Udemy Finance

Part One - **Working with Yahoo Finance**

| Course | Yahoo Finance | YFinance Library | Alexander Hagmann, Instructor |

Contents:


```
# from urllib.request import urlretrieve
# helpers="https://raw.githubusercontent.com/EvanMarie/helpers/main/helpers02.py"
# urlretrieve(helpers, "helpers.py")
from helpers import *
import_all() # imports pandas, numpy, matplotlib, yfinance, et al.
```

Sections: ● Top ● Download & Import ● Historical Data ● Setting Date Range ● High Frenquency ● Splits & Dividends ● Exporting ● Multiple Stocks ● Importing Indexes ● Currency Exchange ● Cryptocurrencies ● Mutual & Exchange Traded Funds ● Treasury Yields ● Ticker Object & Docs ● Stock Fundamentals ● Importing Financials ● Importing Put & Call ● Streaming Real-Time ●

▻ Downloading and Importing Data from Yahoo Finance

<Yahoo Finance Site>

```
import_all()
```

```
microsoft = pd.read_csv('./data/MSFT.csv',
                        index_col = "Date",
                        parse_dates=['Date'],
                        usecols = ["Date", "Close"] )
```

```
head_tail_horz(microsoft, 5, "Microsoft: Last 5 Years")
```

Microsoft: Last 5 Years

head(5)

Date	Close
2022-01-13	304.80
2022-01-14	310.20
2022-01-18	302.65
2022-01-19	303.33
2022-01-20	301.60

tail(5)

Date	Close
2023-01-06	224.93
2023-01-09	227.12
2023-01-10	228.85
2023-01-11	235.77
2023-01-12	238.51

```
microsoft.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 251 entries, 2022-01-13 to 2023-01-12
Data columns (total 1 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Close   251 non-null    float64
dtypes: float64(1)
memory usage: 3.9 KB
```

```
see(microsoft.describe(), 'microsoft.describe()')
```

microsoft.describe()

	Close
count	251.00
mean	266.09
std	24.90
min	214.25
25%	244.72
50%	262.97
75%	287.08
max	315.41

▻ Historical Price and Volume Data from One Stock

⊛ Importing the data for General Electric:

- `yf.download` takes either a string for a ticker or a list of tickers

```
import_all(); import yfinance as yf
```

⊛ GE Head and Tail

```
ticker = "GE"
ge = yf.download(ticker)

head_tail_horz(ge, 5, title="General Electric")
```

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

General Electric

head(5)

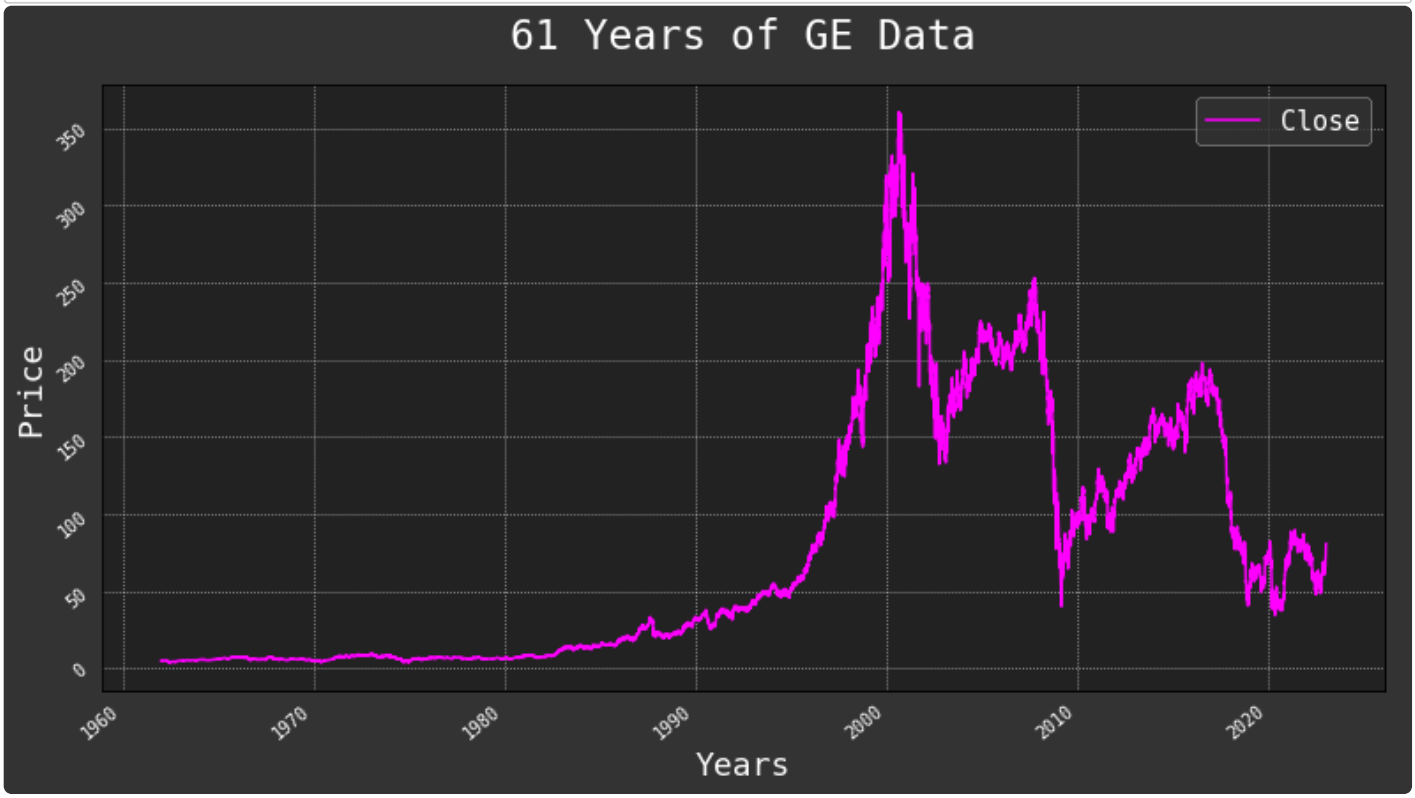
Date	Open	High	Low	Close	Adj Close	Volume
1962-01-02	4.69	4.77	4.64	4.68	0.79	345,317
1962-01-03	4.65	4.65	4.61	4.63	0.78	236,606
1962-01-04	4.63	4.67	4.53	4.57	0.77	294,159
1962-01-05	4.57	4.58	4.38	4.46	0.75	436,442
1962-01-08	4.46	4.46	4.32	4.45	0.75	495,593

