna(value) fill_method (value, optional): Type of fill method

Returns New feature generated.

Return type pd.Series

t3 (*length=None*, *a=None*, *offset=None*, ***kwargs*)
Tim Tillson's T3 Moving Average (T3)

Tim Tillson's T3 Moving Average is considered a smoother and more responsive moving average relative to other moving averages.

Sources: http://www.binarytribune.com/forex-trading-indicators/t3-moving-average-indicator/

Calculation:

Default Inputs: length=10, a=0.7

c1 = -a^3 c2 = 3a^2 + 3a^3 = 3a^2 * (1 + a) c3 = -6a^2 - 3a - 3a^3 c4 = a^3 + 3a^2 + 3a + 1

ema1 = EMA(close, length) ema2 = EMA(ema1, length) ema3 = EMA(ema2, length) ema4 = EMA(ema3, length) ema5 = EMA(ema4, length) ema6 = EMA(ema5, length) T3 = c1 * ema6 + c2 * ema5 + c3 * ema4 + c4 * ema3

Parameters

- **close** (*pd.Series*) – Series of 'close's
- **length** (*int*) – It's period. Default: 10
- **a** (*float*) – 0 < a < 1. Default: 0.7
- **offset** (*int*) – How many periods to offset the result. Default: 0

Kwargs: adjust (bool): Default: True presma (bool, optional): If True, uses SMA for initial value. fillna (value, optional): pd.DataFrame.fillna(value) fill_method (value, optional): Type of fill method

Returns New feature generated.

Return type pd.Series

tema (*length=None*, *offset=None*, ***kwargs*)
Triple Exponential Moving Average (TEMA)

A less laggy Exponential Moving Average.

Sources: https://www.tradingtechnologies.com/help/x-study/technical-indicator-definitions/triple-exponential-moving-average-tema/

Calculation:

Default Inputs: length=10

EMA = Exponential Moving Average

ema1 = EMA(close, length)

ema2 = EMA(ema1, length)

ema3 = EMA(ema2, length)

TEMA = 3 * (ema1 - ema2) + ema3

Parameters

- **close** (*pd.Series*) – Series of 'close's
- **length** (*int*) – It's period. Default: 10
- **offset** (*int*) – How many periods to offset the result. Default: 0

Kwargs: adjust (bool): Default: True

presma (bool, optional): If True, uses SMA for initial value.

fillna (value, optional): pd.DataFrame.fillna(value)

fill_method (value, optional): Type of fill method

Returns New feature generated.

Return type pd.Series

trend_return(*trend: pandas.core.series.Series*, *log: bool = True*, *cumulative: bool = None*, *offset: int = None*, *trend_reset: int = 0*, ***kwargs*)

Trend Return

Calculates the (Cumulative) Returns of a Trend as defined by some conditional. By default it calculates log returns but can also use percent change.

Sources: Kevin Johnson

Calculation:

Default Inputs: trend_reset=0, log=True, cumulative=False

sum = 0

returns = log_return if log else percent_return # These are not cumulative

returns = (trend * returns).apply(zero)

for i, in range(0, trend.size):

if item == trend_reset: sum = 0

else: **return_** = returns.iloc[i] if cumulative:

sum += **return_**

else: sum = **return_**

trend_return.append(sum)

if cumulative and variable: trend_return += returns

Parameters

- **close** (*pd.Series*) – Series of 'close's
- **trend** (*pd.Series*) – Series of 'trend's. Preferably 0's and 1's.
- **trend_reset** (*value*) – Value used to identify if a trend has ended. Default: 0
- **log** (*bool*) – Calculate logarithmic returns. Default: True
- **cumulative** (*bool*) – If True, returns the cumulative returns. Default: False
- **offset** (*int*) – How many periods to offset the result. Default: 0

Kwargs: fillna (value, optional): pd.DataFrame.fillna(value)

fill_method (value, optional): Type of fill method

variable (bool, optional): Whether to include if return fluxuations in the cumulative returns.

Returns New feature generated.

Return type pd.Series

trima (*length=None*, *offset=None*, ***kwargs*)
Triangular Moving Average (TRIMA)

A weighted moving average where the shape of the weights are triangular and the greatest weight is in the middle of the period.

