# Alpha Vantage API Overview

| Alpha Vantage | alpha_vantage Module |

**Evan Marie online**: | EvanMarie.com| EvanMarie@Proton.me | Linked In | GitHub | Hugging Face | Mastadon | Jovian.ai | TikTok | CodeWars | **Discord ➡ ✨ EvanMarie ✨#6114** |

## Pros and Cons of Alpha Vantage:

**Pros**

- Free (5 calls per minute, 500 per day), and premium is available
- Large datasets for crypto, stocks, etc
- 50+ technical indicators (such as SMA, Bollinger Bands)

**Cons**

- Limited call requests
- API key needed, compared to yfinance that requires no key
- No high frequency or real time data
- No data for bonds, funds, indexes, or commodities

```
from helpers_02_04 import *
import_all()
plt.style.use('pinks.mplstyle')
from alpha_vantage.timeseries import TimeSeries
%matplotlib inline
```

```
%%html
<style>
a:link {color: #35193e !important; font-weight: 600 !important;}
a:visited {color: #35193e !important; font-weight: 600 !important;}
</style>
```

| Top | Getting Data | Splits & Dividends | The Datetime Index | Frequency & Intervals | Technical Indicators | Foreign Exchange | Cryptocurrencies |

# Getting Data from Alpha Vantage

```
api_key = ''
ts = TimeSeries(key = api_key, output_format = 'pandas')
ts
```

```
<alpha_vantage.timeseries.TimeSeries at 0x7fb80b19cb20>
```

## Time Series Object

```
pretty('TimeSeries()')
help(ts)
```

TimeSeries()

```
Help on TimeSeries in module alpha_vantage.timeseries object:

class TimeSeries(alpha_vantage.alphavantage.AlphaVantage)
 |  TimeSeries(key=None, output_format='json', treat_info_as_error=True,
indexing_type='date', proxy=None, rapidapi=False)
 |
 |  This class implements all the api calls to times series
 |
 |  Method resolution order:
 |      TimeSeries
 |      alpha_vantage.alphavantage.AlphaVantage
 |      builtins.object
 |
 |  Methods defined here:
 |
 |  get_daily(self, symbol, outputsize='compact')
 |      Return daily time series in two json objects as data and
 |      meta_data. It raises ValueError when problems arise
 |
 |      Keyword Arguments:
 |          symbol:  the symbol for the equity we want to get its data
 |          outputsize:  The size of the call, supported values are
 |              'compact' and 'full; the first returns the last 100 points in the
 |              data series, and 'full' returns the full-length daily times
 |              series, commonly above 1MB (default 'compact')
 |
 |  get_daily_adjusted(self, symbol, outputsize='compact')
 |      Return daily adjusted (date, daily open, daily high, daily low,
 |      daily close, daily split/dividend-adjusted close, daily volume)
 |      time series in two json objects as data and
 |      meta_data. It raises ValueError when problems arise
```

```
|
|      Keyword Arguments:
|          symbol:  the symbol for the equity we want to get its data
|          outputsize:  The size of the call, supported values are
|              'compact' and 'full; the first returns the last 100 points in the
|              data series, and 'full' returns the full-length daily times
|              series, commonly above 1MB (default 'compact')
|
|  get_intraday(self, symbol, interval='15min', outputsize='compact')
|      Return intraday time series in two json objects as data and
|      meta_data. It raises ValueError when problems arise
|
|      Keyword Arguments:
|          symbol:  the symbol for the equity we want to get its data
|          interval:  time interval between two conscutive values,
|              supported values are '1min', '5min', '15min', '30min', '60min'
|              (default '15min')
|          outputsize:  The size of the call, supported values are
|              'compact' and 'full; the first returns the last 100 points in the
|              data series, and 'full' returns the full-length intraday times
|              series, commonly above 1MB (default 'compact')
|
|  get_intraday_extended(self, symbol, interval='15min', slice='year1month1')
|      Return extended intraday time series in one csv_reader object.
|      It raises ValueError when problems arise
|
|      Keyword Arguments:
|          symbol:  the symbol for the equity we want to get its data
|          interval:  time interval between two conscutive values,
|              supported values are '1min', '5min', '15min', '30min', '60min'
|              (default '15min')
|          slice: the trailing 2 years of intraday data is evenly divided into
|              24 "slices" - year1month1, year1month2, ..., year2month12
|
|  get_monthly(self, symbol)
|      Return monthly time series in two json objects as data and
|      meta_data. It raises ValueError when problems arise
|
|      Keyword Arguments:
|          symbol:  the symbol for the equity we want to get its data
|
|  get_monthly_adjusted(self, symbol)
|      Return monthly time series in two json objects as data and
```

```
 |      meta_data. It raises ValueError when problems arise
 |
 |      Keyword Arguments:
 |          symbol:  the symbol for the equity we want to get its data
 |
 |  get_quote_endpoint(self, symbol)
 |      Return the latest price and volume information for a
 |       security of your choice
 |
 |      Keyword Arguments:
 |          symbol:  the symbol for the equity we want to get its data
 |
 |  get_symbol_search(self, keywords)
 |      Return best matching symbols and market information
 |      based on keywords. It raises ValueError when problems arise
 |
 |      Keyword Arguments:
 |          keywords: the keywords to query on
 |
 |  get_weekly(self, symbol)
 |      Return weekly time series in two json objects as data and
 |      meta_data. It raises ValueError when problems arise
 |
 |      Keyword Arguments:
 |          symbol:  the symbol for the equity we want to get its data
 |
 |  get_weekly_adjusted(self, symbol)
 |      weekly adjusted time series (last trading day of each week,
 |      weekly open, weekly high, weekly low, weekly close, weekly adjusted
 |      close, weekly volume, weekly dividend) of the equity specified,
 |      covering up to 20 years of historical data.
 |      Keyword Arguments:
 |          symbol:  the symbol for the equity we want to get its data
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from alpha_vantage.alphavantage.AlphaVantage:
 |
 |  __init__(self, key=None, output_format='json', treat_info_as_error=True,
indexing_type='date', proxy=None, rapidapi=False)
 |      Initialize the class
 |
 |      Keyword Arguments:
 |          key:  Alpha Vantage api key
```

```
|          retries:  Maximum amount of retries in case of faulty connection or
|              server not able to answer the call.
|          treat_info_as_error: Treat information from the api as errors
|          output_format:  Either 'json', 'pandas' os 'csv'
|          indexing_type: Either 'date' to use the default date string given
|          by the alpha vantage api call or 'integer' if you just want an
|          integer indexing on your dataframe. Only valid, when the
|          output_format is 'pandas'
|          proxy: Dictionary mapping protocol or protocol and hostname to
|          the URL of the proxy.
|          rapidapi: Boolean describing whether or not the API key is
|          through the RapidAPI platform or not
|
|  map_to_matype(self, matype)
|      Convert to the alpha vantage math type integer. It returns an
|      integer correspondent to the type of math to apply to a function. It
|      raises ValueError if an integer greater than the supported math types
|      is given.
|
|      Keyword Arguments:
|          matype:  The math type of the alpha vantage api. It accepts
|          integers or a string representing the math type.
|
|              * 0 = Simple Moving Average (SMA),
|              * 1 = Exponential Moving Average (EMA),
|              * 2 = Weighted Moving Average (WMA),
|              * 3 = Double Exponential Moving Average (DEMA),
|              * 4 = Triple Exponential Moving Average (TEMA),
|              * 5 = Triangular Moving Average (TRIMA),
|              * 6 = T3 Moving Average,
|              * 7 = Kaufman Adaptive Moving Average (KAMA),
|              * 8 = MESA Adaptive Moving Average (MAMA)
|
|  set_proxy(self, proxy=None)
|      Set a new proxy configuration
|
|      Keyword Arguments:
|          proxy: Dictionary mapping protocol or protocol and hostname to
|          the URL of the proxy.
|
|  ----------------------------------------------------------------------
|  Data descriptors inherited from alpha_vantage.alphavantage.AlphaVantage:
|
```

```
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)
```

```
GE = ts.get_daily("GE")
```

```
pretty('ts.get_daily()')
help(ts.get_daily)
```

ts.get_daily()

```
Help on method get_daily in module alpha_vantage.timeseries:

get_daily(symbol, outputsize='compact') method of alpha_vantage.timeseries.TimeSeries
instance
    Return daily time series in two json objects as data and
    meta_data. It raises ValueError when problems arise

    Keyword Arguments:
        symbol:  the symbol for the equity we want to get its data
        outputsize:  The size of the call, supported values are
            'compact' and 'full; the first returns the last 100 points in the
            data series, and 'full' returns the full-length daily times
            series, commonly above 1MB (default 'compact')
```

## Resulting Data & Types

```
type(GE)
```

```
tuple
```

```
len(GE)
```

```
2
```

```
GE[0][0:5]
```

| date | 1. open | 2. high | 3. low | 4. close | 5. volume |
|---|---|---|---|---|---|
| **2023-02-03** | 83.51 | 83.69 | 81.82 | 81.96 | 5,778,181.00 |

| date | 1. open | 2. high | 3. low | 4. close | 5. volume |
|---|---|---|---|---|---|
| **2023-02-02** | 82.19 | 84.03 | 81.90 | 83.94 | 8,293,043.00 |
| **2023-02-01** | 80.27 | 82.47 | 80.01 | 82.32 | 7,272,316.00 |
| **2023-01-31** | 80.45 | 80.91 | 79.47 | 80.48 | 6,883,537.00 |
| **2023-01-30** | 82.41 | 82.75 | 80.67 | 80.83 | 5,726,899.00 |

```
pretty('GE[1] contains meta information')
GE[1]
```

GE[1] contains meta information

```
{'1. Information': 'Daily Prices (open, high, low, close) and Volumes',
 '2. Symbol': 'GE',
 '3. Last Refreshed': '2023-02-03',
 '4. Output Size': 'Compact',
 '5. Time Zone': 'US/Eastern'}
```

```
pretty('GE[0] contains the DataFrame')
df_overview(GE[0])
```

GE[0] contains the DataFrame

DataFrame Columns

| | 1. open | 2. high | 3. low | 4. close | 5. volume |
|---|---|---|---|---|---|
| **datatype** | float64 | float64 | float64 | float64 | float64 |
| **missing values** | 0 | 0 | 0 | 0 | 0 |
| **count** | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 |
| **mean** | 76.78 | 77.95 | 75.89 | 77.06 | 6,827,626.88 |
| **std** | 7.74 | 7.67 | 7.71 | 7.70 | 2,520,143.25 |
| **min** | 62.64 | 63.29 | 61.88 | 61.91 | 1,894,075.00 |
| **25%** | 69.97 | 71.00 | 68.94 | 70.16 | 4,948,897.75 |
| **50%** | 78.06 | 80.34 | 77.54 | 79.06 | 6,390,139.00 |
| **75%** | 83.29 | 84.03 | 81.95 | 83.50 | 7,993,457.25 |
| **max** | 87.70 | 88.38 | 87.35 | 88.14 | 16,784,586.00 |

DataFrame Key Points

| | |
|---|---|
| **total rows** | 100 |
| **total columns** | 5 |
| **column names** | 1. open, 2. high, 3. low, 4. close, 5. volume |
| **index start** | 2023-02-03 00:00:00 |
| **index end** | 2022-09-13 00:00:00 |
| **total missing values** | 0 |

DataFrame Head and Tail

head(3)

| date | 1. open | 2. high | 3. low | 4. close | 5. volume |
|---|---|---|---|---|---|
| **2023-02-03** | 83.51 | 83.69 | 81.82 | 81.96 | 5,778,181.00 |
| **2023-02-02** | 82.19 | 84.03 | 81.90 | 83.94 | 8,293,043.00 |
| **2023-02-01** | 80.27 | 82.47 | 80.01 | 82.32 | 7,272,316.00 |

tail(3)

| date | 1. open | 2. high | 3. low | 4. close | 5. volume |
|---|---|---|---|---|---|
| **2022-09-15** | 69.78 | 70.79 | 68.75 | 68.91 | 4,927,334.00 |
| **2022-09-14** | 70.70 | 71.00 | 68.68 | 70.03 | 6,210,344.00 |
| **2022-09-13** | 73.26 | 73.86 | 70.62 | 70.84 | 7,210,803.00 |

## Setting Specific Time Periods

```
GE = ts.get_daily('GE', outputsize = 'compact')[0]
```

### *compact* returns the last 100 timestamps (default)

```
see(GE.head(3), 'outputsize = "compact"')
```

outputsize = "compact"

| date | 1. open | 2. high | 3. low | 4. close | 5. volume |
|---|---|---|---|---|---|
| **2023-02-03** | 83.51 | 83.69 | 81.82 | 81.96 | 5,778,181.00 |
| **2023-02-02** | 82.19 | 84.03 | 81.90 | 83.94 | 8,293,043.00 |
| **2023-02-01** | 80.27 | 82.47 | 80.01 | 82.32 | 7,272,316.00 |

```
pretty(len(GE), 'len(GE)')
```

len(GE)

100

```
GE_full = ts.get_daily('GE', outputsize = 'full')[0]
```

### *full* returns the last 100 timestamps

```
see(GE_full.head(3), 'outputsize = "full"')
```

outputsize = "full"

| date | 1. open | 2. high | 3. low | 4. close | 5. volume |
|---|---|---|---|---|---|
| **2023-02-03** | 83.51 | 83.69 | 81.82 | 81.96 | 5,778,181.00 |
| **2023-02-02** | 82.19 | 84.03 | 81.90 | 83.94 | 8,293,043.00 |
| **2023-02-01** | 80.27 | 82.47 | 80.01 | 82.32 | 7,272,316.00 |

```
pretty(f'{len(GE_full):,}', 'len(GE_full)')
```

len(GE_full)

5,853

---

| Top | Getting Data | Splits & Dividends | The Datetime Index | Frequency & Intervals | Technical Indicators | Foreign Exchange | Cryptocurrencies |

---

# Stock Splits & Dividends

```
import yfinance as yf
```

```
ticker = 'GE'
```

```
GE_full = ts.get_daily(ticker, outputsize = 'full')[0]
```

```
pretty('GE_full meta informtation')
ts.get_daily(ticker, outputsize = 'full')[1]
```

GE_full meta informtation

```
{'1. Information': 'Daily Prices (open, high, low, close) and Volumes',
 '2. Symbol': 'GE',
 '3. Last Refreshed': '2023-02-03',
 '4. Output Size': 'Full size',
 '5. Time Zone': 'US/Eastern'}
```

### Observing differences in importing from Alpha Vantage versus yfinance

```
see(GE_full.head(3), 'Alpha Vantage Version, outputsize = "full"')
```

Alpha Vantage Version, outputsize = "full"

| date | 1. open | 2. high | 3. low | 4. close | 5. volume |
|---|---|---|---|---|---|
| **2023-02-03** | 83.51 | 83.69 | 81.82 | 81.96 | 5,778,181.00 |
| **2023-02-02** | 82.19 | 84.03 | 81.90 | 83.94 | 8,293,043.00 |
| **2023-02-01** | 80.27 | 82.47 | 80.01 | 82.32 | 7,272,316.00 |

### *Adj Close* - Adjusted for stock splits and dividends

```
see(yf.download(ticker, start = GE_full.index[0]),
    'yfinance version, same time period')
```

```
[**********************100%**********************]  1 of 1 completed
```

yfinance version, same time period

| Date | Open | High | Low | Close | Adj Close | Volume |
|---|---|---|---|---|---|---|
| **2023-02-03** | 83.51 | 83.69 | 81.82 | 81.96 | 81.96 | 5777400 |