tail(5)

Date	Open	High	Low	Close	Adj Close	Volume
2023-01-10	72.20	75.37	72.20	75.27	75.27	9,255,500
2023-01-11	75.82	77.70	75.51	77.69	77.69	10,159,300
2023-01-12	77.88	79.21	77.29	78.86	78.86	11,498,700
2023-01-13	78.78	80.60	78.50	80.20	80.20	11,044,800
2023-01-17	79.92	80.66	79.76	80.49	80.49	7,000,695

⊛ Plot of GE Data

```
fancy_plot(ge.Close.to_frame(), kind = "line",
           cmap='spring', title="61 Years of GE Data",
           ylabel = "Price", xlabel = "Years")
```



Sections: ● Top ● Download & Import ● Historical Data ● Setting Date Range ● High Frenquency ● Splits & Dividends ● Exporting ● Multiple Stocks ● Importing Indexes ● Currency Exchange ● Cryptocurrencies ● Mutual & Exchange Traded Funds ● Treasury Yields ● Ticker Object & Docs ● Stock Fundamentals ● Importing Financials ● Importing Put & Call ● Streaming Real-Time ●

▻ Setting Date Range

`yf.download()` & Date Range

- This can be passed as the `start` and end
- `yf.download()` - has a `period` parameter for specifying date range
- passing `ytd` gets the data for the current year to the current day

> - `period : str`
>   `Valid periods: 1d,5d,1mo,3mo,6mo,1y,2y,5y,10y,ytd,max`
>   `Either Use period parameter or use start and end`

For **lower frequency periods**, i.e. longer periods than 1 day, it is best to download the daily data from YFinance and condense the data manually within Pandas.

```
import_all()
```

```
see(yf.download(ticker, period = 'ytd'), 'yf.download(ticker, period = "ytd")')
```

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

yf.download(ticker, period = "ytd")

Date	Open	High	Low	Close	Adj Close	Volume
2023-01-03	65.61	66.42	65.21	66.34	66.34	8204933
2023-01-04	68.41	70.20	66.75	70.20	70.20	16784600
2023-01-05	69.90	71.55	69.00	71.29	71.29	12770200
2023-01-06	72.01	72.33	70.75	71.94	71.94	10389000
2023-01-09	72.00	73.91	71.96	72.67	72.67	7572900
2023-01-10	72.20	75.37	72.20	75.27	75.27	9255500
2023-01-11	75.82	77.70	75.51	77.69	77.69	10159300
2023-01-12	77.88	79.21	77.29	78.86	78.86	11498700
2023-01-13	78.78	80.60	78.50	80.20	80.20	11044800
2023-01-17	79.92	80.66	79.76	80.49	80.49	7000695

```
see(yf.download(ticker, period = 'ytd'), 'yf.download(ticker, period = "1mo")')
```

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

yf.download(ticker, period = "1mo")

Date	Open	High	Low	Close	Adj Close	Volume
2023-01-03	65.61	66.42	65.21	66.34	66.34	8204933
2023-01-04	68.41	70.20	66.75	70.20	70.20	16784600
2023-01-05	69.90	71.55	69.00	71.29	71.29	12770200
2023-01-06	72.01	72.33	70.75	71.94	71.94	10389000
2023-01-09	72.00	73.91	71.96	72.67	72.67	7572900
2023-01-10	72.20	75.37	72.20	75.27	75.27	9255500
2023-01-11	75.82	77.70	75.51	77.69	77.69	10159300
2023-01-12	77.88	79.21	77.29	78.86	78.86	11498700
2023-01-13	78.78	80.60	78.50	80.20	80.20	11044800
2023-01-17	79.92	80.66	79.76	80.49	80.49	7000695

```
see(yf.download(ticker, period = 'yr'), 'yf.download(ticker, period = "1yr")')
```

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

yf.download(ticker, period = "1yr")

Date	Open	High	Low	Close	Adj Close	Volume
2023-01-17	79.92	80.66	79.76	80.49	80.49	7000695

The **time range** will be from the beginning of 2014 to the end of 2022.