Sources: https://www.tradingtechnologies.com/help/x-study/technical-indicator-definitions/triangular-moving-average-trima/ tma = sma(sma(src, ceil(length / 2)), floor(length / 2) + 1) # Tradingview trima = sma(sma(x, n), n) # Tradingview

Calculation:

Default Inputs: length=10

SMA = Simple Moving Average

half_length = math.round(0.5 * (length + 1))

SMA1 = SMA(close, half_length)

TRIMA = SMA(SMA1, half_length)

Parameters

- **close** (*pd.Series*) – Series of 'close's
- **length** (*int*) – It's period. Default: 10
- **offset** (*int*) – How many periods to offset the result. Default: 0

Kwargs: adjust (bool): Default: True

fillna (value, optional): pd.DataFrame.fillna(value)

fill_method (value, optional): Type of fill method

Returns New feature generated.

Return type pd.Series

trix (*length=None*, *drift=None*, *offset=None*, ***kwargs*)
Trix (TRIX)

TRIX is a momentum oscillator to identify divergences.

Sources: https://www.tradingview.com/wiki/TRIX

Calculation:

Default Inputs: length=18, drift=1

EMA = Exponential Moving Average

ROC = Rate of Change

ema1 = EMA(close, length)

ema2 = EMA(ema1, length)

ema3 = EMA(ema2, length)

TRIX = 100 * ROC(ema3, drift)

Parameters

- **close** (*pd.Series*) – Series of 'close's
- **length** (*int*) – It's period. Default: 18
- **drift** (*int*) – The difference period. Default: 1
- **offset** (*int*) – How many periods to offset the result. Default: 0

Kwargs: fillna (value, optional): pd.DataFrame.fillna(value)

fill_method (value, optional): Type of fill method

Returns New feature generated.

Return type pd.Series

true_range(*drift=None*, *offset=None*, ***kwargs*)
True Range

An method to expand a classical range (high minus low) to include possible gap scenarios.

Sources: https://www.macroption.com/true-range/

Calculation:

Default Inputs: drift=1

ABS = Absolute Value

prev_close = close.shift(drift)

TRUE_RANGE = ABS([high - low, high - prev_close, low - prev_close])

Parameters

- **high** (*pd.Series*) – Series of 'high's
- **low** (*pd.Series*) – Series of 'low's
- **close** (*pd.Series*) – Series of 'close's
- **drift** (*int*) – The shift period. Default: 1
- **offset** (*int*) – How many periods to offset the result. Default: 0

Kwargs: fillna (value, optional): pd.DataFrame.fillna(value)

fill_method (value, optional): Type of fill method

Returns New feature

Return type pd.Series

tsi(*fast=None*, *slow=None*, *drift=None*, *offset=None*, ***kwargs*)
True Strength Index (TSI)

The True Strength Index is a momentum indicator used to identify short-term swings while in the direction of the trend as well as determining overbought and oversold conditions.

Sources: https://www.investopedia.com/terms/t/tsi.asp

Calculation:

Default Inputs: fast=13, slow=25, drift=1

EMA = Exponential Moving Average

diff = close.diff(drift)

slow_ema = EMA(diff, slow)

fast_slow_ema = EMA(slow_ema, slow)

abs_diff_slow_ema = absolute_diff_ema = EMA(ABS(diff), slow)

abema = abs_diff_fast_slow_ema = EMA(abs_diff_slow_ema, fast)

TSI = 100 * fast_slow_ema / abema

Parameters

- **close** (*pd.Series*) – Series of 'close's
- **fast** (*int*) – The short period. Default: 13
- **slow** (*int*) – The long period. Default: 25
- **drift** (*int*) – The difference period. Default: 1
- **offset** (*int*) – How many periods to offset the result. Default: 0

Kwargs: fillna (value, optional): pd.DataFrame.fillna(value)

fill_method (value, optional): Type of fill method

Returns New feature generated.

Return type pd.Series

uo (*fast=None*, *medium=None*, *slow=None*, *fast_w=None*, *medium_w=None*, *slow_w=None*, *drift=None*, *offset=None*, ***kwargs*)
Ultimate Oscillator (UO)

The Ultimate Oscillator is a momentum indicator over three different periods. It attempts to correct false divergence trading signals.

Sources: https://www.tradingview.com/wiki/Ultimate_Oscillator_(UO)

Calculation:

Default Inputs: fast=7, medium=14, slow=28, fast_w=4.0, medium_w=2.0, slow_w=1.0, drift=1

min_low_or_pc = close.shift(drift).combine(low, min)

max_high_or_pc = close.shift(drift).combine(high, max)

bp = buying pressure = close - min_low_or_pc

tr = true range = max_high_or_pc - min_low_or_pc

fast_avg = SUM(bp, fast) / SUM(tr, fast)

medium_avg = SUM(bp, medium) / SUM(tr, medium)

slow_avg = SUM(bp, slow) / SUM(tr, slow)

total_weight = fast_w + medium_w + slow_w

weights = (fast_w * fast_avg) + (medium_w * medium_avg) + (slow_w * slow_avg)