### *ts.get_daily_adjusted()* - also returns the dividend amount and split coefficient

```
GE_adj = ts.get_daily_adjusted(ticker, outputsize = 'full')[0]
```

```
pretty('ts.get_daily_adjusted()')
help(ts.get_daily_adjusted)
```

ts.get_daily_adjusted()

```
Help on method get_daily_adjusted in module alpha_vantage.timeseries:

get_daily_adjusted(symbol, outputsize='compact') method of
alpha_vantage.timeseries.TimeSeries instance
    Return daily adjusted (date, daily open, daily high, daily low,
    daily close, daily split/dividend-adjusted close, daily volume)
    time series in two json objects as data and
    meta_data. It raises ValueError when problems arise

    Keyword Arguments:
        symbol:  the symbol for the equity we want to get its data
        outputsize:  The size of the call, supported values are
```

```
'compact' and 'full; the first returns the last 100 points in the
data series, and 'full' returns the full-length daily times
series, commonly above 1MB (default 'compact')
```

```
pretty('GE_adj meta informtation')
ts.get_daily_adjusted(ticker, outputsize = 'full')[1]
```

GE_adj meta informtation

```
{'1. Information': 'Daily Time Series with Splits and Dividend Events',
 '2. Symbol': 'GE',
 '3. Last Refreshed': '2023-02-03',
 '4. Output Size': 'Full size',
 '5. Time Zone': 'US/Eastern'}
```

```
df_overview(GE_adj, title = "GE 'get_daily_adjusted'")
```

GE 'get_daily_adjusted' Columns

| | 1. open | 2. high | 3. low | 4. close | 5. adjusted close | 6. volume | 7. dividend amount | 8. split coefficient |
|---|---|---|---|---|---|---|---|---|
| **datatype** | float64 | float64 | float64 | float64 | float64 | float64 | float64 | float64 |
| **missing values** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **count** | 5,853.00 | 5,853.00 | 5,853.00 | 5,853.00 | 5,853.00 | 5,853.00 | 5,853.00 | 5,853.00 |
| **mean** | 32.28 | 32.65 | 31.89 | 32.28 | 109.51 | 46,176,936.51 | 0.00 | 1.00 |
| **std** | 25.15 | 25.52 | 24.80 | 25.17 | 37.54 | 43,541,739.59 | 0.03 | 0.03 |
| **min** | 5.61 | 5.66 | 5.48 | 5.49 | 28.93 | 1,779,700.00 | 0.00 | 0.12 |
| **25%** | 17.45 | 17.69 | 17.09 | 17.39 | 76.86 | 20,144,600.00 | 0.00 | 1.00 |
| **50%** | 27.31 | 27.55 | 27.07 | 27.32 | 112.15 | 33,891,600.00 | 0.00 | 1.00 |
| **75%** | 35.33 | 35.58 | 35.09 | 35.30 | 136.48 | 58,201,149.00 | 0.00 | 1.00 |
| **max** | 166.13 | 167.94 | 161.31 | 166.00 | 199.77 | 752,904,400.00 | 0.42 | 3.00 |

GE 'get_daily_adjusted' Key Points

| | |
|---|---|
| **total rows** | 5,853 |
| **total columns** | 8 |
| **column names** | 1. open, 2. high, 3. low, 4. close, 5. adjusted close, 6. volume, 7. dividend amount, 8. split coefficient |
| **index start** | 2023-02-03 00:00:00 |
| **index end** | 1999-11-01 00:00:00 |
| **total missing values** | 0 |

GE 'get_daily_adjusted' Head and Tail

head(3)

| date | 1. open | 2. high | 3. low | 4. close | 5. adjusted close | 6. volume | 7. dividend amount | 8. split coefficient |
|---|---|---|---|---|---|---|---|---|
| **2023-02-03** | 83.51 | 83.69 | 81.82 | 81.96 | 81.96 | 5,778,181.00 | 0.00 | 1.00 |
| **2023-02-02** | 82.19 | 84.03 | 81.90 | 83.94 | 83.94 | 8,293,043.00 | 0.00 | 1.00 |
| **2023-02-01** | 80.27 | 82.47 | 80.01 | 82.32 | 82.32 | 7,272,316.00 | 0.00 | 1.00 |

tail(3)

| date | 1. open | 2. high | 3. low | 4. close | 5. adjusted close | 6. volume | 7. dividend amount | 8. split coefficient |
|---|---|---|---|---|---|---|---|---|
| **1999-11-03** | 132.88 | 132.94 | 130.00 | 131.38 | 144.61 | 4,589,000.00 | 0.00 | 1.00 |
| **1999-11-02** | 129.69 | 133.13 | 128.19 | 129.00 | 141.99 | 6,340,600.00 | 0.00 | 1.00 |
| **1999-11-01** | 133.63 | 134.38 | 129.25 | 129.38 | 142.41 | 6,795,500.00 | 0.00 | 1.00 |

### *Splits* - There have been 3 stock splits within the timeframe of this data

```
see(GE_adj.iloc[:, -1].value_counts(), 'Value counts for split coefficient column')
```

Value counts for split coefficient column

| | 8. split coefficient |
|---|---|
| **1.00** | 5850 |
| **1.28** | 1 |
| **0.12** | 1 |
| **3.00** | 1 |

### *split coefficient == 3* - Investigating when the stock split by 3

```
GE_adj[GE_adj.iloc[:, -1] == 3]
```

| date | 1. open | 2. high | 3. low | 4. close | 5. adjusted close | 6. volume | 7. dividend amount | 8. split coefficient |
|---|---|---|---|---|---|---|---|---|
| **2000-05-08** | 52.13 | 52.88 | 51.63 | 52.44 | 174.13 | 3,892,167.00 | 0.00 | 3.00 |

```
header_text('Price Effects of Stock Split')

apply_style(GE_adj.loc['2000-05-03' : '2000-05-10'],
            lambda value: 'color: #ad1759;\
                           font-weight: 600;\
                           font-size: 16px;\
                           background-color: yellow'
            if (value < 100) & (value > 3) else None).format(precision = 2)
```

Price Effects of Stock Split

| date | 1. open | 2. high | 3. low | 4. close | 5. adjusted close | 6. volume | 7. dividend amount | 8. split coefficient |
|---|---|---|---|---|---|---|---|---|
| 2000-05-10 00:00:00 | 51.50 | 52.06 | 50.06 | 50.63 | 168.12 | 15059400.00 | 0.00 | 1.00 |
| 2000-05-09 00:00:00 | 52.38 | 52.69 | 50.88 | 52.13 | 173.10 | 13439400.00 | 0.00 | 1.00 |
| 2000-05-08 00:00:00 | 52.13 | 52.88 | 51.63 | 52.44 | 174.13 | 3892167.00 | 0.00 | 3.00 |
| 2000-05-05 00:00:00 | 154.00 | 160.00 | 153.50 | 158.00 | 174.88 | 6895300.00 | 0.00 | 1.00 |
| 2000-05-04 00:00:00 | 157.44 | 157.50 | 152.75 | 154.00 | 170.45 | 5137000.00 | 0.00 | 1.00 |
| 2000-05-03 00:00:00 | 159.50 | 160.00 | 154.56 | 156.06 | 172.73 | 5531600.00 | 0.00 | 1.00 |

| Top | Getting Data | Splits & Dividends | The Datetime Index | Frequency & Intervals | Technical Indicators | Foreign Exchange | Cryptocurrencies |

# Datetime Index

```
see(GE_full.head(3), 'Alpha Vantage Data')
```

Alpha Vantage Data

| date | 1. open | 2. high | 3. low | 4. close | 5. volume |
|---|---|---|---|---|---|
| 2023-02-03 | 83.51 | 83.69 | 81.82 | 81.96 | 5,778,181.00 |
| 2023-02-02 | 82.19 | 84.03 | 81.90 | 83.94 | 8,293,043.00 |
| 2023-02-01 | 80.27 | 82.47 | 80.01 | 82.32 | 7,272,316.00 |

```
GE_full.columns = ['open', 'high', 'low', 'close', 'volume']
```

```
timeseries_overview(GE_full, 'close')
```

DataFrame Overview | Primary Metric: close

| | measurement |
|---|---|
| total records | 5,853 |
| start date | 11/01/1999 |
| end date | 02/03/2023 |
| total columns | 5 |

| measurement | |
|---|---|
| column labels | open, high, low, close, volume |
| total missing values | 0 |
| close average | 32 |
| close std | 25 |
| close min | 5 |
| close 25% | 17 |
| close 50% | 27 |
| close 75% | 35 |
| close max | 166 |

### Getting a single year

```
GE_full.loc['2017']
```

| date | open | high | low | close | volume |
|---|---|---|---|---|---|
| **2017-12-29** | 17.27 | 17.53 | 17.27 | 17.45 | 75,906,686.00 |
| **2017-12-28** | 17.35 | 17.40 | 17.25 | 17.36 | 60,756,258.00 |
| **2017-12-27** | 17.46 | 17.63 | 17.31 | 17.38 | 58,655,208.00 |
| **2017-12-26** | 17.45 | 17.66 | 17.40 | 17.43 | 55,337,900.00 |
| **2017-12-22** | 17.51 | 17.56 | 17.40 | 17.50 | 46,370,400.00 |
| **...** | ... | ... | ... | ... | ... |
| **2017-01-09** | 31.64 | 31.66 | 31.43 | 31.46 | 21,262,120.00 |
| **2017-01-06** | 31.58 | 31.77 | 31.36 | 31.61 | 22,120,800.00 |
| **2017-01-05** | 31.57 | 31.75 | 31.31 | 31.52 | 25,856,523.00 |
| **2017-01-04** | 31.75 | 31.83 | 31.62 | 31.70 | 21,438,996.00 |
| **2017-01-03** | 31.67 | 31.84 | 31.40 | 31.69 | 32,149,537.00 |

251 rows × 5 columns

---

| Top | Getting Data | Splits & Dividends | The Datetime Index | Frequency & Intervals | Technical Indicators | Foreign Exchange | Cryptocurrencies |

---

# Frequency & Interval Settings

```
ticker = 'MSFT'
```

### *get_monthly_adjusted()* - monthly data

```
pretty('ts.get_monthly_adjusted()')
help(ts.get_monthly_adjusted)
```

ts.get_monthly_adjusted()

```
Help on method get_monthly_adjusted in module alpha_vantage.timeseries:

get_monthly_adjusted(symbol) method of alpha_vantage.timeseries.TimeSeries instance
    Return monthly time series in two json objects as data and
    meta_data. It raises ValueError when problems arise

    Keyword Arguments:
        symbol:  the symbol for the equity we want to get its data
```

```
ts.get_monthly_adjusted(ticker)[0].head(3)
```

| date | 1. open | 2. high | 3. low | 4. close | 5. adjusted close | 6. volume | 7. dividend amount |
|---|---|---|---|---|---|---|---|
| **2023-02-03** | 248.00 | 264.69 | 245.47 | 258.35 | 258.35 | 100,277,605.00 | 0.00 |
| **2023-01-31** | 243.08 | 249.83 | 219.35 | 247.81 | 247.81 | 666,168,068.00 | 0.00 |
| **2022-12-30** | 253.87 | 263.92 | 233.87 | 239.82 | 239.82 | 591,366,468.00 | 0.00 |

### *get_weekly_adjusted()* - weekly data

```
pretty('ts.get_weekly_adjusted()')
help(ts.get_weekly_adjusted)
```

ts.get_weekly_adjusted()

```
Help on method get_weekly_adjusted in module alpha_vantage.timeseries:

get_weekly_adjusted(symbol) method of alpha_vantage.timeseries.TimeSeries instance
    weekly adjusted time series (last trading day of each week,
    weekly open, weekly high, weekly low, weekly close, weekly adjusted
    close, weekly volume, weekly dividend) of the equity specified,
    covering up to 20 years of historical data.
    Keyword Arguments:
        symbol:  the symbol for the equity we want to get its data
```

```
ts.get_weekly_adjusted(ticker)[0].head(3)
```

| | 1. open | 2. high | 3. low | 4. close | 5. adjusted close | 6. volume | 7. dividend amount |
|---|---|---|---|---|---|---|---|

| date | 1. open | 2. high | 3. low | 4. close | 5. adjusted close | 6. volume | 7. dividend amount |
|---|---|---|---|---|---|---|---|
| date | | | | | | | |
| 2023-02-03 | 244.51 | 264.69 | 242.20 | 258.35 | 258.35 | 152,686,042.00 | 0.00 |
| 2023-01-27 | 241.10 | 249.83 | 230.90 | 248.16 | 248.16 | 198,648,466.00 | 0.00 |
| 2023-01-20 | 237.97 | 242.38 | 230.68 | 240.22 | 240.22 | 123,872,791.00 | 0.00 |

### *intraday* - interval of 60 minutes (default = 15 minutes)

```
pretty('ts.get_intraday()')
help(ts.get_intraday)
```

ts.get_intraday()

```
Help on method get_intraday in module alpha_vantage.timeseries:

get_intraday(symbol, interval='15min', outputsize='compact') method of
alpha_vantage.timeseries.TimeSeries instance
    Return intraday time series in two json objects as data and
    meta_data. It raises ValueError when problems arise

    Keyword Arguments:
        symbol:  the symbol for the equity we want to get its data
        interval:  time interval between two conscutive values,
            supported values are '1min', '5min', '15min', '30min', '60min'
            (default '15min')
        outputsize:  The size of the call, supported values are
            'compact' and 'full; the first returns the last 100 points in the
            data series, and 'full' returns the full-length intraday times
            series, commonly above 1MB (default 'compact')
```

```
ts.get_intraday(ticker, outputsize = 'full', interval = '60min')[0].head(3)
```

| | 1. open | 2. high | 3. low | 4. close | 5. volume |
|---|---|---|---|---|---|
| date | | | | | |
| 2023-02-03 20:00:00 | 258.22 | 258.25 | 258.00 | 258.00 | 7,798.00 |
| 2023-02-03 19:00:00 | 258.24 | 258.30 | 258.11 | 258.22 | 6,339.00 |
| 2023-02-03 18:00:00 | 258.35 | 258.35 | 258.01 | 258.33 | 9,124.00 |

### *intraday* - interval of 1 minute

```
ts.get_intraday(ticker, outputsize = 'full', interval = '1min')[0].head(3)
```

| date | 1. open | 2. high | 3. low | 4. close | 5. volume |
|---|---|---|---|---|---|
| 2023-02-03 20:00:00 | 258.00 | 258.00 | 258.00 | 258.00 | 465.00 |
| 2023-02-03 19:59:00 | 258.00 | 258.00 | 258.00 | 258.00 | 449.00 |
| 2023-02-03 19:58:00 | 258.00 | 258.00 | 258.00 | 258.00 | 228.00 |

| Top | Getting Data | Splits & Dividends | The Datetime Index | Frequency & Intervals | Technical Indicators | Foreign Exchange | Cryptocurrencies |

# Technical Indicators

- Typically used by day traders and technical analysts to find patterns in historical price and volume data
- Alpha Vantage offers 50 technical indicators

```
from alpha_vantage.techindicators import TechIndicators
```

```
indicator = TechIndicators(key = api_key, output_format = 'pandas')
```

```
pretty('TechIndicators()')
help(indicator)
```