> - start: str
>   Download start date string (YYYY-MM-DD) or _datetime.
>   Default is 1900-01-01
> - end: str
>   Download end date string (YYYY-MM-DD) or _datetime.
>   Default is now

```
GE = yf.download(ticker, start = "2014-01-01", end = "2022-12-31")
```

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

⊛ GE DataFrame

```
head_tail_vert(GE, 5, "GE 2014-2022")
```

GE 2014-2022: head(5)

Date	Open	High	Low	Close	Adj Close	Volume
2014-01-02	167.30	167.78	164.59	165.14	138.33	6903285
2014-01-03	165.26	165.74	164.59	165.02	138.23	4591365
2014-01-06	166.10	166.52	163.03	163.69	137.13	4888755
2014-01-07	165.02	165.08	163.39	163.87	137.28	4117954
2014-01-08	164.23	164.29	162.85	163.39	136.88	4307948

GE 2014-2022: tail(5)

Date	Open	High	Low	Close	Adj Close	Volume
2022-12-23	63.79	64.32	63.20	63.85	63.85	4218845
2022-12-27	64.01	64.86	63.93	64.67	64.67	5430800
2022-12-28	64.79	64.93	63.47	63.99	63.99	4956830
2022-12-29	64.21	65.50	64.01	65.38	65.38	5657408
2022-12-30	64.72	65.60	64.62	65.41	65.41	5287584

⊛ GE.info()

```
GE.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 2266 entries, 2014-01-02 00:00:00-05:00 to 2022-12-30 00:00:00-05:00
Data columns (total 6 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   Open       2266 non-null   float64
 1   High       2266 non-null   float64
 2   Low        2266 non-null   float64
 3   Close      2266 non-null   float64
 4   Adj Close  2266 non-null   float64
 5   Volume     2266 non-null   int64  
dtypes: float64(5), int64(1)
memory usage: 123.9 KB
```

Sections: ● Top ● Download & Import ● Historical Data ● Setting Date Range ● High Frenquency ● Splits & Dividends ● Exporting ● Multiple Stocks ● Importing Indexes ● Currency Exchange ● Cryptocurrencies ● Mutual & Exchange Traded Funds ● Treasury Yields ● Ticker Object & Docs ● Stock Fundamentals ● Importing Financials ● Importing Put & Call ● Streaming Real-Time ●

▻ Setting Higher Frequency

To do this, we use the `interval` parameter for `yf.download()`. The default frequency for `interval` is 1 day.

> - `interval : str`
>   `Valid intervals: 1m,2m,5m,15m,30m,60m,90m,1h,1d,5d,1wk,1mo,3mo`
>   `Intraday data cannot extend last 60 days`

```
import_all()
```

Increments of 1 hour:

```
head_tail_horz(yf.download(ticker, period = "1mo", interval = "1h"),
               5, title = "(period = 1 mo, interval = 1h)", intraday=True)
```

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

(period = 1 mo, interval = 1h)

head(5)

Datetime	Open	High	Low	Close	Adj Close	Volume
2022-12-19 09:30:00-05:00	60.30	61.12	60.27	61.11	61.11	769,439
2022-12-19 10:30:00-05:00	61.10	61.16	60.77	61.05	61.05	494,781
2022-12-19 11:30:00-05:00	61.05	61.07	60.58	60.84	60.84	390,392
2022-12-19 12:30:00-05:00	60.83	61.16	60.73	61.11	61.11	393,848
2022-12-19 13:30:00-05:00	61.11	61.16	60.77	60.77	60.77	535,251

tail(5)

Datetime	Open	High	Low	Close	Adj Close	Volume
2023-01-17 11:30:00-05:00	80.11	80.59	80.09	80.25	80.25	532,721
2023-01-17 12:30:00-05:00	80.27	80.46	80.24	80.39	80.39	339,948
2023-01-17 13:30:00-05:00	80.39	80.54	80.25	80.35	80.35	704,213
2023-01-17 14:30:00-05:00	80.36	80.54	80.31	80.48	80.48	524,414
2023-01-17 15:30:00-05:00	80.48	80.56	80.34	80.49	80.49	1,215,875

Increments of 30 minutes:

```
head_tail_horz(yf.download(ticker, period = "1mo", interval = "30m"),
               5, title = "(period = 1 mo, interval = 30m)", intraday=True)
```

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

(period = 1 mo, interval = 30m)

head(5)

Datetime	Open	High	Low	Close	Adj Close	Volume
2022-12-19 09:30:00-05:00	60.30	61.07	60.27	60.68	60.68	453,008
2022-12-19 10:00:00-05:00	60.67	61.12	60.55	61.11	61.11	316,431
2022-12-19 10:30:00-05:00	61.10	61.11	60.77	60.86	60.86	234,476
2022-12-19 11:00:00-05:00	60.87	61.16	60.82	61.05	61.05	260,305
2022-12-19 11:30:00-05:00	61.05	61.07	60.58	60.64	60.64	260,111

tail(5)

Datetime	Open	High	Low	Close	Adj Close	Volume
2023-01-17 14:00:00-05:00	80.43	80.54	80.25	80.35	80.35	481,659
2023-01-17 14:30:00-05:00	80.36	80.47	80.31	80.38	80.38	227,634
2023-01-17 15:00:00-05:00	80.38	80.54	80.33	80.48	80.48	296,780
2023-01-17 15:30:00-05:00	80.48	80.56	80.34	80.50	80.50	1,215,875
2023-01-17 16:00:00-05:00	80.49	80.49	80.49	80.49	80.49	0

Increments of 5 minutes:

```
head_tail_horz(yf.download(ticker, period = "1mo", interval = "5m"),
               5, title = "(period = 1 mo, interval = 5m)", intraday=True)
```