UO = 100 * weights / total_weight

Parameters

- **high** (*pd.Series*) – Series of 'high's
- **low** (*pd.Series*) – Series of 'low's
- **close** (*pd.Series*) – Series of 'close's
- **fast** (*int*) – The Fast %K period. Default: 7
- **medium** (*int*) – The Slow %K period. Default: 14
- **slow** (*int*) – The Slow %D period. Default: 28
- **fast_w** (*float*) – The Fast %K period. Default: 4.0
- **medium_w** (*float*) – The Slow %K period. Default: 2.0
- **slow_w** (*float*) – The Slow %D period. Default: 1.0
- **drift** (*int*) – The difference period. Default: 1
- **offset** (*int*) – How many periods to offset the result. Default: 0

Kwargs: fillna (value, optional): pd.DataFrame.fillna(value)

fill_method (value, optional): Type of fill method

Returns New feature generated.

Return type pd.Series

update(*currency: str*, *base: str*, *time: str*, *limit: int*)

Set the current ohlcv dataframe to the latest data with the given args

Parameters

- **{str}** (*time*) –
- **{str}** –
- **{str}** –

variance(*length=None*, *offset=None*, ***kwargs*)

Rolling Variance

Sources:

Calculation:

Default Inputs: length=30

VARIANCE = close.rolling(length).var()

Parameters

- **close** (*pd.Series*) – Series of 'close's
- **length** (*int*) – It's period. Default: 30
- **offset** (*int*) – How many periods to offset the result. Default: 0

Kwargs: fillna (value, optional): pd.DataFrame.fillna(value) fill_method (value, optional): Type of fill method

Returns New feature generated.

Return type pd.Series

vortex (*length=None*, *drift=None*, *offset=None*, ***kwargs*)
Vortex

Two oscillators that capture positive and negative trend movement.

Sources: https://stockcharts.com/school/doku.php?id=chart_school:technical_indicators:vortex_indicator

Calculation:

Default Inputs: length=14, drift=1

TR = True Range

SMA = Simple Moving Average

tr = TR(high, low, close)

tr_sum = tr.rolling(length).sum()

vmp = (high - low.shift(drift)).abs()

vmn = (low - high.shift(drift)).abs()

VIP = vmp.rolling(length).sum() / tr_sum

VIM = vmn.rolling(length).sum() / tr_sum

Parameters

- **high** (*pd.Series*) – Series of 'high's
- **low** (*pd.Series*) – Series of 'low's
- **close** (*pd.Series*) – Series of 'close's
- **length** (*int*) – ROC 1 period. Default: 14
- **drift** (*int*) – The difference period. Default: 1
- **offset** (*int*) – How many periods to offset the result. Default: 0

Kwargs: fillna (value, optional): pd.DataFrame.fillna(value)

fill_method (value, optional): Type of fill method

Returns vip and vim columns

Return type pd.DataFrame

vp (*width=None*, ***kwargs*)
Volume Profile (VP)

Calculates the Volume Profile by slicing price into ranges. Note: Value Area is not calculated.

Sources: https://stockcharts.com/school/doku.php?id=chart_school:technical_indicators:volume_by_price https://www.tradingview.com/wiki/Volume_Profile http://www.ranchodinero.com/volume-tpo-essentials/ https://www.tradingtechnologies.com/blog/2013/05/15/volume-at-price/

Calculation:

Default Inputs: width=10

vp = pd.concat([close, pos_volume, neg_volume], axis=1)

vp_ranges = np.array_split(vp, width)

result = ({high_close, low_close, mean_close, neg_volume, pos_volume} foreach range in vp_ranges)

vpdf = pd.DataFrame(result)

vpdf['total_volume'] = vpdf['pos_volume'] + vpdf['neg_volume']

Parameters

- **close** (*pd.Series*) – Series of 'close's
- **volume** (*pd.Series*) – Series of 'volume's
- **width** (*int*) – How many ranges to distrubute price into. Default: 10

Kwargs: fillna (value, optional): pd.DataFrame.fillna(value)

fill_method (value, optional): Type of fill method

sort_close (value, optional): Whether to sort by close before splitting into ranges. Default: False

Returns New feature generated.

Return type pd.DataFrame

vwap(*offset=None*, ***kwargs*)
Volume Weighted Average Price (VWAP)

The Volume Weighted Average Price that measures the average typical price by volume. It is typically used with intraday charts to identify general direction.