TechIndicators()

```
Help on TechIndicators in module alpha_vantage.techindicators object:

class TechIndicators(alpha_vantage.alphavantage.AlphaVantage)
 |  TechIndicators(*args, **kwargs)
 |
 |  This class implements all the technical indicator api calls
 |
 |  Method resolution order:
 |      TechIndicators
 |      alpha_vantage.alphavantage.AlphaVantage
 |      builtins.object
 |
 |  Methods defined here:
 |
 |  __init__(self, *args, **kwargs)
 |      Inherit AlphaVantage base class with its default arguments
 |
```

```
 |  get_ad(self, symbol, interval='daily')
 |      Return the Chaikin A/D line values in two json
 |      objects as data and meta_data. It raises ValueError when problems arise
 |
 |      Keyword Arguments:
 |          symbol:  the symbol for the equity we want to get its data
 |          interval:  time interval between two conscutive values,
 |              supported values are '1min', '5min', '15min', '30min', '60min',
'daily',
 |              'weekly', 'monthly' (default 'daily')
 |
 |  get_adosc(self, symbol, interval='daily', fastperiod=None, slowperiod=None)
 |      Return the Chaikin A/D oscillator values in two
 |      json objects as data and meta_data. It raises ValueError when problems
 |      arise
 |
 |      Keyword Arguments:
 |          symbol:  the symbol for the equity we want to get its data
 |          interval:  time interval between two conscutive values,
 |              supported values are '1min', '5min', '15min', '30min', '60min',
'daily',
 |              'weekly', 'monthly' (default 'daily'
 |          fastperiod:  Positive integers are accepted (default=None)
 |          slowperiod:  Positive integers are accepted (default=None)
 |
 |  get_adx(self, symbol, interval='daily', time_period=20)
 |      Return  the average directional movement index values in two json
 |      objects as data and meta_data. It raises ValueError when problems arise
 |
 |      Keyword Arguments:
 |          symbol:  the symbol for the equity we want to get its data
 |          interval:  time interval between two conscutive values,
 |              supported values are '1min', '5min', '15min', '30min', '60min',
'daily',
 |              'weekly', 'monthly' (default 'daily')
 |          time_period:  How many data points to average (default 20)
 |
 |  get_adxr(self, symbol, interval='daily', time_period=20)
 |      Return  the average directional movement index  rating in two json
 |      objects as data and meta_data. It raises ValueError when problems arise
 |
 |      Keyword Arguments:
 |          symbol:  the symbol for the equity we want to get its data
```

```
 |          interval:  time interval between two conscutive values,
 |              supported values are '1min', '5min', '15min', '30min', '60min',
'daily',
 |              'weekly', 'monthly' (default 'daily')
 |          time_period:  How many data points to average (default 20)
 |
 |  get_apo(self, symbol, interval='daily', series_type='close', fastperiod=None,
slowperiod=None, matype=None)
 |      Return the absolute price oscillator values in two
 |      json objects as data and meta_data. It raises ValueError when problems
 |      arise
 |
 |      Keyword Arguments:
 |          symbol:  the symbol for the equity we want to get its data
 |          interval:  time interval between two conscutive values,
 |              supported values are '1min', '5min', '15min', '30min', '60min',
'daily',
 |              'weekly', 'monthly' (default '60min)'
 |          series_type:  The desired price type in the time series. Four types
 |              are supported: 'close', 'open', 'high', 'low' (default 'close')
 |          fastperiod:  Positive integers are accepted (default=None)
 |          slowperiod:  Positive integers are accepted (default=None)
 |          matype    :  Moving average type. By default, fastmatype=0.
 |              Integers 0 - 8 are accepted (check  down the mappings) or the string
 |              containing the math type can also be used.
 |
 |              * 0 = Simple Moving Average (SMA),
 |              * 1 = Exponential Moving Average (EMA),
 |              * 2 = Weighted Moving Average (WMA),
 |              * 3 = Double Exponential Moving Average (DEMA),
 |              * 4 = Triple Exponential Moving Average (TEMA),
 |              * 5 = Triangular Moving Average (TRIMA),
 |              * 6 = T3 Moving Average,
 |              * 7 = Kaufman Adaptive Moving Average (KAMA),
 |              * 8 = MESA Adaptive Moving Average (MAMA)
 |
 |  get_aroon(self, symbol, interval='daily', time_period=20, series_type='close')
 |      Return the aroon values in two json
 |      objects as data and meta_data. It raises ValueError when problems arise
 |
 |      Keyword Arguments:
 |          symbol:  the symbol for the equity we want to get its data
 |          interval:  time interval between two conscutive values,
```

```
 |              supported values are '1min', '5min', '15min', '30min', '60min',
'daily',
 |              'weekly', 'monthly' (default 'daily')
 |          time_period:  How many data points to average (default 20)
 |          series_type:  The desired price type in the time series. Four types
 |              are supported: 'close', 'open', 'high', 'low' (default 'close')
 |
 |  get_aroonosc(self, symbol, interval='daily', time_period=20, series_type='close')
 |      Return the aroon oscillator values in two json
 |      objects as data and meta_data. It raises ValueError when problems arise
 |
 |      Keyword Arguments:
 |          symbol:  the symbol for the equity we want to get its data
 |          interval:  time interval between two conscutive values,
 |              supported values are '1min', '5min', '15min', '30min', '60min',
'daily',
 |              'weekly', 'monthly' (default 'daily')
 |          time_period:  How many data points to average (default 20)
 |          series_type:  The desired price type in the time series. Four types
 |              are supported: 'close', 'open', 'high', 'low' (default 'close')
 |
 |  get_atr(self, symbol, interval='daily', time_period=20)
 |      Return the average true range values in two json objects as
 |      data and meta_data. It raises ValueError when problems arise
 |
 |      Keyword Arguments:
 |          symbol:  the symbol for the equity we want to get its data
 |          interval:  time interval between two conscutive values,
 |              supported values are '1min', '5min', '15min', '30min', '60min',
'daily',
 |              'weekly', 'monthly' (default 'daily')
 |          time_period:  How many data points to average (default 20)
 |
 |  get_bbands(self, symbol, interval='daily', time_period=20, series_type='close',
nbdevup=None, nbdevdn=None, matype=None)
 |      Return the bollinger bands values in two
 |      json objects as data and meta_data. It raises ValueError when problems
 |      arise
 |
 |      Keyword Arguments:
 |          symbol:  the symbol for the equity we want to get its data
 |          interval:  time interval between two conscutive values,
 |              supported values are '1min', '5min', '15min', '30min', '60min',
```

```
'daily',
 |              'weekly', 'monthly' (default 'daily')
 |          time_period:  Number of data points used to calculate each BBANDS value.
 |              Positive integers are accepted (e.g., time_period=60, time_period=200)
 |              (default=20)
 |          series_type:  The desired price type in the time series. Four types
 |              are supported: 'close', 'open', 'high', 'low' (default 'close')
 |          nbdevup:  The standard deviation multiplier of the upper band. Positive
 |              integers are accepted as default (default=2)
 |          nbdevdn:  The standard deviation multiplier of the lower band. Positive
 |              integers are accepted as default (default=2)
 |          matype :  Moving average type. By default, matype=0.
 |              Integers 0 - 8 are accepted (check  down the mappings) or the string
 |              containing the math type can also be used.
 |
 |              * 0 = Simple Moving Average (SMA),
 |              * 1 = Exponential Moving Average (EMA),
 |              * 2 = Weighted Moving Average (WMA),
 |              * 3 = Double Exponential Moving Average (DEMA),
 |              * 4 = Triple Exponential Moving Average (TEMA),
 |              * 5 = Triangular Moving Average (TRIMA),
 |              * 6 = T3 Moving Average,
 |              * 7 = Kaufman Adaptive Moving Average (KAMA),
 |              * 8 = MESA Adaptive Moving Average (MAMA)
 |
 |  get_bop(self, symbol, interval='daily', time_period=20)
 |      Return the balance of power values in two json
 |      objects as data and meta_data. It raises ValueError when problems arise
 |
 |      Keyword Arguments:
 |          symbol:  the symbol for the equity we want to get its data
 |          interval:  time interval between two conscutive values,
 |              supported values are '1min', '5min', '15min', '30min', '60min',
'daily',
 |              'weekly', 'monthly' (default 'daily')
 |          time_period:  How many data points to average (default 20)
 |
 |  get_cci(self, symbol, interval='daily', time_period=20)
 |      Return the commodity channel index values  in two json
 |      objects as data and meta_data. It raises ValueError when problems arise
 |
 |      Keyword Arguments:
 |          symbol:  the symbol for the equity we want to get its data
```

```
 |          interval:  time interval between two conscutive values,
 |              supported values are '1min', '5min', '15min', '30min', '60min',
'daily',
 |              'weekly', 'monthly' (default 'daily')
 |          time_period:  How many data points to average (default 20)
 |
 |  get_cmo(self, symbol, interval='daily', time_period=20, series_type='close')
 |      Return the Chande momentum oscillator in two json
 |      objects as data and meta_data. It raises ValueError when problems arise
 |
 |      Keyword Arguments:
 |          symbol:  the symbol for the equity we want to get its data
 |          interval:  time interval between two conscutive values,
 |              supported values are '1min', '5min', '15min', '30min', '60min',
'daily',
 |              'weekly', 'monthly' (default 'daily')
 |          time_period:  How many data points to average (default 20)
 |          series_type:  The desired price type in the time series. Four types
 |              are supported: 'close', 'open', 'high', 'low' (default 'close')
 |
 |  get_dema(self, symbol, interval='daily', time_period=20, series_type='close')
 |      Return double exponential moving average time series in two json
 |      objects as data and meta_data. It raises ValueError when problems arise
 |
 |      Keyword Arguments:
 |          symbol:  the symbol for the equity we want to get its data
 |          interval:  time interval between two conscutive values,
 |              supported values are '1min', '5min', '15min', '30min', '60min',
'daily',
 |              'weekly', 'monthly' (default 'daily')
 |          time_period:  How many data points to average (default 20)
 |          series_type:  The desired price type in the time series. Four types
 |              are supported: 'close', 'open', 'high', 'low' (default 'close')
 |
 |  get_dx(self, symbol, interval='daily', time_period=20, series_type='close')
 |      Return the directional movement index values in two json objects as
 |      data and meta_data. It raises ValueError when problems arise
 |
 |      Keyword Arguments:
 |          symbol:  the symbol for the equity we want to get its data
 |          interval:  time interval between two conscutive values,
 |              supported values are '1min', '5min', '15min', '30min', '60min',
'daily',
```

```
 |              'weekly', 'monthly' (default 'daily')
 |          time_period:  How many data points to average (default 20)
 |          series_type:  The desired price type in the time series. Four types
 |              are supported: 'close', 'open', 'high', 'low' (default 'close')
 |
 |  get_ema(self, symbol, interval='daily', time_period=20, series_type='close')
 |      Return exponential moving average time series in two json objects
 |      as data and meta_data. It raises ValueError when problems arise
 |
 |      Keyword Arguments:
 |          symbol:  the symbol for the equity we want to get its data
 |          interval:  time interval between two conscutive values,
 |              supported values are '1min', '5min', '15min', '30min', '60min',
'daily',
 |              'weekly', 'monthly' (default 'daily')
 |          time_period:  How many data points to average (default 20)
 |          series_type:  The desired price type in the time series. Four types
 |              are supported: 'close', 'open', 'high', 'low' (default 'close')
 |
 |  get_ht_dcperiod(self, symbol, interval='daily', series_type='close')
 |      Return the Hilbert transform, dominant cycle period in two
 |      json objects as data and meta_data. It raises ValueError when problems arise
 |
 |      Keyword Arguments:
 |          symbol:  the symbol for the equity we want to get its data
 |          interval:  time interval between two conscutive values,
 |              supported values are '1min', '5min', '15min', '30min', '60min',
'daily',
 |              'weekly', 'monthly' (default 'daily')
 |          series_type:  The desired price type in the time series. Four types
 |              are supported: 'close', 'open', 'high', 'low' (default 'close')
 |
 |  get_ht_dcphase(self, symbol, interval='daily', series_type='close')
 |      Return the Hilbert transform, dominant cycle phase in two
 |      json objects as data and meta_data. It raises ValueError when problems arise
 |
 |      Keyword Arguments:
 |          symbol:  the symbol for the equity we want to get its data
 |          interval:  time interval between two conscutive values,
 |              supported values are '1min', '5min', '15min', '30min', '60min',
'daily',
 |              'weekly', 'monthly' (default 'daily')
 |          series_type:  The desired price type in the time series. Four types
```

```
 |              are supported: 'close', 'open', 'high', 'low' (default 'close')
 |
 |  get_ht_phasor(self, symbol, interval='daily', series_type='close')
 |      Return the Hilbert transform, phasor components in two
 |      json objects as data and meta_data. It raises ValueError when problems arise
 |
 |      Keyword Arguments:
 |          symbol:  the symbol for the equity we want to get its data
 |          interval:  time interval between two conscutive values,
 |              supported values are '1min', '5min', '15min', '30min', '60min',
'daily',
 |              'weekly', 'monthly' (default 'daily')
 |          series_type:  The desired price type in the time series. Four types
 |              are supported: 'close', 'open', 'high', 'low' (default 'close')
 |
 |  get_ht_sine(self, symbol, interval='daily', series_type='close')
 |      Return the Hilbert transform, sine wave values in two
 |      json objects as data and meta_data. It raises ValueError when problems arise
 |
 |      Keyword Arguments:
 |          symbol:  the symbol for the equity we want to get its data
 |          interval:  time interval between two conscutive values,
 |              supported values are '1min', '5min', '15min', '30min', '60min',
'daily',
 |              'weekly', 'monthly' (default 'daily')
 |          series_type:  The desired price type in the time series. Four types
 |          are supported: 'close', 'open', 'high', 'low' (default 'close')
 |
 |  get_ht_trendline(self, symbol, interval='daily', series_type='close')
 |      Return the Hilbert transform, instantaneous trendline values in two
 |      json objects as data and meta_data. It raises ValueError when problems arise
 |
 |      Keyword Arguments:
 |          symbol:  the symbol for the equity we want to get its data
 |          interval:  time interval between two conscutive values,
 |              supported values are '1min', '5min', '15min', '30min', '60min',
'daily',
 |              'weekly', 'monthly' (default 'daily')
 |          series_type:  The desired price type in the time series. Four types
 |              are supported: 'close', 'open', 'high', 'low' (default 'close')
 |
 |  get_ht_trendmode(self, symbol, interval='daily', series_type='close')
 |      Return the Hilbert transform, trend vs cycle mode in two
```

```
 |      json objects as data and meta_data. It raises ValueError when problems arise
 |
 |      Keyword Arguments:
 |          symbol:  the symbol for the equity we want to get its data
 |          interval:  time interval between two conscutive values,
 |              supported values are '1min', '5min', '15min', '30min', '60min',
'daily',
 |              'weekly', 'monthly' (default 'daily')
 |          series_type:  The desired price type in the time series. Four types
 |              are supported: 'close', 'open', 'high', 'low' (default 'close')
 |
 |  get_kama(self, symbol, interval='daily', time_period=20, series_type='close')
 |      Return Kaufman adaptative moving average time series in two json
 |      objects as data and meta_data. It raises ValueError when problems arise
 |
 |      Keyword Arguments:
 |          symbol:  the symbol for the equity we want to get its data
 |          interval:  time interval between two conscutive values,
 |              supported values are '1min', '5min', '15min', '30min', '60min',
'daily',
 |              'weekly', 'monthly' (default 'daily')
 |          time_period:  How many data points to average (default 20)
 |          series_type:  The desired price type in the time series. Four types
 |              are supported: 'close', 'open', 'high', 'low' (default 'close')
 |
 |  get_macd(self, symbol, interval='daily', series_type='close', fastperiod=None,
slowperiod=None, signalperiod=None)
 |      Return the moving average convergence/divergence time series in two
 |      json objects as data and meta_data. It raises ValueError when problems
 |      arise
 |
 |      Keyword Arguments:
 |          symbol:  the symbol for the equity we want to get its data
 |          interval:  time interval between two conscutive values,
 |              supported values are '1min', '5min', '15min', '30min', '60min',
'daily',
 |              'weekly', 'monthly' (default 'daily'
 |          series_type:  The desired price type in the time series. Four types
 |              are supported: 'close', 'open', 'high', 'low' (default 'close')
 |          fastperiod:  Positive integers are accepted (default=None)
 |          slowperiod:  Positive integers are accepted (default=None)
 |          signalperiod:  Positive integers are accepted (default=None)
 |
```

```
 |  get_macdext(self, symbol, interval='daily', series_type='close', fastperiod=None,
slowperiod=None, signalperiod=None, fastmatype=None, slowmatype=None,
signalmatype=None)
 |      Return the moving average convergence/divergence time series in two
 |      json objects as data and meta_data. It raises ValueError when problems
 |      arise
 |
 |      Keyword Arguments:
 |          symbol:  the symbol for the equity we want to get its data
 |          interval:  time interval between two conscutive values,
 |              supported values are '1min', '5min', '15min', '30min', '60min',
'daily',
 |              'weekly', 'monthly' (default 'daily')
 |          series_type:  The desired price type in the time series. Four types
 |              are supported: 'close', 'open', 'high', 'low' (default 'close')
 |          fastperiod:  Positive integers are accepted (default=None)
 |          slowperiod:  Positive integers are accepted (default=None)
 |          signalperiod:  Positive integers are accepted (default=None)
 |          fastmatype:  Moving average type for the faster moving average.
 |              By default, fastmatype=0. Integers 0 - 8 are accepted
 |              (check  down the mappings) or the string containing the math type can
 |              also be used.
 |          slowmatype:  Moving average type for the slower moving average.
 |              By default, slowmatype=0. Integers 0 - 8 are accepted
 |              (check down the mappings) or the string containing the math type can
 |              also be used.
 |          signalmatype:  Moving average type for the signal moving average.
 |              By default, signalmatype=0. Integers 0 - 8 are accepted
 |              (check down the mappings) or the string containing the math type can
 |              also be used.
 |
 |              * 0 = Simple Moving Average (SMA),
 |              * 1 = Exponential Moving Average (EMA),
 |              * 2 = Weighted Moving Average (WMA),
 |              * 3 = Double Exponential Moving Average (DEMA),
 |              * 4 = Triple Exponential Moving Average (TEMA),
 |              * 5 = Triangular Moving Average (TRIMA),
 |              * 6 = T3 Moving Average,
 |              * 7 = Kaufman Adaptive Moving Average (KAMA),
 |              * 8 = MESA Adaptive Moving Average (MAMA)
 |
 |  get_mama(self, symbol, interval='daily', series_type='close', fastlimit=None,
slowlimit=None)
```

```
 |      Return MESA adaptative moving average time series in two json
 |      objects as data and meta_data. It raises ValueError when problems arise
 |
 |      Keyword Arguments:
 |          symbol:  the symbol for the equity we want to get its data
 |          interval:  time interval between two conscutive values,
 |              supported values are '1min', '5min', '15min', '30min', '60min',
'daily',
 |              'weekly', 'monthly' (default 'daily')
 |          series_type:  The desired price type in the time series. Four types
 |              are supported: 'close', 'open', 'high', 'low' (default 'close')
 |          fastlimit:  Positive floats for the fast limit are accepted
 |              (default=None)
 |          slowlimit:  Positive floats for the slow limit are accepted
 |              (default=None)
 |
 |  get_mfi(self, symbol, interval='daily', time_period=20, series_type='close')
 |      Return the money flow index values in two json
 |      objects as data and meta_data. It raises ValueError when problems arise
 |
 |      Keyword Arguments:
 |          symbol:  the symbol for the equity we want to get its data
 |          interval:  time interval between two conscutive values,
 |              supported values are '1min', '5min', '15min', '30min', '60min',
'daily',
 |              'weekly', 'monthly' (default 'daily')
 |          time_period:  How many data points to average (default 20)
 |          series_type:  The desired price type in the time series. Four types
 |              are supported: 'close', 'open', 'high', 'low' (default 'close')
 |
 |  get_midpoint(self, symbol, interval='daily', time_period=20, series_type='close')
 |      Return the midpoint values in two json objects as
 |      data and meta_data. It raises ValueError when problems arise
 |
 |      Keyword Arguments:
 |          symbol:  the symbol for the equity we want to get its data
 |          interval:  time interval between two conscutive values,
 |              supported values are '1min', '5min', '15min', '30min', '60min',
'daily',
 |              'weekly', 'monthly' (default 'daily')
 |          time_period:  How many data points to average (default 20)
 |          series_type:  The desired price type in the time series. Four types
 |              are supported: 'close', 'open', 'high', 'low' (default 'close')