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

(period = 1 mo, interval = 5m)

head(5)

Datetime	Open	High	Low	Close	Adj Close	Volume
2022-12-19 09:30:00-05:00	60.30	60.78	60.27	60.73	60.73	175,101
2022-12-19 09:35:00-05:00	60.76	61.07	60.65	60.90	60.90	106,490
2022-12-19 09:40:00-05:00	60.88	60.91	60.67	60.70	60.70	35,044
2022-12-19 09:45:00-05:00	60.72	60.72	60.39	60.52	60.52	30,392
2022-12-19 09:50:00-05:00	60.51	60.67	60.51	60.59	60.59	73,081

tail(5)

Datetime	Open	High	Low	Close	Adj Close	Volume
2023-01-17 15:40:00-05:00	80.47	80.53	80.46	80.48	80.48	133,332
2023-01-17 15:45:00-05:00	80.47	80.51	80.46	80.50	80.50	152,468
2023-01-17 15:50:00-05:00	80.51	80.54	80.34	80.40	80.40	206,981
2023-01-17 15:55:00-05:00	80.41	80.54	80.41	80.50	80.50	535,741
2023-01-17 16:00:00-05:00	80.49	80.49	80.49	80.49	80.49	0

Increments of 1 minute:

```
head_tail_horz(yf.download(ticker, period = "5d", interval = "1m"),
               5, title = "(period = 5d, interval = 1m)", intraday=True)
```

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

(period = 5d, interval = 1m)

head(5)

Datetime	Open	High	Low	Close	Adj Close	Volume
2023-01-10 09:30:00-05:00	72.20	72.72	72.20	72.50	72.50	106,241
2023-01-10 09:31:00-05:00	72.59	73.15	72.50	73.10	73.10	40,117
2023-01-10 09:32:00-05:00	73.10	73.42	73.10	73.42	73.42	8,275
2023-01-10 09:33:00-05:00	73.40	73.66	73.36	73.66	73.66	158,498
2023-01-10 09:34:00-05:00	73.70	73.99	73.49	73.55	73.55	68,966

tail(5)

Datetime	Open	High	Low	Close	Adj Close	Volume
2023-01-17 15:56:00-05:00	80.46	80.51	80.46	80.49	80.49	58,583
2023-01-17 15:57:00-05:00	80.49	80.51	80.49	80.49	80.49	45,146
2023-01-17 15:58:00-05:00	80.50	80.54	80.49	80.49	80.49	78,248
2023-01-17 15:59:00-05:00	80.50	80.52	80.42	80.50	80.50	290,298
2023-01-17 16:00:00-05:00	80.49	80.49	80.49	80.49	80.49	0

⊛ GE: 5 day period & 1 minute interval

```
ge51 = yf.download(ticker, period = "5d", interval = "1m")
```

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

```
head_tail_horz(ge51, 5, "GE: 5 day period, 1 minute interval", intraday=True)
```

GE: 5 day period, 1 minute interval

head(5)

Datetime	Open	High	Low	Close	Adj Close	Volume
2023-01-10 09:30:00-05:00	72.20	72.72	72.20	72.50	72.50	106,241
2023-01-10 09:31:00-05:00	72.59	73.15	72.50	73.10	73.10	40,117
2023-01-10 09:32:00-05:00	73.10	73.42	73.10	73.42	73.42	8,275
2023-01-10 09:33:00-05:00	73.40	73.66	73.36	73.66	73.66	158,498
2023-01-10 09:34:00-05:00	73.70	73.99	73.49	73.55	73.55	68,966

tail(5)

Datetime	Open	High	Low	Close	Adj Close	Volume
2023-01-17 15:56:00-05:00	80.46	80.51	80.46	80.49	80.49	58,583
2023-01-17 15:57:00-05:00	80.49	80.51	80.49	80.49	80.49	45,146
2023-01-17 15:58:00-05:00	80.50	80.54	80.49	80.49	80.49	78,248
2023-01-17 15:59:00-05:00	80.50	80.52	80.42	80.50	80.50	290,298
2023-01-17 16:00:00-05:00	80.49	80.49	80.49	80.49	80.49	0

⊛ ge51.describe()

```
see(ge51.describe(), 'ge51.describe() overview')
```

ge51.describe() overview

	Open	High	Low	Close	Adj Close	Volume
count	1,949.00	1,949.00	1,949.00	1,949.00	1,949.00	1,949.00
mean	77.92	77.96	77.89	77.93	77.93	22,511.31
std	2.37	2.37	2.38	2.37	2.37	55,856.67
min	72.20	72.72	72.20	72.50	72.50	0.00
25%	76.47	76.53	76.43	76.49	76.49	7,173.00
50%	78.59	78.62	78.55	78.60	78.60	11,596.00
75%	80.13	80.16	80.10	80.13	80.13	21,180.00
max	80.56	80.66	80.54	80.56	80.56	1,831,539.00

Using `prepost` for 4:30 - 9:30am pre-market trading

- Default setting is False
- There is also a post-market trading session

```
head_tail_horz(yf.download(ticker, prepost=True, period = "5d",
                           interval = "1m").drop(columns = ["Volume"]),
               5, '(prepost=True, period = "5d", interval = "1m")', intraday=True)
```

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

(prepost=True, period = "5d", interval = "1m")

head(5)