Sources: https://www.tradingview.com/wiki/Volume_Weighted_Average_Price_(VWAP)
https://www.tradingtechnologies.com/help/x-study/technical-indicator-definitions/volume-weighted-average-price-vwap/

Calculation: tp = typical_price = hlc3(high, low, close)

tpv = tp * volume

VWAP = tpv.cumsum() / volume.cumsum()

Parameters

- **high** (*pd.Series*) – Series of 'high's
- **low** (*pd.Series*) – Series of 'low's
- **close** (*pd.Series*) – Series of 'close's
- **volume** (*pd.Series*) – Series of 'volume's
- **offset** (*int*) – How many periods to offset the result. Default: 0

Kwargs: fillna (value, optional): pd.DataFrame.fillna(value)

fill_method (value, optional): Type of fill method

Returns New feature generated.

Return type pd.Series

vwma(*length=None*, *offset=None*, ***kwargs*) → pandas.core.series.Series
Volume Weighted Moving Average (VWMA)

Volume Weighted Moving Average.

Sources: https://www.motivewave.com/studies/volume_weighted_moving_average.htm

Calculation:

Default Inputs: length=10

SMA = Simple Moving Average

pv = close * volume

VWMA = SMA(pv, length) / SMA(volume, length)

Parameters

- **close** (*pd.Series*) – Series of 'close's
- **volume** (*pd.Series*) – Series of 'volume's
- **length** (*int*) – It's period. Default: 10
- **offset** (*int*) – How many periods to offset the result. Default: 0

Kwargs: fillna (value, optional): pd.DataFrame.fillna(value)

fill_method (value, optional): Type of fill method

Returns New feature generated.

Return type pd.Series

willr (*length=None*, *offset=None*, ***kwargs*)

William's Percent R (WILLR)

William's Percent R is a momentum oscillator similar to the RSI that attempts to identify overbought and oversold conditions.

Sources: https://www.tradingview.com/wiki/Williams_%25R_(%25R)

Calculation:

Default Inputs: length=20

lowest_low = low.rolling(length).min()

highest_high = high.rolling(length).max()

WILLR = 100 * ((close - lowest_low) / (highest_high - lowest_low) - 1)

Parameters

- **high** (*pd.Series*) – Series of 'high's
- **low** (*pd.Series*) – Series of 'low's
- **close** (*pd.Series*) – Series of 'close's
- **length** (*int*) – It's period. Default: 14
- **offset** (*int*) – How many periods to offset the result. Default: 0

Kwargs: fillna (value, optional): pd.DataFrame.fillna(value)

fill_method (value, optional): Type of fill method

Returns New feature generated.

Return type pd.Series

wma (*length=None*, *asc=None*, *offset=None*, ***kwargs*)
Weighted Moving Average (WMA)

The Weighted Moving Average where the weights are linearly increasing and the most recent data has the heaviest weight.

Sources: https://en.wikipedia.org/wiki/Moving_average#Weighted_moving_average

Calculation:

Default Inputs: length=10, asc=True

total_weight = 0.5 * length * (length + 1)

weights_ = [1, 2, . . . , length + 1] # Ascending

weights = weights if asc else weights[::-1]

def linear_weights(w):

def _compute(x): return (w * x).sum() / total_weight

return _compute

WMA = close.rolling(length)_.apply(linear_weights(weights), raw=True)

Parameters

- **close** (*pd.Series*) – Series of 'close's
- **length** (*int*) – It's period. Default: 10
- **asc** (*bool*) – Recent values weigh more. Default: True
- **offset** (*int*) – How many periods to offset the result. Default: 0

Kwargs: fillna (value, optional): pd.DataFrame.fillna(value)

fill_method (value, optional): Type of fill method

Returns New feature generated.

Return type pd.Series

zlma (*length=None*, *offset=None*, *mamode=None*, ***kwargs*)
Zero Lag Moving Average (ZLMA)

The Zero Lag Moving Average attempts to eliminate the lag associated with moving averages. This is an adaption created by John Ehler and Ric Way.

Sources: https://en.wikipedia.org/wiki/Zero_lag_exponential_moving_average

Calculation:

Default Inputs: length=10, mamode=EMA

EMA = Exponential Moving Average

lag = int(0.5 * (length - 1))

source = 2 * close - close.shift(lag)

ZLMA = EMA(source, length)

Parameters

- **close** (*pd.Series*) – Series of 'close's
- **length** (*int*) – It's period. Default: 10
- **mamode** (*str*) – Two options: None or 'ema'. Default: 'ema'
- **offset** (*int*) – How many periods to offset the result. Default: 0

Kwargs: fillna (value, optional): pd.DataFrame.fillna(value)

fill_method (value, optional): Type of fill method

Returns New feature generated.