```

```
 |
 |  get_midprice(self, symbol, interval='daily', time_period=20)
 |      Return the midprice values in two json objects as
 |      data and meta_data. It raises ValueError when problems arise
 |
 |      Keyword Arguments:
 |          symbol:  the symbol for the equity we want to get its data
 |          interval:  time interval between two conscutive values,
 |              supported values are '1min', '5min', '15min', '30min', '60min',
'daily',
 |              'weekly', 'monthly' (default 'daily')
 |          time_period:  How many data points to average (default 20)
 |
 |  get_minus_di(self, symbol, interval='daily', time_period=20)
 |      Return the minus directional indicator values in two json
 |      objects as data and meta_data. It raises ValueError when problems arise
 |
 |      Keyword Arguments:
 |          symbol:  the symbol for the equity we want to get its data
 |          interval:  time interval between two conscutive values,
 |              supported values are '1min', '5min', '15min', '30min', '60min',
'daily',
 |              'weekly', 'monthly' (default 'daily')
 |          time_period:  How many data points to average (default 20)
 |
 |  get_minus_dm(self, symbol, interval='daily', time_period=20)
 |      Return the minus directional movement values in two json
 |      objects as data and meta_data. It raises ValueError when problems arise
 |
 |      Keyword Arguments:
 |          symbol:  the symbol for the equity we want to get its data
 |          interval:  time interval between two conscutive values,
 |              supported values are '1min', '5min', '15min', '30min', '60min',
'daily',
 |              'weekly', 'monthly' (default 'daily')
 |
 |  get_mom(self, symbol, interval='daily', time_period=20, series_type='close')
 |      Return the momentum values in two json
 |      objects as data and meta_data. It raises ValueError when problems arise
 |
 |      Keyword Arguments:
 |          symbol:  the symbol for the equity we want to get its data
 |          interval:  time interval between two conscutive values,
```

```
 |              supported values are '1min', '5min', '15min', '30min', '60min',
'daily',
 |              'weekly', 'monthly' (default 'daily')
 |          time_period:  How many data points to average (default 20)
 |          series_type:  The desired price type in the time series. Four types
 |              are supported: 'close', 'open', 'high', 'low' (default 'close')
 |
 |  get_natr(self, symbol, interval='daily', time_period=20)
 |      Return the normalized average true range values in two json objects
 |      as data and meta_data. It raises ValueError when problems arise
 |
 |      Keyword Arguments:
 |          symbol:  the symbol for the equity we want to get its data
 |          interval:  time interval between two conscutive values,
 |              supported values are '1min', '5min', '15min', '30min', '60min',
'daily',
 |              'weekly', 'monthly' (default 'daily')
 |          time_period:  How many data points to average (default 20)
 |
 |  get_obv(self, symbol, interval='daily')
 |      Return the on balance volume values in two json
 |      objects as data and meta_data. It raises ValueError when problems arise
 |
 |      Keyword Arguments:
 |          symbol:  the symbol for the equity we want to get its data
 |          interval:  time interval between two conscutive values,
 |              supported values are '1min', '5min', '15min', '30min', '60min',
'daily',
 |              'weekly', 'monthly' (default 'daily')
 |
 |  get_plus_di(self, symbol, interval='daily', time_period=20)
 |      Return the plus directional indicator values in two json
 |      objects as data and meta_data. It raises ValueError when problems arise
 |
 |      Keyword Arguments:
 |          symbol:  the symbol for the equity we want to get its data
 |          interval:  time interval between two conscutive values,
 |              supported values are '1min', '5min', '15min', '30min', '60min',
'daily',
 |              'weekly', 'monthly' (default 'daily')
 |          time_period:  How many data points to average (default 20)
 |
 |  get_plus_dm(self, symbol, interval='daily', time_period=20)
```

```
 |      Return the plus directional movement values in two json
 |      objects as data and meta_data. It raises ValueError when problems arise
 |
 |      Keyword Arguments:
 |          symbol:  the symbol for the equity we want to get its data
 |          interval:  time interval between two conscutive values,
 |              supported values are '1min', '5min', '15min', '30min', '60min',
'daily',
 |              'weekly', 'monthly' (default 'daily')
 |
 |  get_ppo(self, symbol, interval='daily', series_type='close', fastperiod=None,
slowperiod=None, matype=None)
 |      Return the percentage price oscillator values in two
 |      json objects as data and meta_data. It raises ValueError when problems
 |      arise
 |
 |      Keyword Arguments:
 |          symbol:  the symbol for the equity we want to get its data
 |          interval:  time interval between two conscutive values,
 |              supported values are '1min', '5min', '15min', '30min', '60min',
'daily',
 |              'weekly', 'monthly' (default 'daily'
 |          series_type:  The desired price type in the time series. Four types
 |              are supported: 'close', 'open', 'high', 'low' (default 'close')
 |          fastperiod:  Positive integers are accepted (default=None)
 |          slowperiod:  Positive integers are accepted (default=None)
 |          matype    :  Moving average type. By default, fastmatype=0.
 |              Integers 0 - 8 are accepted (check  down the mappings) or the string
 |              containing the math type can also be used.
 |
 |              * 0 = Simple Moving Average (SMA),
 |              * 1 = Exponential Moving Average (EMA),
 |              * 2 = Weighted Moving Average (WMA),
 |              * 3 = Double Exponential Moving Average (DEMA),
 |              * 4 = Triple Exponential Moving Average (TEMA),
 |              * 5 = Triangular Moving Average (TRIMA),
 |              * 6 = T3 Moving Average,
 |              * 7 = Kaufman Adaptive Moving Average (KAMA),
 |              * 8 = MESA Adaptive Moving Average (MAMA)
 |
 |  get_roc(self, symbol, interval='daily', time_period=20, series_type='close')
 |      Return the rate of change values in two json
 |      objects as data and meta_data. It raises ValueError when problems arise
```

```
 |
 |      Keyword Arguments:
 |          symbol:  the symbol for the equity we want to get its data
 |          interval:  time interval between two conscutive values,
 |              supported values are '1min', '5min', '15min', '30min', '60min',
'daily',
 |              'weekly', 'monthly' (default 'daily')
 |          time_period:  How many data points to average (default 20)
 |          series_type:  The desired price type in the time series. Four types
 |              are supported: 'close', 'open', 'high', 'low' (default 'close')
 |
 |  get_rocr(self, symbol, interval='daily', time_period=20, series_type='close')
 |      Return the rate of change ratio values in two json
 |      objects as data and meta_data. It raises ValueError when problems arise
 |
 |      Keyword Arguments:
 |          symbol:  the symbol for the equity we want to get its data
 |          interval:  time interval between two conscutive values,
 |              supported values are '1min', '5min', '15min', '30min', '60min',
'daily',
 |              'weekly', 'monthly' (default 'daily')
 |          time_period:  How many data points to average (default 20)
 |          series_type:  The desired price type in the time series. Four types
 |              are supported: 'close', 'open', 'high', 'low' (default 'close')
 |
 |  get_rsi(self, symbol, interval='daily', time_period=20, series_type='close')
 |      Return the relative strength index time series in two json
 |      objects as data and meta_data. It raises ValueError when problems arise
 |
 |      Keyword Arguments:
 |          symbol:  the symbol for the equity we want to get its data
 |          interval:  time interval between two conscutive values,
 |              supported values are '1min', '5min', '15min', '30min', '60min',
'daily',
 |              'weekly', 'monthly' (default 'daily')
 |          time_period:  How many data points to average (default 20)
 |          series_type:  The desired price type in the time series. Four types
 |              are supported: 'close', 'open', 'high', 'low' (default 'close')
 |
 |  get_sar(self, symbol, interval='daily', acceleration=None, maximum=None)
 |      Return the midprice values in two json objects as
 |      data and meta_data. It raises ValueError when problems arise
 |
```

```
 |      Keyword Arguments:
 |          symbol:  the symbol for the equity we want to get its data
 |          interval:  time interval between two conscutive values,
 |              supported values are '1min', '5min', '15min', '30min', '60min',
'daily',
 |              'weekly', 'monthly' (default 'daily')
 |          acceleration:  The acceleration factor. Positive floats are accepted (
 |              default 0.01)
 |          maximum:  The acceleration factor maximum value. Positive floats
 |              are accepted (default 0.20 )
 |
 |  get_sma(self, symbol, interval='daily', time_period=20, series_type='close')
 |      Return simple moving average time series in two json objects as data and
 |      meta_data. It raises ValueError when problems arise
 |
 |      Keyword Arguments:
 |          symbol:  the symbol for the equity we want to get its data
 |          interval:  time interval between two conscutive values,
 |              supported values are '1min', '5min', '15min', '30min', '60min',
'daily',
 |              'weekly', 'monthly' (default 'daily')
 |          time_period:  How many data points to average (default 20)
 |          series_type:  The desired price type in the time series. Four types
 |              are supported: 'close', 'open', 'high', 'low' (default 'close')
 |
 |  get_stoch(self, symbol, interval='daily', fastkperiod=None, slowkperiod=None,
slowdperiod=None, slowkmatype=None, slowdmatype=None)
 |      Return the stochatic oscillator values in two
 |      json objects as data and meta_data. It raises ValueError when problems
 |      arise
 |
 |      Keyword Arguments:
 |          symbol:  the symbol for the equity we want to get its data
 |          interval:  time interval between two conscutive values,
 |              supported values are '1min', '5min', '15min', '30min', '60min',
'daily',
 |              'weekly', 'monthly' (default 'daily')
 |          fastkperiod:  The time period of the fastk moving average. Positive
 |              integers are accepted (default=None)
 |          slowkperiod:  The time period of the slowk moving average. Positive
 |              integers are accepted (default=None)
 |          slowdperiod: The time period of the slowd moving average. Positive
 |              integers are accepted (default=None)
```

```
 |          slowkmatype:  Moving average type for the slowk moving average.
 |              By default, fastmatype=0. Integers 0 - 8 are accepted
 |              (check  down the mappings) or the string containing the math type can
 |              also be used.
 |          slowdmatype:  Moving average type for the slowd moving average.
 |              By default, slowmatype=0. Integers 0 - 8 are accepted
 |              (check down the mappings) or the string containing the math type can
 |              also be used.
 |
 |              * 0 = Simple Moving Average (SMA),
 |              * 1 = Exponential Moving Average (EMA),
 |              * 2 = Weighted Moving Average (WMA),
 |              * 3 = Double Exponential Moving Average (DEMA),
 |              * 4 = Triple Exponential Moving Average (TEMA),
 |              * 5 = Triangular Moving Average (TRIMA),
 |              * 6 = T3 Moving Average,
 |              * 7 = Kaufman Adaptive Moving Average (KAMA),
 |              * 8 = MESA Adaptive Moving Average (MAMA)
 |
 |  get_stochf(self, symbol, interval='daily', fastkperiod=None, fastdperiod=None,
fastdmatype=None)
 |      Return the stochatic oscillator values in two
 |      json objects as data and meta_data. It raises ValueError when problems
 |      arise
 |
 |      Keyword Arguments:
 |          symbol:  the symbol for the equity we want to get its data
 |          interval:  time interval between two conscutive values,
 |              supported values are '1min', '5min', '15min', '30min', '60min',
'daily',
 |              'weekly', 'monthly' (default 'daily')
 |          fastkperiod:  The time period of the fastk moving average. Positive
 |              integers are accepted (default=None)
 |          fastdperiod:  The time period of the fastd moving average. Positive
 |              integers are accepted (default=None)
 |          fastdmatype:  Moving average type for the fastdmatype moving average.
 |              By default, fastmatype=0. Integers 0 - 8 are accepted
 |              (check  down the mappings) or the string containing the math type can
 |              also be used.
 |
 |              * 0 = Simple Moving Average (SMA),
 |              * 1 = Exponential Moving Average (EMA),
 |              * 2 = Weighted Moving Average (WMA),
```

```
 |              * 3 = Double Exponential Moving Average (DEMA),
 |              * 4 = Triple Exponential Moving Average (TEMA),
 |              * 5 = Triangular Moving Average (TRIMA),
 |              * 6 = T3 Moving Average,
 |              * 7 = Kaufman Adaptive Moving Average (KAMA),
 |              * 8 = MESA Adaptive Moving Average (MAMA)
 |
 |  get_stochrsi(self, symbol, interval='daily', time_period=20, series_type='close',
fastkperiod=None, fastdperiod=None, fastdmatype=None)
 |      Return the stochatic relative strength index in two
 |      json objects as data and meta_data. It raises ValueError when problems
 |      arise
 |
 |      Keyword Arguments:
 |          symbol:  the symbol for the equity we want to get its data
 |          interval:  time interval between two conscutive values,
 |              supported values are '1min', '5min', '15min', '30min', '60min',
'daily',
 |              'weekly', 'monthly' (default 'daily')
 |          time_period:  How many data points to average (default 20)
 |          series_type:  The desired price type in the time series. Four types
 |              are supported: 'close', 'open', 'high', 'low' (default 'close')
 |          fastkperiod:  The time period of the fastk moving average. Positive
 |              integers are accepted (default=None)
 |          fastdperiod:  The time period of the fastd moving average. Positive
 |              integers are accepted (default=None)
 |          fastdmatype:  Moving average type for the fastdmatype moving average.
 |              By default, fastmatype=0. Integers 0 - 8 are accepted
 |              (check  down the mappings) or the string containing the math type can
 |              also be used.
 |
 |              * 0 = Simple Moving Average (SMA),
 |              * 1 = Exponential Moving Average (EMA),
 |              * 2 = Weighted Moving Average (WMA),
 |              * 3 = Double Exponential Moving Average (DEMA),
 |              * 4 = Triple Exponential Moving Average (TEMA),
 |              * 5 = Triangular Moving Average (TRIMA),
 |              * 6 = T3 Moving Average,
 |              * 7 = Kaufman Adaptive Moving Average (KAMA),
 |              * 8 = MESA Adaptive Moving Average (MAMA)
 |
 |  get_t3(self, symbol, interval='daily', time_period=20, series_type='close')
 |      Return triple exponential moving average time series in two json
```

```
 |      objects as data and meta_data. It raises ValueError when problems arise
 |
 |      Keyword Arguments:
 |          symbol:  the symbol for the equity we want to get its data
 |          interval:  time interval between two conscutive values,
 |              supported values are '1min', '5min', '15min', '30min', '60min',
'daily',
 |              'weekly', 'monthly' (default 'daily')
 |          time_period:  How many data points to average (default 20)
 |          series_type:  The desired price type in the time series. Four types
 |              are supported: 'close', 'open', 'high', 'low' (default 'close')
 |
 |  get_tema(self, symbol, interval='daily', time_period=20, series_type='close')
 |      Return triple exponential moving average time series in two json
 |      objects as data and meta_data. It raises ValueError when problems arise
 |
 |      Keyword Arguments:
 |          symbol:  the symbol for the equity we want to get its data
 |          interval:  time interval between two conscutive values,
 |              supported values are '1min', '5min', '15min', '30min', '60min',
'daily',
 |              'weekly', 'monthly' (default 'daily')
 |          time_period:  How many data points to average (default 20)
 |          series_type:  The desired price type in the time series. Four types
 |              are supported: 'close', 'open', 'high', 'low' (default 'close')
 |
 |  get_trange(self, symbol, interval='daily')
 |      Return the true range values in two json
 |      objects as data and meta_data. It raises ValueError when problems arise
 |
 |      Keyword Arguments:
 |          symbol:  the symbol for the equity we want to get its data
 |          interval:  time interval between two conscutive values,
 |              supported values are '1min', '5min', '15min', '30min', '60min',
'daily',
 |              'weekly', 'monthly' (default 'daily')
 |
 |  get_trima(self, symbol, interval='daily', time_period=20, series_type='close')
 |      Return triangular moving average time series in two json
 |      objects as data and meta_data. It raises ValueError when problems arise
 |
 |      Keyword Arguments:
 |          symbol:  the symbol for the equity we want to get its data
```

```
 |          interval:  time interval between two conscutive values,
 |              supported values are '1min', '5min', '15min', '30min', '60min',
'daily',
 |              'weekly', 'monthly' (default 'daily')
 |          time_period:  How many data points to average (default 20)
 |          series_type:  The desired price type in the time series. Four types
 |              are supported: 'close', 'open', 'high', 'low' (default 'close')
 |
 |  get_trix(self, symbol, interval='daily', time_period=20, series_type='close')
 |      Return the1-day rate of change of a triple smooth exponential
 |      moving average in two json objects as data and meta_data.
 |      It raises ValueError when problems arise
 |
 |      Keyword Arguments:
 |          symbol:  the symbol for the equity we want to get its data
 |          interval:  time interval between two conscutive values,
 |              supported values are '1min', '5min', '15min', '30min', '60min',
'daily',
 |              'weekly', 'monthly' (default 'daily')
 |          time_period:  How many data points to average (default 20)
 |          series_type:  The desired price type in the time series. Four types
 |              are supported: 'close', 'open', 'high', 'low' (default 'close')
 |
 |  get_ultosc(self, symbol, interval='daily', timeperiod1=None, timeperiod2=None,
timeperiod3=None)
 |      Return the ultimate oscillaror values in two json objects as
 |      data and meta_data. It raises ValueError when problems arise
 |
 |      Keyword Arguments:
 |          symbol:  the symbol for the equity we want to get its data
 |          interval:  time interval between two conscutive values,
 |              supported values are '1min', '5min', '15min', '30min', '60min',
'daily',
 |              'weekly', 'monthly' (default 'daily')
 |          timeperiod1:  The first time period indicator. Positive integers are
 |              accepted. By default, timeperiod1=7
 |          timeperiod2:  The first time period indicator. Positive integers are
 |              accepted. By default, timeperiod2=14
 |          timeperiod3:  The first time period indicator. Positive integers are
 |              accepted. By default, timeperiod3=28
 |
 |  get_vwap(self, symbol, interval='5min')
 |      Returns the volume weighted average price (VWAP) for intraday time series.
```

```
 |
 |      Keyword Arguments:
 |          symbol:  the symbol for the equity we want to get its data
 |          interval:  time interval between two conscutive values,
 |              supported values are '1min', '5min', '15min', '30min', '60min'
 |              (default 5min)
 |
 |  get_willr(self, symbol, interval='daily', time_period=20)
 |      Return the Williams' %R (WILLR) values in two json objects as data
 |      and meta_data. It raises ValueError when problems arise
 |
 |      Keyword Arguments:
 |          symbol:  the symbol for the equity we want to get its data
 |          interval:  time interval between two conscutive values,
 |              supported values are '1min', '5min', '15min', '30min', '60min',
'daily',
 |              'weekly', 'monthly' (default 'daily')
 |          time_period:  How many data points to average (default 20)
 |
 |  get_wma(self, symbol, interval='daily', time_period=20, series_type='close')
 |      Return weighted moving average time series in two json objects
 |      as data and meta_data. It raises ValueError when problems arise
 |
 |      Keyword Arguments:
 |          symbol:  the symbol for the equity we want to get its data
 |          interval:  time interval between two conscutive values,
 |              supported values are '1min', '5min', '15min', '30min', '60min',
'daily',
 |              'weekly', 'monthly' (default 'daily')
 |          time_period:  How many data points to average (default 20)
 |          series_type:  The desired price type in the time series. Four types
 |              are supported: 'close', 'open', 'high', 'low' (default 'close')
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from alpha_vantage.alphavantage.AlphaVantage:
 |
 |  map_to_matype(self, matype)
 |      Convert to the alpha vantage math type integer. It returns an
 |      integer correspondent to the type of math to apply to a function. It
 |      raises ValueError if an integer greater than the supported math types
 |      is given.
 |
 |      Keyword Arguments:
```

```
|          matype:  The math type of the alpha vantage api. It accepts
|          integers or a string representing the math type.
|
|              * 0 = Simple Moving Average (SMA),
|              * 1 = Exponential Moving Average (EMA),
|              * 2 = Weighted Moving Average (WMA),
|              * 3 = Double Exponential Moving Average (DEMA),
|              * 4 = Triple Exponential Moving Average (TEMA),
|              * 5 = Triangular Moving Average (TRIMA),
|              * 6 = T3 Moving Average,
|              * 7 = Kaufman Adaptive Moving Average (KAMA),
|              * 8 = MESA Adaptive Moving Average (MAMA)
|
|  set_proxy(self, proxy=None)
|      Set a new proxy configuration
|
|      Keyword Arguments:
|          proxy: Dictionary mapping protocol or protocol and hostname to
|          the URL of the proxy.
|
|  ----------------------------------------------------------------------
|  Data descriptors inherited from alpha_vantage.alphavantage.AlphaVantage:
|
|  __dict__
|      dictionary for instance variables (if defined)
|
|  __weakref__
|      list of weak references to the object (if defined)
```