Datetime	Open	High	Low	Close	Adj Close
2023-01-10 04:00:00-05:00	72.20	72.20	72.20	72.20	72.20
2023-01-10 07:00:00-05:00	72.92	72.92	72.85	72.85	72.85
2023-01-10 07:05:00-05:00	72.85	72.85	72.85	72.85	72.85
2023-01-10 07:08:00-05:00	72.11	72.21	72.00	72.00	72.00
2023-01-10 07:19:00-05:00	72.19	72.20	72.00	72.00	72.00

tail(5)

Datetime	Open	High	Low	Close	Adj Close
2023-01-17 17:12:00-05:00	80.43	80.43	80.43	80.43	80.43
2023-01-17 17:14:00-05:00	80.43	80.43	80.43	80.43	80.43
2023-01-17 17:16:00-05:00	80.43	80.43	80.43	80.43	80.43
2023-01-17 17:18:00-05:00	80.44	80.44	80.44	80.44	80.44
2023-01-17 17:19:00-05:00	80.49	80.49	80.49	80.49	80.49

Sections: ● Top ● Download & Import ● Historical Data ● Setting Date Range ● High Frenquency ● Splits & Dividends ● Exporting ● Multiple Stocks ● Importing Indexes ● Currency Exchange ● Cryptocurrencies ● Mutual & Exchange Traded Funds ● Treasury Yields ● Ticker Object & Docs ● Stock Fundamentals ● Importing Financials ● Importing Put & Call ● Streaming Real-Time ●

▻ Stock Splits and Dividends

Also called "corporate actions". This is the number one source of mistakes and errors when working with stock price data.

- in some cases, historical stock prices should be adjusted for stock splits and dividends
- in other cases, they should not be adjusted

```
import_all()
```

⊛ **Apple** stock:

- last 10 years of data
- `actions = True` causes the download of stock splits and dividends data as well

```
ticker = 'AAPL'
AAPL = yf.download(ticker, period = "10y", actions = True)
```

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

⊛ Apple stock head and tail:

- In the most recent data, the `tail(5)` below, the adjusted close and close are identical
- But going back in time, toward the `head(5)`, the difference between adjusted close and close increase
- Adjusted close is always lower than close price
- The reason for this is dividends
- Adjusted close prices are backward adjusted for divdends

```
head_tail_vert(AAPL, 5, "AAPL (period = 10y, actions = True)")
```

AAPL (period = 10y, actions = True): head(5)

Date	Open	High	Low	Close	Adj Close	Volume	Dividends	Stock Splits
2013-01-18	17.80	17.94	17.73	17.86	15.36	472922800	0.00	0.00
2013-01-22	18.02	18.14	17.74	18.03	15.50	461546400	0.00	0.00
2013-01-23	18.17	18.39	18.03	18.36	15.79	861509600	0.00	0.00
2013-01-24	16.43	16.63	16.08	16.09	13.84	1460852400	0.00	0.00
2013-01-25	16.13	16.29	15.54	15.71	13.51	1208026400	0.00	0.00

AAPL (period = 10y, actions = True): tail(5)

Date	Open	High	Low	Close	Adj Close	Volume	Dividends	Stock Splits
2023-01-10	130.26	131.26	128.12	130.73	130.73	63896200	0.00	0.00
2023-01-11	131.25	133.51	130.46	133.49	133.49	69458900	0.00	0.00
2023-01-12	133.88	134.26	131.44	133.41	133.41	71379600	0.00	0.00
2023-01-13	132.03	134.92	131.66	134.76	134.76	57758000	0.00	0.00
2023-01-17	134.83	137.29	134.15	135.94	135.94	63453444	0.00	0.00

⊛ Getting the rows where the value in **dividends** is greater than 0:

- Dividends are usually paid out on a quarterly basis
- The value represents the amount per share

```
see(AAPL[AAPL.Dividends > 0].tail(10).drop(columns = ["Stock Splits"]),
    "AAPL - Dates dividends were paid (quarterly):")
```

AAPL - Dates dividends were paid (quarterly):

Date	Open	High	Low	Close	Adj Close	Volume	Dividends
2020-08-07	113.21	113.68	110.29	111.11	109.61	198045600	0.20
2020-11-06	118.32	119.20	116.13	118.69	117.28	114457900	0.20
2021-02-05	137.35	137.42	135.86	136.76	135.34	75693800	0.20
2021-05-07	130.85	131.26	129.48	130.21	129.08	78973300	0.22
2021-08-06	146.35	147.11	145.63	146.14	145.08	54126800	0.22
2021-11-05	151.89	152.20	150.06	151.28	150.41	65463900	0.22
2022-02-04	171.68	174.10	170.68	172.39	171.61	82465400	0.22
2022-05-06	156.01	159.44	154.18	157.28	156.80	116124600	0.23
2022-08-05	163.21	165.85	163.00	165.35	165.08	56697000	0.23
2022-11-04	142.09	142.67	134.38	138.38	138.38	140814800	0.23

Viewing dates directly surrounding the most recent dividend:

- investigating the backward adjustment, difference between close and adjusted close, around the dates that dividends are paid
- The drop in close on the day of dividends is not completely accurate
- **Adjusted close** better reflects the performance of the stock
- Important when the interest is the total return of the stock, the price change plus dividend payments, which are an essential part of the total return
- **Dividends** are immediately reinvested into the stock, which accounts for the **backward adjustment**

```
see(AAPL.loc["2022-11-02" : "2022-11-07"],
'Dates directly surrounding most recent dividend payments:')
```