Return type pd.Series

zscore (*length=None*, *std=None*, *offset=None*, ***kwargs*)

Rolling Z Score

Sources:

Calculation:

Default Inputs: length=30, std=1

SMA = Simple Moving Average

STDEV = Standard Deviation

std = std * STDEV(close, length)

mean = SMA(close, length)

ZSCORE = (close - mean) / std

Parameters

- **close** (*pd.Series*) – Series of 'close's
- **length** (*int*) – It's period. Default: 30
- **std** (*float*) – It's period. Default: 1
- **offset** (*int*) – How many periods to offset the result. Default: 0

Kwargs: fillna (value, optional): pd.DataFrame.fillna(value)

fill_method (value, optional): Type of fill method

Returns New feature generated.

Return type pd.Series

2.3.4 Ionomy

class Ionomy.**Ionomy** (*api_key: str*, *api_secret: str*)

Bases: object

Base Ionomy API Wrapper

Parameters

- **{str} -- Ionomy API key** (*api_key*) –

- **{str} -- Ionomy API Secret** (*api_secret*) –

balance (*currency: str*) → Dict[str, Union[str, float]]

balances () → List[Dict[str, Union[str, float]]]

cancel_order (*orderId: str*) → bool

currencies () → List[Dict[str, Union[str, bool, int, float]]]

deposit_address (*currency: str*) → Dict[str, str]

deposit_history (*currency: str*) → List[Dict[str, Union[str, float]]]

limit_buy (*amount: Union[int, float], price: Union[int, float], market: str*) → dict

limit_sell (*amount: Union[int, float], price: Union[int, float], market: str*) → dict

market_history (*market: str*) → List[Dict[str, Union[str, float]]]

market_summaries () → List[Dict[str, Union[str, int, float]]]

market_summary (*market: str*) → Dict[str, Union[str, int, float]]

markets () → List[Dict[str, Union[str, float, bool]]]

open_orders (*market: str*) → List[Dict[str, str]]

order_book (*market: str*) → Dict[str, List[Dict[str, float]]]

order_status (*orderId: str*) → Dict[str, Optional[str]]

withdraw (*currency*, *amount*, *address*)

withdrawal_history (*currency: str*) → List[Dict[str, Union[str, float]]]

2.3.5 IonPanda

class Ionomy.**IonPanda** (*api_key: str*, *api_secret: str*)

Bases: Ionomy.ionomy.Ionomy

Pandas DataFrame Wrapper for Ionomy Base Class

Parameters

- **{str} -- Ionomy API key** (*api_key*) –
- **{str} -- Ionomy API Secret** (*api_secret*) –

balances () → pandas.core.frame.DataFrame

currencies () → pandas.core.frame.DataFrame

deposit_history (*currency: str*) → pandas.core.frame.DataFrame

market_history (*market: str*) → pandas.core.frame.DataFrame

market_summaries () → pandas.core.frame.DataFrame

markets () → pandas.core.frame.DataFrame

open_orders (*market: str*) → pandas.core.frame.DataFrame

order_book (*market: str*) → pandas.core.frame.DataFrame

withdrawal_history (*currency: str*) → pandas.core.frame.DataFrame

CHAPTER 3

Misc

3.1 Contributing to ionomy-python

We welcome your contributions to our project.

3.1.1 Repository

The repository of ionomy-python is currently located at:

https://github.com/Distortedlogic/ionomy-python

3.1.2 Flow

This project makes heavy use of git flow. If you are not familiar with it, then the most important thing for your to understand is that:

pull requests need to be made against the develop branch

3.1.3 How to Contribute

0. Familiarize yourself with contributing on github
1. Fork or branch from the develop branch.
2. Create commits following the commit style
3. Start a pull request to the develop branch
4. Wait for a @distortedlogic or another member to review

3.1.4 Issues

Feel free to submit issues and enhancement requests.

3.1.5 Contributing

Please refer to each project's style guidelines and guidelines for submitting patches and additions. In general, we follow the "fork-and-pull" Git workflow.

1. **Fork** the repo on GitHub
2. **Clone** the project to your own machine
3. **Commit** changes to your own branch
4. **Push** your work back up to your fork
5. Submit a **Pull request** so that we can review your changes

Note: Be sure to merge the latest from "develop" before making a pull request!

3.1.6 Copyright and Licensing

This library is open sources under the MIT license. We require you to release your code under that license as well.

3.2 Support and Questions

Help and discussion channel for beem can be found here:

- https://discord.gg/ZvD9yXD

3.3 Indices and tables

- genindex
- modindex
- search

Index

W

Z