### *sma50* - 50 day moving average

**- 50 and 200 days are typically used**

```
pretty('indicator.get_sma()')
help(indicator.get_sma)
```

indicator.get_sma()

```
Help on method get_sma in module alpha_vantage.techindicators:

get_sma(symbol, interval='daily', time_period=20, series_type='close') method of
alpha_vantage.techindicators.TechIndicators instance
    Return simple moving average time series in two json objects as data and
```

```
    meta_data. It raises ValueError when problems arise

    Keyword Arguments:
        symbol:  the symbol for the equity we want to get its data
        interval:  time interval between two conscutive values,
            supported values are '1min', '5min', '15min', '30min', '60min', 'daily',
            'weekly', 'monthly' (default 'daily')
        time_period:  How many data points to average (default 20)
        series_type:  The desired price type in the time series. Four types
            are supported: 'close', 'open', 'high', 'low' (default 'close')
```

```
sma50 = indicator.get_sma('MSFT', interval = 'daily', time_period = 50)[0]
```

```
pretty('sma50 meta informtation')
indicator.get_sma('MSFT', interval = 'daily', time_period = 50)[1]
```

sma50 meta informtation

```
{'1: Symbol': 'MSFT',
 '2: Indicator': 'Simple Moving Average (SMA)',
 '3: Last Refreshed': '2023-02-03',
 '4: Interval': 'daily',
 '5: Time Period': 50,
 '6: Series Type': 'close',
 '7: Time Zone': 'US/Eastern'}
```

```
head_tail_horz(sma50, 5, '50 Day SMA for MSFT')
```

50 Day SMA for MSFT

head(5)

| date | SMA |
|---|---|
| 2000-01-11 | 31.34 |
| 2000-01-12 | 31.43 |
| 2000-01-13 | 31.52 |
| 2000-01-14 | 31.65 |
| 2000-01-18 | 31.80 |

tail(5)

| date | SMA |
|---|---|
| 2023-01-30 | 242.01 |
| 2023-01-31 | 242.13 |
| 2023-02-01 | 242.35 |
| 2023-02-02 | 242.82 |
| 2023-02-03 | 243.14 |

***# 50-day slow moving average overview***

```
timeseries_overview(sma50, 'SMA')
```

DataFrame Overview | Primary Metric: SMA

| | measurement |
|---|---|
| total records | 5,804 |
| start date | 01/11/2000 |
| end date | 02/03/2023 |
| total columns | 1 |
| column labels | SMA |
| total missing values | 0 |
| SMA average | 62 |
| SMA std | 77 |
| SMA min | 13 |
| SMA 25% | 19 |
| SMA 50% | 23 |
| SMA 75% | 61 |
| SMA max | 330 |

***# Microsoft closing data for same period***

```
msft = ts.get_daily('MSFT', outputsize = 'full')[0]
```

```
msft_close = msft['4. close']
msft_close = pd.DataFrame(msft_close)
msft_close.columns = ['close']
```

```
timeseries_overview(msft_close, 'close')
```

DataFrame Overview | Primary Metric: close

| | measurement |
|---|---|
| total records | 5,853 |
| start date | 11/01/1999 |
| end date | 02/03/2023 |
| total columns | 1 |
| column labels | close |
| total missing values | 0 |
| close average | 74 |
| close std | 75 |
| close min | 15 |
| close 25% | 27 |

| | measurement |
|---|---|
| close 50% | 43 |
| close 75% | 78 |
| close max | 343 |

### *# Dataframes combined* (removing days with NaN values)

```
msft_close['sma50'] = sma50
```

```
msft = msft_close.loc['2000-01-11':]
```

```
df_overview(msft, title = 'Microsoft & SMA50', fontsize = '16px')
```

Microsoft & SMA50 Columns

| | close | sma50 |
|---|---|---|
| **datatype** | float64 | float64 |
| **missing values** | 0 | 0 |
| **count** | 5,804.00 | 5,804.00 |
| **mean** | 73.81 | 62.04 |
| **std** | 74.84 | 76.76 |
| **min** | 15.15 | 13.18 |
| **25%** | 27.35 | 19.09 |
| **50%** | 42.20 | 22.96 |
| **75%** | 74.19 | 60.60 |
| **max** | 343.11 | 329.66 |

Microsoft & SMA50 Key Points

| | |
|---|---|
| **total rows** | 5,804 |
| **total columns** | 2 |
| **column names** | close, sma50 |
| **index start** | 2023-02-03 00:00:00 |
| **index end** | 2000-01-11 00:00:00 |
| **total missing values** | 0 |