Dates directly surrounding most recent dividend payments:

Date	Open	High	Low	Close	Adj Close	Volume	Dividends	Stock Splits
2022-11-02	148.95	152.17	145.00	145.03	144.79	93604600	0.00	0.00
2022-11-03	142.06	142.80	138.75	138.88	138.65	97918500	0.00	0.00
2022-11-04	142.09	142.67	134.38	138.38	138.38	140814800	0.23	0.00
2022-11-07	137.11	139.15	135.67	138.92	138.92	83374600	0.00	0.00

`diff()` - calculates difference from one timestamp to the next

```
see(AAPL.loc["2022-10-31" : "2022-11-11"].diff(),
    "diff() for 10-31-22 to 11-11-22")
```

diff() for 10-31-22 to 11-11-22

Date	Open	High	Low	Close	Adj Close	Volume	Dividends	Stock Splits
2022-10-31	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2022-11-01	1.92	1.21	-2.79	-2.69	-2.69	-17,563,900.00	0.00	0.00
2022-11-02	-6.13	-3.28	-4.13	-5.62	-5.61	13,225,300.00	0.00	0.00
2022-11-03	-6.89	-9.37	-6.25	-6.15	-6.14	4,313,900.00	0.00	0.00
2022-11-04	0.03	-0.13	-4.37	-0.50	-0.27	42,896,300.00	0.23	0.00
2022-11-07	-4.98	-3.52	1.29	0.54	0.54	-57,440,200.00	-0.23	0.00
2022-11-08	3.30	2.28	1.82	0.58	0.58	6,533,900.00	0.00	0.00
2022-11-09	-1.91	-2.88	-2.90	-4.63	-4.63	-14,990,700.00	0.00	0.00
2022-11-10	2.74	8.32	4.91	12.00	12.00	43,936,200.00	0.00	0.00
2022-11-11	4.58	3.14	4.87	2.83	2.83	-24,874,300.00	0.00	0.00

➢ Stock Split:

- corporation/company **divides existing shares into multiple shares**
- typically done to boost the **liquidity** of the shares
- high price stock it makes more difficult for small investors to invest

```
see(AAPL[AAPL['Stock Splits'] > 0],'Stock Splits')
```

Stock Splits

Date	Open	High	Low	Close	Adj Close	Volume	Dividends	Stock Splits
2014-06-09	23.17	23.47	22.94	23.42	20.89	301660000	0.00	7.00
2020-08-31	127.58	131.00	126.00	129.04	127.29	225702700	0.00	4.00

Investigating dates surrounding **most recent split - 2020-08-31**:

- Close and adjusted close have already **been adjusted** for stock splits
- Prices are not always backward adjusted on all APIs and sources, which can cause a great deal of confusion

```
see(AAPL.loc['2020-08-28' : '2020-09-01'],
    "Dates surrounding most recent stock split on 2020-08-31")
```

Dates surrounding most recent stock split on 2020-08-31

Date	Open	High	Low	Close	Adj Close	Volume	Dividends	Stock Splits
2020-08-28	126.01	126.44	124.58	124.81	123.11	187630000	0.00	0.00

Date	Open	High	Low	Close	Adj Close	Volume	Dividends	Stock Splits
2020-08-31	127.58	131.00	126.00	129.04	127.29	225702700	0.00	4.00
2020-09-01	132.76	134.80	130.53	134.18	132.36	151948100	0.00	0.00

Sections: ● Top ● Download & Import ● Historical Data ● Setting Date Range ● High Frenquency ● Splits & Dividends ● Exporting ● Multiple Stocks ● Importing Indexes ● Currency Exchange ● Cryptocurrencies ● Mutual & Exchange Traded Funds ● Treasury Yields ● Ticker Object & Docs ● Stock Fundamentals ● Importing Financials ● Importing Put & Call ● Streaming Real-Time ●

▻ Exporting to CSV and Excel

```
import_all()
```

Data = last 5 years of GE

```
ticker = "GE"
GE = yf.download(ticker, period = '5y')
```

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

GE Last 5 Years - Head and Tail

```
head_tail_horz(GE, 5, "GE Last 5 Years")
```

GE Last 5 Years

head(5)

Date	Open	High	Low	Close	Adj Close	Volume
2018-01-18	104.49	104.97	100.64	100.70	96.32	28,687,024
2018-01-19	99.98	100.46	96.20	97.64	93.39	35,795,157
2018-01-22	95.48	98.06	94.88	97.10	92.88	22,350,374
2018-01-23	97.88	102.26	97.70	101.42	97.01	20,522,574
2018-01-24	102.08	104.25	97.58	98.72	94.43	27,813,458

tail(5)

Date	Open	High	Low	Close	Adj Close	Volume
2023-01-10	72.20	75.37	72.20	75.27	75.27	9,255,500
2023-01-11	75.82	77.70	75.51	77.69	77.69	10,159,300
2023-01-12	77.88	79.21	77.29	78.86	78.86	11,498,700
2023-01-13	78.78	80.60	78.50	80.20	80.20	11,044,800
2023-01-17	79.92	80.66	79.76	80.49	80.49	7,000,695

⊛ to_csv()

- exporting the last 5 years of GE data to CSV

```
GE.to_csv("ge_last_5_years.csv")
```

⊛ read_csv()

- creating a datetime index column by parsing dates and setting index

```
GE = pd.read_csv('ge_last_5_years.csv',
                 parse_dates = ["Date"],
                 index_col = 'Date')
```

⊛ to_excel()

```
# GE.to_excel("ge_last_5_years.xlsx")
```

⊛ read_excel()

```
# see(pd.read_excel("ge_last_5_years.xlsx",
#                   parse_dates = ['Date'],
#                   index_col = "Date")
#                  .head(5), "read_excel()")
```

Sections: ● Top ● Download & Import ● Historical Data ● Setting Date Range ● High Frenquency ● Splits & Dividends ● Exporting ● Multiple Stocks ● Importing Indexes ● Currency Exchange ● Cryptocurrencies ● Mutual & Exchange Traded Funds ● Treasury Yields ● Ticker Object & Docs ● Stock Fundamentals ● Importing Financials ● Importing Put & Call ● Streaming Real-Time ●

⊳ Importing Multiple Stocks

```
import_all()
%matplotlib inline
```

Working with 3 separate tickers: GE, AAPL, and META

```
tickers = ["GE", "AAPL", "META"]
```

⊛ GE, AAPL, and META: last 5 years of data

```
stocks_3_5y = yf.download(tickers, period = "5y")
```

```
[**********100%**********]  3 of 3 completed
```

⊛ **Mutli-Indexed Data**: outer index = Adj Close, Close, High, Low, Open, & Volume. On the inner index level are the stock tickers.

```
head_tail_vert(stocks_3_5y, 5, title="GE, AAPL, META 2018-2023")
```

GE, AAPL, META 2018-2023: head(5)

	Adj Close			Close			High			Low				
	AAPL	GE	META	AAPL	GE	META	AAPL	GE	META	AAPL	GE	META	AAPL	GE
Date														
2018-01-18	42.61	96.32	179.80	44.81	100.70	179.80	45.03	104.97	180.98	44.56	100.64	177.08	44.84	104.49
2018-01-19	42.42	93.39	181.29	44.62	97.64	181.29	44.90	100.46	182.37	44.35	96.20	180.17	44.65	99.98
2018-01-22	42.08	92.88	185.37	44.25	97.10	185.37	44.44	98.06	185.39	44.15	94.88	180.41	44.33	95.48
2018-01-23	42.09	97.01	189.35	44.26	101.42	189.35	44.86	102.26	189.55	44.21	97.70	185.55	44.33	97.88
2018-01-24	41.42	94.43	186.55	43.56	98.72	186.55	44.33	104.25	190.66	43.30	97.58	186.52	44.31	102.08

GE, AAPL, META 2018-2023: tail(5)

	Adj Close			Close			High			Low				
	AAPL	GE	META	AAPL	GE	META	AAPL	GE	META	AAPL	GE	META	AAPL	GE
Date														
2023-01-10	130.73	75.27	132.99	130.73	75.27	132.99	131.26	75.37	133.44	128.12	72.20	127.15	130.26	72.20
2023-01-11	133.49	77.69	132.89	133.49	77.69	132.89	133.51	77.70	133.85	130.46	75.51	130.34	131.25	75.82
2023-01-12	133.41	78.86	136.71	133.41	78.86	136.71	134.26	79.21	137.68	131.44	77.29	131.76	133.88	77.88

	Adj Close			Close			High			Low				
	AAPL	GE	META	AAPL	GE	META	AAPL	GE	META	AAPL	GE	META	AAPL	GE
Date														
2023-01-13	134.76	80.20	136.98	134.76	80.20	136.98	134.92	80.60	137.39	131.66	78.50	134.84	132.03	78.78
2023-01-17	135.94	80.49	135.36	135.94	80.49	135.36	137.29	80.66	136.75	134.15	79.76	134.32	134.83	79.92

GE, AAPL, and META: last 5 years of data grouped by ticker

```
stocks_3_5y_ticker = yf.download(tickers, period = "5y", group_by = "Ticker")
```

```
[**********************100%**********************]  3 of 3 completed
```

⊛ **Mutli-Index Switched**: grouping by tickers cause the indices, outer and inner, to be switched. `group_by` is by default set to `columns` .

```
head_tail_vert(stocks_3_5y_ticker, 5,
           title="GE, AAPL, META 2018-2023 grouped by ticker")
```

GE, AAPL, META 2018-2023 grouped by ticker: head(5)

	AAPL						GE							
	Open	High	Low	Close	Adj Close	Volume	Open	High	Low	Close	Adj Close	Volume	Open	
Date														
2018-01-18	44.84	45.03	44.56	44.81	42.61	124773600	104.49	104.97	100.64	100.70	96.32	28687024	178.13	1
2018-01-19	44.65	44.90	44.35	44.62	42.42	129700400	99.98	100.46	96.20	97.64	93.39	35795157	180.85	1
2018-01-22	44.33	44.44	44.15	44.25	42.08	108434400	95.48	98.06	94.88	97.10	92.88	22350374	180.80	1
2018-01-23	44.33	44.86	44.21	44.26	42.09	130756400	97.88	102.26	97.70	101.42	97.01	20522574	186.05	1
2018-01-24	44.31	44.33	43.30	43.56	41.42	204420400	102.08	104.25	97.58	98.72	94.43	27813458	189.89	1

GE, AAPL, META 2018-2023 grouped by ticker: tail(5)

	AAPL						GE							
	Open	High	Low	Close	Adj Close	Volume	Open	High	Low	Close	Adj Close	Volume	Open	
Date														
2023-01-10	130.26	131.26	128.12	130.73	130.73	63896200	72.20	75.37	72.20	75.27	75.27	9255500	127.27	1