Microsoft & SMA50 Head and Tail

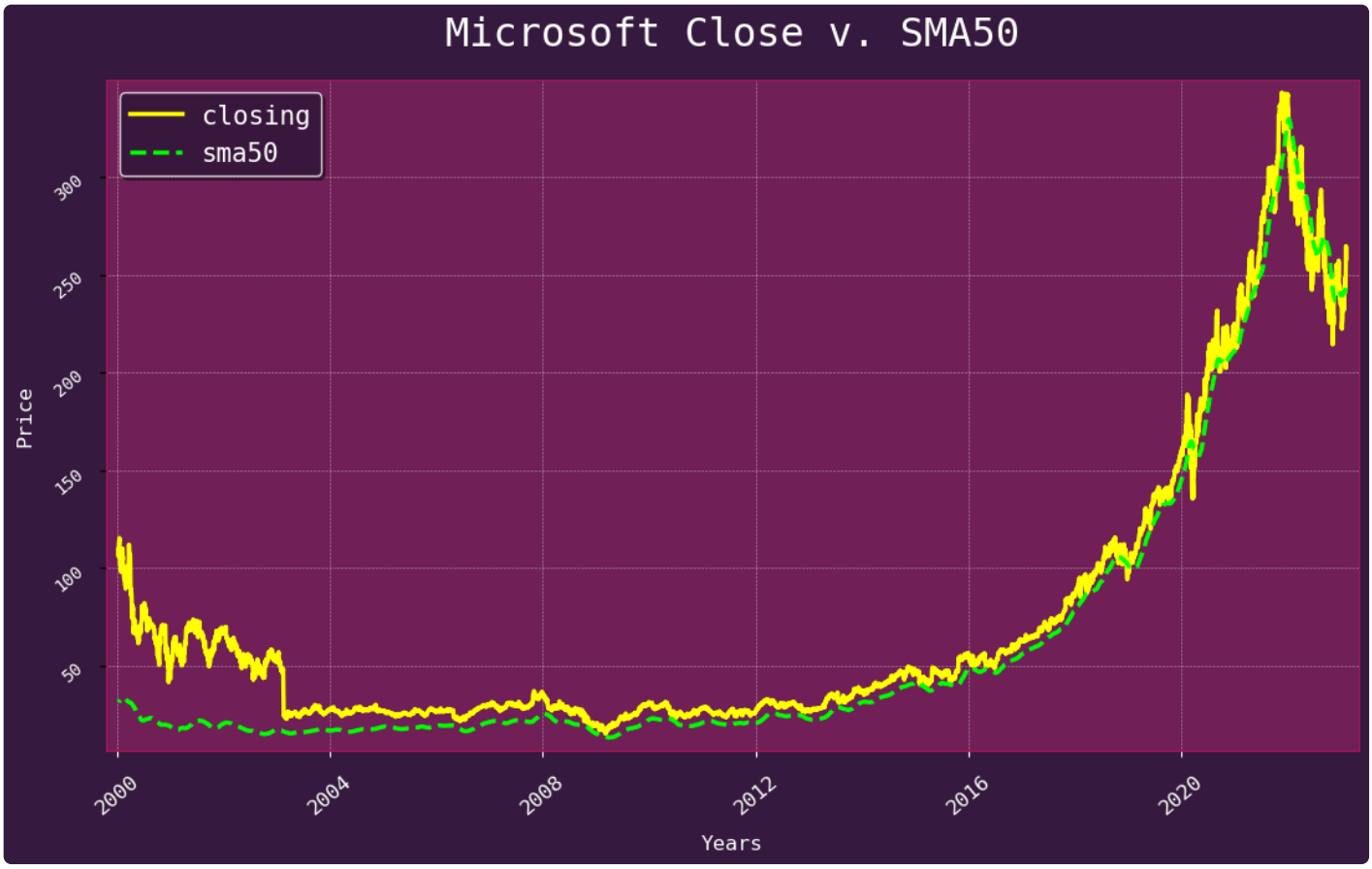
head(3)

| date | close | sma50 |
|---|---|---|
| **2023-02-03** | 258.35 | 243.14 |
| **2023-02-02** | 264.60 | 242.82 |
| **2023-02-01** | 252.75 | 242.35 |

tail(3)

| date | close | sma50 |
|---|---|---|
| **2000-01-13** | 107.81 | 31.52 |
| **2000-01-12** | 105.81 | 31.43 |
| **2000-01-11** | 109.37 | 31.34 |

```
plot_lines([(msft.close, 'closing', 'yellow'),
            (msft.sma50, 'sma50', 'lime')],
           title = 'Microsoft Close v. SMA50',
           xlabel = 'Years',
           ylabel = 'Price')
```




## Bollinger Bands

**- Middle Band is the same as the SMA50**
**- Lower is minus 2 std from the SMA**
**- Upper is plus 2 std from the SMA**

```
pretty('indicator.get_bbands()')
help(indicator.get_bbands)
```

indicator.get_bbands()

```
Help on method get_bbands in module alpha_vantage.techindicators:

get_bbands(symbol, interval='daily', time_period=20, series_type='close', nbdevup=None,
nbdevdn=None, matype=None) method of alpha_vantage.techindicators.TechIndicators
instance
    Return the bollinger bands values in two
    json objects as data and meta_data. It raises ValueError when problems
    arise

    Keyword Arguments:
        symbol:  the symbol for the equity we want to get its data
        interval:  time interval between two conscutive values,
            supported values are '1min', '5min', '15min', '30min', '60min', 'daily',
            'weekly', 'monthly' (default 'daily')
```

```
        time_period:  Number of data points used to calculate each BBANDS value.
            Positive integers are accepted (e.g., time_period=60, time_period=200)
            (default=20)
        series_type:  The desired price type in the time series. Four types
            are supported: 'close', 'open', 'high', 'low' (default 'close')
        nbdevup:  The standard deviation multiplier of the upper band. Positive
            integers are accepted as default (default=2)
        nbdevdn:  The standard deviation multiplier of the lower band. Positive
            integers are accepted as default (default=2)
        matype :  Moving average type. By default, matype=0.
            Integers 0 - 8 are accepted (check  down the mappings) or the string
            containing the math type can also be used.

            * 0 = Simple Moving Average (SMA),
            * 1 = Exponential Moving Average (EMA),
            * 2 = Weighted Moving Average (WMA),
            * 3 = Double Exponential Moving Average (DEMA),
            * 4 = Triple Exponential Moving Average (TEMA),
            * 5 = Triangular Moving Average (TRIMA),
            * 6 = T3 Moving Average,
            * 7 = Kaufman Adaptive Moving Average (KAMA),
            * 8 = MESA Adaptive Moving Average (MAMA)
```

```
bollinger = indicator.get_bbands('MSFT', interval = 'daily', time_period = 50)[0]
```

```
pretty('bollinger meta informtation')
indicator.get_bbands('MSFT', interval = 'daily', time_period = 50)[1]
```

bollinger meta informtation

```
{'1: Symbol': 'MSFT',
 '2: Indicator': 'Bollinger Bands (BBANDS)',
 '3: Last Refreshed': '2023-02-03',
 '4: Interval': 'daily',
 '5: Time Period': 50,
 '6.1: Deviation multiplier for upper band': 2,
 '6.2: Deviation multiplier for lower band': 2,
 '6.3: MA Type': 0,
 '7: Series Type': 'close',
 '8: Time Zone': 'US/Eastern Time'}
```

```
head_tail_horz(bollinger, 5, "Bollinger Bands")
```

Bollinger Bands

head(5)

| date | Real Upper Band | Real Middle Band | Real Lower Band |
|---|---|---|---|
| **2023-02-03** | 260.65 | 243.14 | 225.64 |
| **2023-02-02** | 259.78 | 242.82 | 225.86 |
| **2023-02-01** | 258.13 | 242.35 | 226.57 |
| **2023-01-31** | 257.62 | 242.13 | 226.63 |
| **2023-01-30** | 257.42 | 242.01 | 226.59 |

tail(5)

| date | Real Upper Band | Real Middle Band | Real Lower Band |
|---|---|---|---|
| **2000-01-18** | 39.26 | 31.80 | 24.33 |
| **2000-01-14** | 39.06 | 31.65 | 24.24 |
| **2000-01-13** | 38.90 | 31.52 | 24.14 |
| **2000-01-12** | 38.81 | 31.43 | 24.04 |
| **2000-01-11** | 38.74 | 31.34 | 23.94 |

***# combining with Microsoft close and SMA50***

```
msft_full = msft.merge(bollinger,
                       how = 'left',
                       left_on = 'date',
                       right_on = 'date').drop(columns = ['Real Middle Band'])
msft_full.columns = ['close', 'sma50', 'upper_bband', 'lower_bband']
```

```
head_tail_vert(msft_full, 3, 'Microsoft Data with Upper & Lower Bollinger Bands')
```

Microsoft Data with Upper & Lower Bollinger Bands: head(3)

| date | close | sma50 | upper_bband | lower_bband |
|---|---|---|---|---|
| **2023-02-03** | 258.35 | 243.14 | 260.65 | 225.64 |
| **2023-02-02** | 264.60 | 242.82 | 259.78 | 225.86 |
| **2023-02-01** | 252.75 | 242.35 | 258.13 | 226.57 |

Microsoft Data with Upper & Lower Bollinger Bands: tail(3)

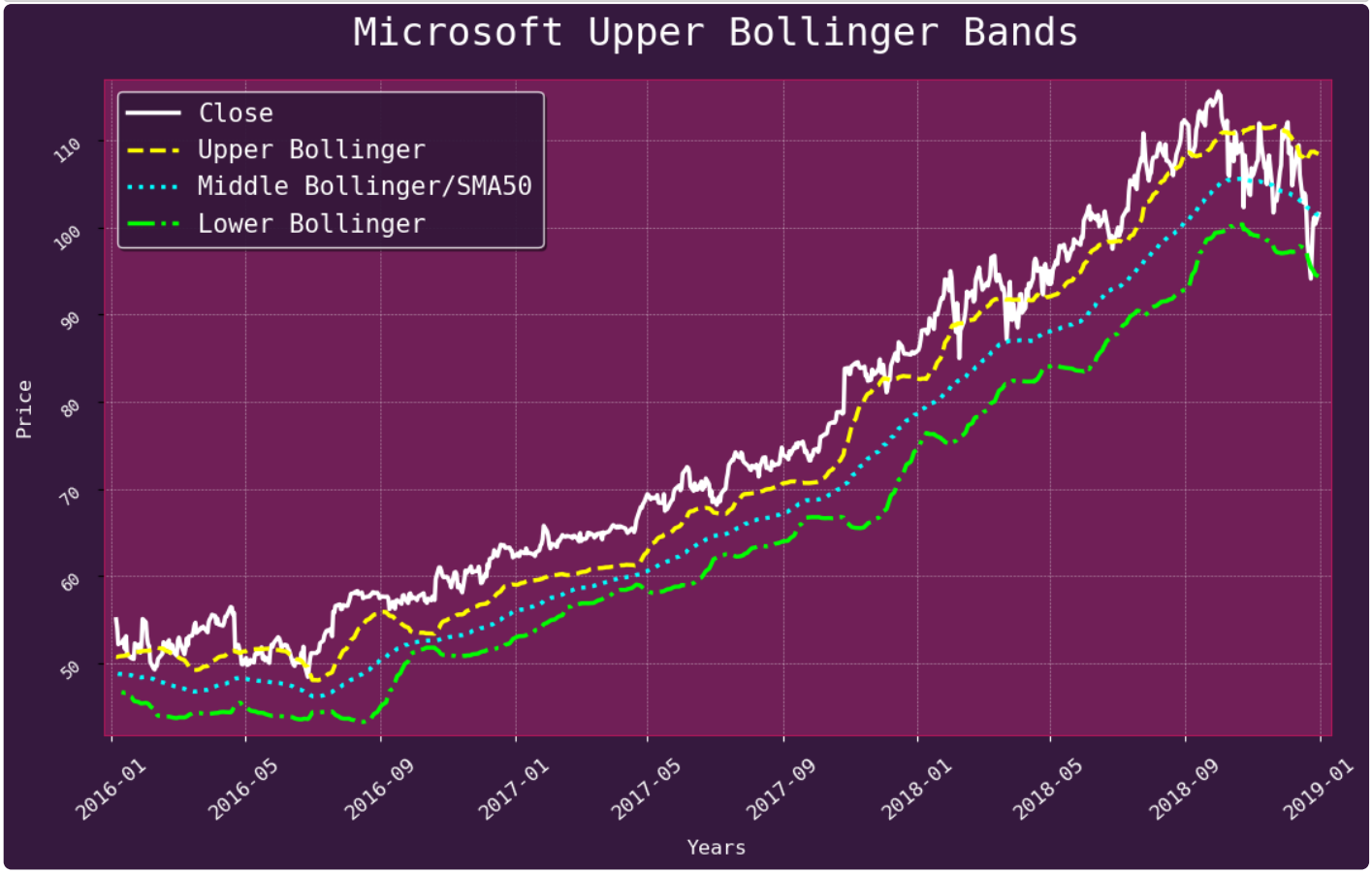
| date | close | sma50 | upper_bband | lower_bband |
|---|---|---|---|---|
| **2000-01-13** | 107.81 | 31.52 | 38.90 | 24.14 |
| **2000-01-12** | 105.81 | 31.43 | 38.81 | 24.04 |

| date | close | sma50 | upper_bband | lower_bband |
|---|---|---|---|---|
| 2000-01-11 | 109.37 | 31.34 | 38.74 | 23.94 |

*Bands indicate volatility:*

- if bands are close together, it indicates low volatility
- if they are farther apart, it indicates higher volatility
- if current price is above the upper band, signals selling
- if current price is below the lower band, signals buying

```
plot_lines([(msft_full.loc['2016-01-05':'2018-12-31'].close,
             'Close', 'white'),
            (msft_full.loc['2016-01-05':'2018-12-31'].upper_bband,
             'Upper Bollinger', 'yellow'),
            (msft_full.loc['2016-01-05':'2018-12-31'].sma50,
             'Middle Bollinger/SMA50', 'cyan'),
            (msft_full.loc['2016-01-05':'2018-12-31'].lower_bband,
             'Lower Bollinger', 'lime')],
           title = 'Microsoft Upper Bollinger Bands',
           xlabel = 'Years',
           ylabel = 'Price')
```




*macd* - moving average convergence divergence

(from Investopedia)

Moving average convergence/divergence (MACD, or MAC-D) is a trend-following momentum indicator that shows the relationship between two exponential moving averages (EMAs) of a security's price. The MACD line is calculated by subtracting the 26-period EMA from the 12-period EMA.

The result of that calculation is the MACD line. A nine-day EMA of the MACD line is called the signal line, which is then plotted on top of the MACD line, which can function as a trigger for buy or sell signals. Traders may buy the security when the MACD line crosses above the signal line and sell—or short—the security when the MACD line crosses below the signal line. MACD indicators can be interpreted in several ways, but the more common methods are crossovers, divergences, and rapid rises/falls.

```
pretty('indicator.get_macd()')
help(indicator.get_macd)
```

indicator.get_macd()

```
Help on method get_macd in module alpha_vantage.techindicators:

get_macd(symbol, interval='daily', series_type='close', fastperiod=None,
slowperiod=None, signalperiod=None) method of
alpha_vantage.techindicators.TechIndicators instance
    Return the moving average convergence/divergence time series in two
    json objects as data and meta_data. It raises ValueError when problems
    arise

    Keyword Arguments:
        symbol:  the symbol for the equity we want to get its data
        interval:  time interval between two conscutive values,
            supported values are '1min', '5min', '15min', '30min', '60min', 'daily',
            'weekly', 'monthly' (default 'daily'
        series_type:  The desired price type in the time series. Four types
            are supported: 'close', 'open', 'high', 'low' (default 'close')
        fastperiod:  Positive integers are accepted (default=None)
        slowperiod:  Positive integers are accepted (default=None)
        signalperiod:  Positive integers are accepted (default=None)
```

```
macd = indicator.get_macd('MSFT', interval = 'daily')[0]
```

```
pretty('macd meta informtation')
indicator.get_macd('MSFT', interval = 'daily')[1]
```

macd meta informtation

```
{'1: Symbol': 'MSFT',
```

```
'2: Indicator': 'Moving Average Convergence/Divergence (MACD)',
'3: Last Refreshed': '2023-02-03',
'4: Interval': 'daily',
'5.1: Fast Period': 12,
'5.2: Slow Period': 26,
'5.3: Signal Period': 9,
'6: Series Type': 'close',
'7: Time Zone': 'US/Eastern'}
```

```
head_tail_horz(macd, 5, 'Moving Average Convergence Divergence')
```

Moving Average Convergence Divergence

head(5)

| date | MACD | MACD_Signal | MACD_Hist |
|---|---|---|---|
| **2023-02-03** | 4.56 | 2.00 | 2.56 |
| **2023-02-02** | 3.95 | 1.36 | 2.59 |
| **2023-02-01** | 2.48 | 0.71 | 1.77 |
| **2023-01-31** | 1.78 | 0.26 | 1.51 |
| **2023-01-30** | 1.36 | -0.12 | 1.47 |

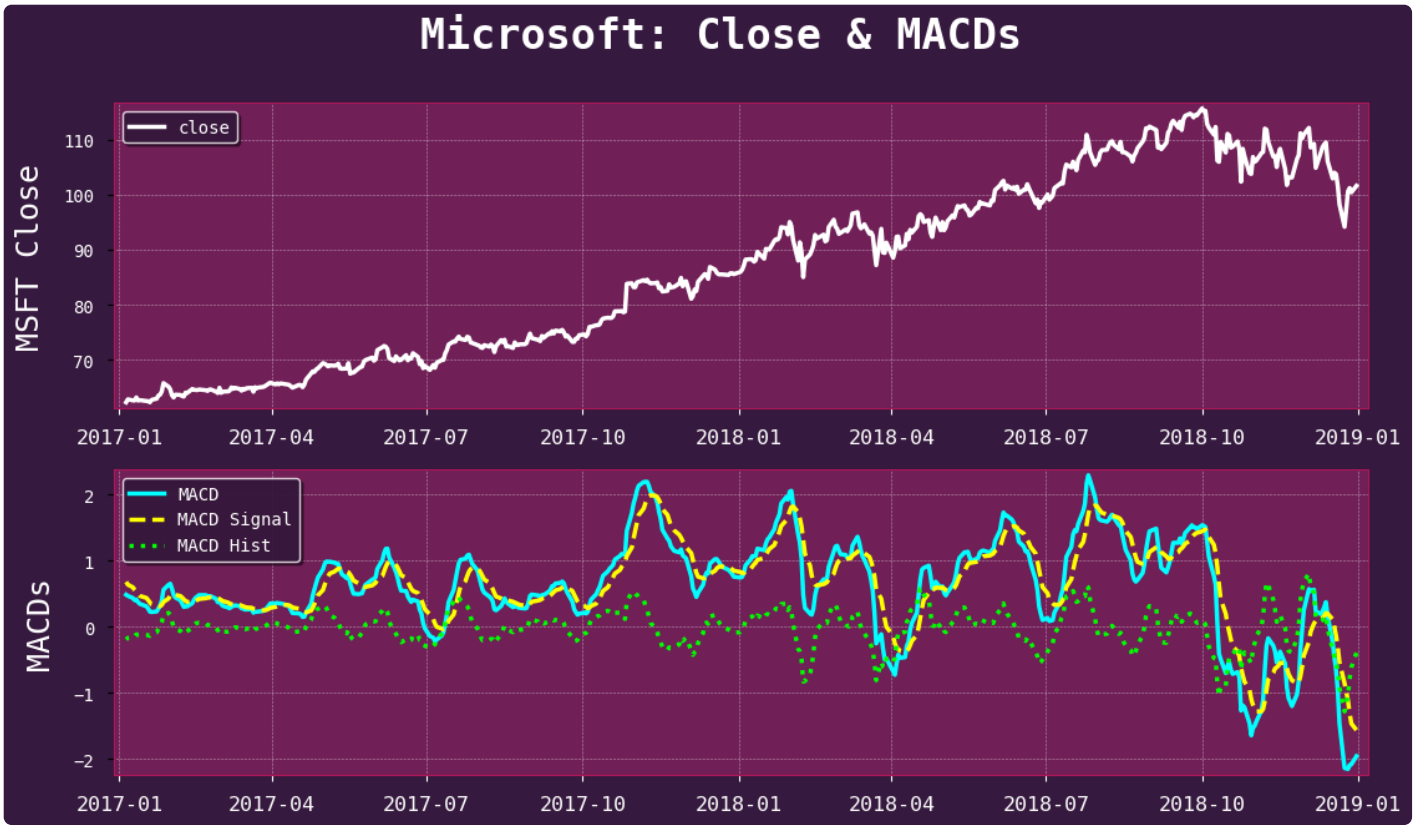
tail(5)

| date | MACD | MACD_Signal | MACD_Hist |
|---|---|---|---|
| **1999-12-23** | 2.24 | 1.52 | 0.71 |
| **1999-12-22** | 2.13 | 1.34 | 0.79 |
| **1999-12-21** | 1.97 | 1.15 | 0.82 |
| **1999-12-20** | 1.78 | 0.94 | 0.84 |
| **1999-12-17** | 1.61 | 0.73 | 0.88 |

***# combining with Microsoft close and MACDs***

```
msft_full = msft_full.merge(macd,
                        how = 'left',
                        left_on = 'date',
                        right_on = 'date')
```

```
plt.style.use('pinks.mplstyle')
fig, axs = plt.subplots(2)
fig.suptitle('Vertically stacked subplots')
axs[0].plot(msft_full.loc['2017-01-05':'2018-12-31'].close,
             label = 'close', color = 'white');
axs[0].set_ylabel('MSFT Close')
axs[0].legend(loc = 2, fontsize = 10);
axs[1].plot(msft_full.loc['2017-01-05':'2018-12-31'].MACD,
            label = 'MACD', color = 'cyan');
axs[1].plot(msft_full.loc['2017-01-05':'2018-12-31'].MACD_Signal,
             label = 'MACD Signal', color = 'yellow');
axs[1].plot(msft_full.loc['2017-01-05':'2018-12-31'].MACD_Hist,
             label = 'MACD Hist', color = 'lime');
axs[1].legend(loc = 2, fontsize = 10);
axs[1].set_ylabel('MACDs');
plt.suptitle('Microsoft: Close & MACDs');
```




| Top | Getting Data | Splits & Dividends | The Datetime Index | Frequency & Intervals | Technical Indicators | Foreign Exchange | Cryptocurrencies |

# Currencies / Foreign Exchanges

```
from alpha_vantage.foreignexchange import ForeignExchange
```

```
foreign = ForeignExchange(key = api_key, output_format = 'pandas')
```

```
pretty('ForeignExchange()')
help(foreign)
```

ForeignExchange()

```
Help on ForeignExchange in module alpha_vantage.foreignexchange object:

class ForeignExchange(alpha_vantage.alphavantage.AlphaVantage)
 |  ForeignExchange(*args, **kwargs)
 |
 |  Realtime currency exchange rates for physical and digital currencies.
 |
 |  Method resolution order:
 |      ForeignExchange
```

```
 |      alpha_vantage.alphavantage.AlphaVantage
 |      builtins.object
 |
 |  Methods defined here:
 |
 |  __init__(self, *args, **kwargs)
 |      Inherit AlphaVantage base class with its default arguments
 |
 |  get_currency_exchange_daily(self, from_symbol, to_symbol, outputsize='compact')
 |      Returns the daily exchange rate for any pair of physical
 |      currency (e.g., EUR) or physical currency (e.g., USD).
 |
 |      Keyword Arguments:
 |          from_symbol: The currency you would like to get the exchange rate
 |              for.
 |              For example: from_symbol=EUR or from_symbol=USD.
 |          to_symbol: The destination currency for the exchange rate.
 |              For example: to_symbol=USD or to_symbol=JPY.
 |          outputsize:  The size of the call, supported values are
 |              'compact' and 'full; the first returns the last 100 points in the
 |              data series, and 'full' returns the full-length daily times
 |              series, commonly above 1MB (default 'compact')
 |
 |  get_currency_exchange_intraday(self, from_symbol, to_symbol, interval='15min',
outputsize='compact')
 |      Returns the intraday exchange rate for any pair of physical
 |      currency (e.g., EUR) or physical currency (e.g., USD).
 |
 |      Keyword Arguments:
 |          from_symbol: The currency you would like to get the exchange rate
 |              for.
 |              For example: from_currency=EUR or from_currency=USD.
 |          to_symbol: The destination currency for the exchange rate.
 |              For example: to_currency=USD or to_currency=JPY.
 |          interval:  time interval between two conscutive values,
 |              supported values are '1min', '5min', '15min', '30min', '60min'
 |              (default '15min')
 |          outputsize:  The size of the call, supported values are
 |              'compact' and 'full; the first returns the last 100 points in the
 |              data series, and 'full' returns the full-length intraday times
 |              series, commonly above 1MB (default 'compact')
 |
 |  get_currency_exchange_monthly(self, from_symbol, to_symbol, outputsize='compact')
```

```
|      Returns the monthly exchange rate for any pair of physical
|      currency (e.g., EUR) or physical currency (e.g., USD).
|
|      Keyword Arguments:
|          from_symbol: The currency you would like to get the exchange rate
|              for.
|              For example: from_symbol=EUR or from_symbol=USD.
|          to_symbol: The destination currency for the exchange rate.
|              For example: to_symbol=USD or to_symbol=JPY.
|          interval:  time interval between two conscutive values,
|              supported values are '1min', '5min', '15min', '30min', '60min'
|              (default '15min')
|          outputsize:  The size of the call, supported values are
|              'compact' and 'full; the first returns the last 100 points in the
|              data series, and 'full' returns the full-length monthly times
|              series, commonly above 1MB (default 'compact')
|
|  get_currency_exchange_rate(self, from_currency, to_currency)
|      Returns the realtime exchange rate for any pair of physical
|      currency (e.g., EUR) or physical currency (e.g., USD).
|
|      Keyword Arguments:
|          from_currency: The currency you would like to get the exchange rate
|          for. It can either be a physical currency or digital/crypto currency.
|          For example: from_currency=USD or from_currency=BTC.
|          to_currency: The destination currency for the exchange rate.
|          It can either be a physical currency or digital/crypto currency.
|          For example: to_currency=USD or to_currency=BTC.
|
|  get_currency_exchange_weekly(self, from_symbol, to_symbol, outputsize='compact')
|      Returns the weekly exchange rate for any pair of physical
|      currency (e.g., EUR) or physical currency (e.g., USD).
|
|      Keyword Arguments:
|          from_symbol: The currency you would like to get the exchange rate
|              for.
|              For example: from_symbol=EUR or from_symbol=USD.
|          to_symbol: The destination currency for the exchange rate.
|              For example: to_symbol=USD or to_symbol=JPY.
|          outputsize:  The size of the call, supported values are
|              'compact' and 'full; the first returns the last 100 points in the
|              data series, and 'full' returns the full-length weekly times
|              series, commonly above 1MB (default 'compact')
```

```
|
|  ----------------------------------------------------------------------
|  Methods inherited from alpha_vantage.alphavantage.AlphaVantage:
|
|  map_to_matype(self, matype)
|      Convert to the alpha vantage math type integer. It returns an
|      integer correspondent to the type of math to apply to a function. It
|      raises ValueError if an integer greater than the supported math types
|      is given.
|
|      Keyword Arguments:
|          matype:  The math type of the alpha vantage api. It accepts
|          integers or a string representing the math type.
|
|              * 0 = Simple Moving Average (SMA),
|              * 1 = Exponential Moving Average (EMA),
|              * 2 = Weighted Moving Average (WMA),
|              * 3 = Double Exponential Moving Average (DEMA),
|              * 4 = Triple Exponential Moving Average (TEMA),
|              * 5 = Triangular Moving Average (TRIMA),
|              * 6 = T3 Moving Average,
|              * 7 = Kaufman Adaptive Moving Average (KAMA),
|              * 8 = MESA Adaptive Moving Average (MAMA)
|
|  set_proxy(self, proxy=None)
|      Set a new proxy configuration
|
|      Keyword Arguments:
|          proxy: Dictionary mapping protocol or protocol and hostname to
|          the URL of the proxy.
|
|  ----------------------------------------------------------------------
|  Data descriptors inherited from alpha_vantage.alphavantage.AlphaVantage:
|
|  __dict__
|      dictionary for instance variables (if defined)
|
|  __weakref__
|      list of weak references to the object (if defined)
```

***# Euro to USD***

```
eur_usd = foreign.get_currency_exchange_daily('USD', 'EUR', outputsize = 'full')[0]
```

```
pretty('foreign.get_currency_exchange_daily()')
help(foreign.get_currency_exchange_daily)
```

foreign.get_currency_exchange_daily()

```
Help on method get_currency_exchange_daily in module alpha_vantage.foreignexchange:

get_currency_exchange_daily(from_symbol, to_symbol, outputsize='compact') method of
alpha_vantage.foreignexchange.ForeignExchange instance
    Returns the daily exchange rate for any pair of physical
    currency (e.g., EUR) or physical currency (e.g., USD).

    Keyword Arguments:
        from_symbol: The currency you would like to get the exchange rate
            for.
            For example: from_symbol=EUR or from_symbol=USD.
        to_symbol: The destination currency for the exchange rate.
            For example: to_symbol=USD or to_symbol=JPY.
        outputsize:  The size of the call, supported values are
            'compact' and 'full; the first returns the last 100 points in the
            data series, and 'full' returns the full-length daily times
            series, commonly above 1MB (default 'compact')
```

```
pretty('usd_eur meta information')
foreign.get_currency_exchange_daily('USD', 'EUR', outputsize = 'full')[1]
```

usd_eur meta information

```
{'1. Information': 'Forex Daily Prices (open, high, low, close)',
 '2. From Symbol': 'USD',
 '3. To Symbol': 'EUR',
 '4. Output Size': 'Full size',
 '5. Last Refreshed': '2023-02-03 21:55:00',
 '6. Time Zone': 'UTC'}
```

```
head_tail_horz(eur_usd, 3, 'Euro to USD')
```

Euro to USD

head(3)

| date | 1. open | 2. high | 3. low | 4. close |
|---|---|---|---|---|
| **2023-02-03** | 0.92 | 0.93 | 0.91 | 0.93 |
| **2023-02-02** | 0.91 | 0.92 | 0.91 | 0.92 |
| **2023-02-01** | 0.92 | 0.92 | 0.91 | 0.91 |

tail(3)

| date | 1. open | 2. high | 3. low | 4. close |
|---|---|---|---|---|
| **2014-11-26** | 0.80 | 0.80 | 0.80 | 0.80 |
| **2014-11-25** | 0.80 | 0.81 | 0.80 | 0.80 |
| **2014-11-24** | 0.81 | 0.81 | 0.80 | 0.80 |

***# USD to Euro***

```
usd_eur = foreign.get_currency_exchange_intraday('USD', 'EUR', outputsize = 'full')[0]
```

```
pretty('foreign.get_currency_exchange_intraday()')
help(foreign.get_currency_exchange_intraday)
```

foreign.get_currency_exchange_intraday()

```
Help on method get_currency_exchange_intraday in module alpha_vantage.foreignexchange:

get_currency_exchange_intraday(from_symbol, to_symbol, interval='15min',
outputsize='compact') method of alpha_vantage.foreignexchange.ForeignExchange instance
    Returns the intraday exchange rate for any pair of physical
    currency (e.g., EUR) or physical currency (e.g., USD).

    Keyword Arguments:
        from_symbol: The currency you would like to get the exchange rate
            for.
            For example: from_currency=EUR or from_currency=USD.
        to_symbol: The destination currency for the exchange rate.
            For example: to_currency=USD or to_currency=JPY.
        interval:  time interval between two conscutive values,
            supported values are '1min', '5min', '15min', '30min', '60min'
            (default '15min')
        outputsize:  The size of the call, supported values are
            'compact' and 'full; the first returns the last 100 points in the
            data series, and 'full' returns the full-length intraday times
            series, commonly above 1MB (default 'compact')
```

```
pretty('foreign.usd_eur meta information')
foreign.get_currency_exchange_intraday('USD', 'EUR', outputsize = 'full')[1]
```

foreign.usd_eur meta information

```
{'1. Information': 'FX Intraday (15min) Time Series',
 '2. From Symbol': 'USD',
 '3. To Symbol': 'EUR',
 '4. Last Refreshed': '2023-02-03 21:45:00',
 '5. Interval': '15min',
 '6. Output Size': 'Full size',
 '7. Time Zone': 'UTC'}
```

```
head_tail_horz(usd_eur, 3, 'USD to EUR')
```

USD to EUR

head(3)

| date | 1. open | 2. high | 3. low | 4. close |
|---|---|---|---|---|
| 2023-02-03 | 0.93 | 0.93 | 0.93 | 0.93 |
| 2023-02-03 | 0.93 | 0.93 | 0.93 | 0.93 |
| 2023-02-03 | 0.93 | 0.93 | 0.93 | 0.93 |

tail(3)

| date | 1. open | 2. high | 3. low | 4. close |
|---|---|---|---|---|
| 2023-01-20 | 0.92 | 0.92 | 0.92 | 0.92 |
| 2023-01-20 | 0.92 | 0.92 | 0.92 | 0.92 |
| 2023-01-20 | 0.92 | 0.92 | 0.92 | 0.92 |

***# Euro to USD intraday, 60 min***

```
see(foreign.get_currency_exchange_intraday('EUR', 'USD',
                                           interval = '60min',
                                           outputsize = 'full')[0].head(5),
    'EUR-USD Intraday: 60 min')
```

EUR-USD Intraday: 60 min

| date | 1. open | 2. high | 3. low | 4. close |
|---|---|---|---|---|
| 2023-02-03 | 1.08 | 1.08 | 1.08 | 1.08 |
| 2023-02-03 | 1.08 | 1.08 | 1.08 | 1.08 |
| 2023-02-03 | 1.08 | 1.08 | 1.08 | 1.08 |
| 2023-02-03 | 1.08 | 1.08 | 1.08 | 1.08 |
| 2023-02-03 | 1.09 | 1.09 | 1.08 | 1.08 |

***# USD to Euro intraday, 1 min***

```
see(foreign.get_currency_exchange_intraday('USD', 'EUR',
                                           interval = '1min',
                                           outputsize = 'full')[0].head(5),
    'USD-EUR Intraday: 1 min')
```

USD-EUR Intraday: 1 min

| date | 1. open | 2. high | 3. low | 4. close |
|---|---|---|---|---|
| **2023-02-03** | 0.93 | 0.93 | 0.93 | 0.93 |
| **2023-02-03** | 0.93 | 0.93 | 0.93 | 0.93 |
| **2023-02-03** | 0.93 | 0.93 | 0.93 | 0.93 |
| **2023-02-03** | 0.93 | 0.93 | 0.93 | 0.93 |
| **2023-02-03** | 0.93 | 0.93 | 0.93 | 0.93 |

## CURRENCIES PLAYGROUND:

```
currencies = pd.read_csv('https://mydatabucky.s3.amazonaws.com/global_currencies.csv')[
```

```
df_overview(currencies)
```

DataFrame Columns

| | country | currency_name | currency_code |
|---|---|---|---|
| **datatype** | object | object | object |
| **missing values** | 0 | 0 | 0 |
| **count** | 272 | 272 | 272 |
| **unique** | 254 | 172 | 174 |
| **top** | SWITZERLAND | Euro | EUR |
| **freq** | 3 | 36 | 36 |

DataFrame Key Points

| | |
|---|---|
| **total rows** | 272 |
| **total columns** | 3 |
| **column names** | country, currency_name, currency_code |
| **index start** | 0 |
| **index end** | 271 |
| **total missing values** | 0 |

DataFrame Head and Tail

head(3)

| | country | currency_name | currency_code |
|---|---|---|---|
| **0** | AFGHANISTAN | Afghani | AFN |
| **1** | ÅLAND ISLANDS | Euro | EUR |
| **2** | ALBANIA | Lek | ALL |

tail(3)

| | country | currency_name | currency_code |
|---|---|---|---|
| **269** | Palladium | Palladium | XPD |
| **270** | Platinum | Platinum | XPT |
| **271** | Silver | Silver | XAG |

```
currencies[currencies.country.str.startswith('R')]
```

| | country | currency_name | currency_code |
|---|---|---|---|
| 193 | RÉUNION | Euro | EUR |
| 194 | ROMANIA | Romanian Leu | RON |
| 195 | RUSSIAN FEDERATION (THE) | Russian Ruble | RUB |
| 196 | RWANDA | Rwanda Franc | RWF |

***exchange_rate(from_currency, to_currency)***

```
def exchange_rate(from_currency, to_currency):
    import requests

    # replace the "demo" apikey below with your own key from https://www.alphavantage.c
    url = f'https://www.alphavantage.co/query?function=CURRENCY_EXCHANGE_RATE&from_curr
    r = requests.get(url)
    data = r.json()

    results = pd.DataFrame(pd.Series(data['Realtime Currency Exchange Rate']))
    results = results.drop(labels = ['1. From_Currency Code', '3. To_Currency Code'],
                           axis = 0)
    results.index = ['From Currency', 'To Currency', 'Exchange Rate', 'Last Refreshed',
                     'Time Zone', 'Bid Price', 'Ask Price']
    results = results.reset_index()

    styles = [{'selector': '.col0',
                            'props': [('font-size', '13px'),
                                      ('text-align', 'left'),
                                      ('padding-right', '15px'),
                                      ('font-weight', 'bold')]},
              {'selector': '.col1',
                     'props': [('font-weight', 'medium'),
                               ('text-align', 'left'),
                               ('font-size', '13px'),
                               ('padding-right', '15px'),
                               ('padding-left', '15px')]},
              {'selector': 'caption',
                     'props': [('font-weight', 'bold'),
                               ('font-size', '23px'),]},
              {'selector': '.row2',
                     'props': [('font-weight', 'bold'),
                               ('background-color', 'cyan'),]}]


    display(results.style.hide(axis = 'index')\
                         .hide(axis = 'columns')\
                         .set_caption('Exchange Results')\
                         .set_table_attributes("style='margin-left: auto;\
                                                       margin-right: auto;'")\
```

```
                    .set_table_styles(styles).format(precision = 2,
                                                     thousands = ','))
```

```
results = exchange_rate('USD', 'EUR')
```

Exchange Results

| | |
|---|---|
| From Currency | United States Dollar |
| To Currency | Euro |
| Exchange Rate | 0.92610000 |
| Last Refreshed | 2023-02-04 19:51:44 |
| Time Zone | UTC |
| Bid Price | 0.92609000 |
| Ask Price | 0.92613000 |

---

| Top | Getting Data | Splits & Dividends | The Datetime Index | Frequency & Intervals | Technical Indicators | Foreign Exchange | Cryptocurrencies |

---

# Cryptocurrencies

- Alpha Vantage offers data on over 500 cryptocurrencies

```
from alpha_vantage.cryptocurrencies import CryptoCurrencies
```

```
crypto = CryptoCurrencies(key = api_key, output_format = 'pandas')
```

```
crypto
```

```
<alpha_vantage.cryptocurrencies.CryptoCurrencies at 0x7fb7b02d6380>
```

### *Bitcoin to USD*

```
BTC = crypto.get_digital_currency_daily(symbol = 'BTC', market = 'USD')
```

```
pretty('BTC to USD meta data')
BTC[1]
```

BTC to USD meta data

```
{'1. Information': 'Daily Prices and Volumes for Digital Currency',
 '2. Digital Currency Code': 'BTC',
 '3. Digital Currency Name': 'Bitcoin',
 '4. Market Code': 'USD',
```

```
 '5. Market Name': 'United States Dollar',
 '6. Last Refreshed': '2023-02-04 00:00:00',
 '7. Time Zone': 'UTC'}
```

```
see(BTC[0].head(5),
   'Most Recent Bitcoin Data')
```

Most Recent Bitcoin Data

| date | 1a. open (USD) | 1b. open (USD) | 2a. high (USD) | 2b. high (USD) | 3a. low (USD) | 3b. low (USD) | 4a. close (USD) | 4b. close (USD) | 5. volume | 6. marke cap (USD |
|---|---|---|---|---|---|---|---|---|---|---|
| 2023-02-04 | 23,431.90 | 23,431.90 | 23,465.15 | 23,465.15 | 23,382.01 | 23,382.01 | 23,435.65 | 23,435.65 | 11,789.15 | 11,789.1 |
| 2023-02-03 | 23,489.33 | 23,489.33 | 23,715.70 | 23,715.70 | 23,204.62 | 23,204.62 | 23,431.90 | 23,431.90 | 332,571.03 | 332,571.0 |
| 2023-02-02 | 23,731.41 | 23,731.41 | 24,255.00 | 24,255.00 | 23,363.27 | 23,363.27 | 23,488.94 | 23,488.94 | 364,177.21 | 364,177.2 |
| 2023-02-01 | 23,125.13 | 23,125.13 | 23,812.66 | 23,812.66 | 22,760.23 | 22,760.23 | 23,732.66 | 23,732.66 | 310,790.42 | 310,790.4 |
| 2023-01-31 | 22,827.38 | 22,827.38 | 23,320.00 | 23,320.00 | 22,714.77 | 22,714.77 | 23,125.13 | 23,125.13 | 264,649.35 | 264,649.3 |

### *Bitcoin to Euro*

```
BTC = crypto.get_digital_currency_daily(symbol = 'BTC', market = 'EUR')
pretty('BTC to EUR meta data')
BTC[1]
```

BTC to EUR meta data

```
{'1. Information': 'Daily Prices and Volumes for Digital Currency',
 '2. Digital Currency Code': 'BTC',
 '3. Digital Currency Name': 'Bitcoin',
 '4. Market Code': 'EUR',
 '5. Market Name': 'Euro',
 '6. Last Refreshed': '2023-02-04 00:00:00',
 '7. Time Zone': 'UTC'}
```

```
see(BTC[0].head(5),
   'Bitcoin to Euro (keeping USD market data)')
```

Bitcoin to Euro (keeping USD market data)

| date | 1a. open (EUR) | 1b. open (USD) | 2a. high (EUR) | 2b. high (USD) | 3a. low (EUR) | 3b. low (USD) | 4a. close (EUR) | 4b. close (USD) | 5. volume | 6. marke cap (USD |
|---|---|---|---|---|---|---|---|---|---|---|
| 2023-02-04 | 21,702.63 | 23,431.90 | 21,733.42 | 23,465.15 | 21,656.42 | 23,382.01 | 21,706.42 | 23,436.00 | 11,788.53 | 11,788.5 |

| date | 1a. open (EUR) | 1b. open (USD) | 2a. high (EUR) | 2b. high (USD) | 3a. low (EUR) | 3b. low (USD) | 4a. close (EUR) | 4b. close (USD) | 5. volume | 6. marke cap (USD |
|---|---|---|---|---|---|---|---|---|---|---|
| 2023-02-03 | 21,755.82 | 23,489.33 | 21,965.48 | 23,715.70 | 21,492.12 | 23,204.62 | 21,702.63 | 23,431.90 | 332,571.03 | 332,571.0 |
| 2023-02-02 | 21,980.03 | 23,731.41 | 22,464.98 | 24,255.00 | 21,639.06 | 23,363.27 | 21,755.46 | 23,488.94 | 364,177.21 | 364,177.2 |
| 2023-02-01 | 21,418.50 | 23,125.13 | 22,055.29 | 23,812.66 | 21,080.53 | 22,760.23 | 21,981.19 | 23,732.66 | 310,790.42 | 310,790.4 |
| 2023-01-31 | 21,142.72 | 22,827.38 | 21,598.98 | 23,320.00 | 21,038.42 | 22,714.77 | 21,418.50 | 23,125.13 | 264,649.35 | 264,649.3 |

### *Ethereum*

```
pretty('ETH meta data')
crypto.get_digital_currency_daily(symbol = 'ETH', market = 'USD')[1]
```

ETH meta data

```
{'1. Information': 'Daily Prices and Volumes for Digital Currency',
 '2. Digital Currency Code': 'ETH',
 '3. Digital Currency Name': 'Ethereum',
 '4. Market Code': 'USD',
 '5. Market Name': 'United States Dollar',
 '6. Last Refreshed': '2023-02-04 00:00:00',
 '7. Time Zone': 'UTC'}
```

```
see(crypto.get_digital_currency_daily(symbol = 'ETH', market = 'USD')[0].head(5),
   'Most Current Ethereum Data')
```

Most Current Ethereum Data

| date | 1a. open (USD) | 1b. open (USD) | 2a. high (USD) | 2b. high (USD) | 3a. low (USD) | 3b. low (USD) | 4a. close (USD) | 4b. close (USD) | 5. volume | 6. market cap (USD) |
|---|---|---|---|---|---|---|---|---|---|---|
| 2023-02-04 | 1,663.52 | 1,663.52 | 1,665.95 | 1,665.95 | 1,658.01 | 1,658.01 | 1,663.09 | 1,663.09 | 18,334.72 | 18,334.72 |
| 2023-02-03 | 1,643.12 | 1,643.12 | 1,676.00 | 1,676.00 | 1,625.93 | 1,625.93 | 1,663.52 | 1,663.52 | 464,815.03 | 464,815.03 |
| 2023-02-02 | 1,641.67 | 1,641.67 | 1,714.68 | 1,714.68 | 1,626.85 | 1,626.85 | 1,643.12 | 1,643.12 | 629,188.89 | 629,188.89 |
| 2023-02-01 | 1,585.32 | 1,585.32 | 1,647.77 | 1,647.77 | 1,555.18 | 1,555.18 | 1,641.68 | 1,641.68 | 474,617.17 | 474,617.17 |
| 2023-01-31 | 1,566.21 | 1,566.21 | 1,605.18 | 1,605.18 | 1,561.63 | 1,561.63 | 1,585.33 | 1,585.33 | 348,856.68 | 348,856.68 |