	AAPL						GE					
	Open	High	Low	Close	Adj Close	Volume	Open	High	Low	Close	Adj Close	Volume	Open
Date													
2023-01-11	131.25	133.51	130.46	133.49	133.49	69458900	75.82	77.70	75.51	77.69	77.69	10159300	130.96
2023-01-12	133.88	134.26	131.44	133.41	133.41	71379600	77.88	79.21	77.29	78.86	78.86	11498700	133.44
2023-01-13	132.03	134.92	131.66	134.76	134.76	57758000	78.78	80.60	78.50	80.20	80.20	11044800	134.97
2023-01-17	134.83	137.29	134.15	135.94	135.94	63453444	79.92	80.66	79.76	80.49	80.49	7000695	136.18

Retaining the Close data only, therefor not grouping by ticker

```
stocks = yf.download(tickers, period = "5y").Close

plot_by_df(stocks, sort_param="Date",
           cmap="cool", title="GE, AAPL, META - Last 5 Years",
          ylabel = 'prices', xlabel = 'years')
```

```
[**********100%**********]  3 of 3 completed
```

GE, AAPL, META - Last 5 Years (12 samples & overall plot)

```
GridspecLayout(children=(Output(layout=Layout(border='1px solid black',
grid_area='widget001')), Output(layout…
```

Sections: ● Top ● Download & Import ● Historical Data ● Setting Date Range ● High Frenquency ● Splits & Dividends ● Exporting ● Multiple Stocks ● Importing Indexes ● Currency Exchange ● Cryptocurrencies ● Mutual & Exchange Traded Funds ● Treasury Yields ● Ticker Object & Docs ● Stock Fundamentals ● Importing Financials ● Importing Put & Call ● Streaming Real-Time ●

▻ Importing Financial Indexes

```
import_all()
```

Historical data import: tickers are the **Dow Jones Industrial Average Index** with thirty different US company stocks and the **S&P500**, which aggregates 500 large US companies.

```
index_tickers = ["^DJI", "^GSPC"]
```

A look at the last five years worth of data for the two indexes

```
head_tail_vert(yf.download(index_tickers, period = "5y"), 5,
           "DJI & GSPC last 5 years")
```

```
[**********************100%**********************]  2 of 2 completed
```

DJI & GSPC last 5 years: head(5)

	Adj Close		Close		High		Low		Open	
	^DJI	^GSPC	^DJI	^GSPC	^DJI	^GSPC	^DJI	^GSPC	^DJI	^GSPC
Date										
2018-01-18	26,017.81	2,798.03	26,017.81	2,798.03	26,153.42	2,805.83	25,947.32	2,792.56	26,149.55	2,802.40	4927
2018-01-19	26,071.72	2,810.30	26,071.72	2,810.30	26,071.72	2,810.33	25,942.83	2,798.08	25,987.35	2,802.60	5850
2018-01-22	26,214.60	2,832.97	26,214.60	2,832.97	26,215.23	2,833.03	25,974.65	2,808.12	26,025.32	2,809.16	4312
2018-01-23	26,210.81	2,839.13	26,210.81	2,839.13	26,246.19	2,842.24	26,143.90	2,830.59	26,214.87	2,835.05	4334
2018-01-24	26,252.12	2,837.54	26,252.12	2,837.54	26,392.80	2,852.97	26,106.94	2,824.81	26,282.07	2,845.42	5187

DJI & GSPC last 5 years: tail(5)

	Adj Close		Close		High		Low		Open	
	^DJI	^GSPC	^DJI	^GSPC	^DJI	^GSPC	^DJI	^GSPC	^DJI	^GSPC
Date										
2023-01-10	33,704.10	3,919.25	33,704.10	3,919.25	33,726.54	3,919.83	33,421.80	3,877.29	33,516.43	3,888.57	2682
2023-01-11	33,973.01	3,969.61	33,973.01	3,969.61	33,974.69	3,970.07	33,711.04	3,928.54	33,754.03	3,932.35	3016
2023-01-12	34,189.97	3,983.17	34,189.97	3,983.17	34,292.67	3,997.76	33,792.10	3,937.56	34,047.86	3,977.57	3081
2023-01-13	34,302.61	3,999.09	34,302.61	3,999.09	34,342.32	4,003.95	33,915.49	3,947.67	34,075.31	3,960.60	2741
2023-01-17	33,910.85	3,990.97	33,910.85	3,990.97	34,269.97	4,015.39	33,860.67	3,984.57	34,222.32	3,999.28	3530

⊛ **`indexes`** will hold the closing data for the last 5 years for both indexes

```
indexes = yf.download(index_tickers, period="5y").Close
```

```
[**********************100%**********************]  2 of 2 completed
```

```
head_tail_horz(indexes, 5, 'DJI & GSPC Closing Data')
```

DJI & GSPC Closing Data

head(5)

Date	^DJI	^GSPC
2018-01-18	26,017.81	2,798.03
2018-01-19	26,071.72	2,810.30
2018-01-22	26,214.60	2,832.97
2018-01-23	26,210.81	2,839.13
2018-01-24	26,252.12	2,837.54

tail(5)

Date	^DJI	^GSPC
2023-01-10	33,704.10	3,919.25
2023-01-11	33,973.01	3,969.61
2023-01-12	34,189.97	3,983.17
2023-01-13	34,302.61	3,999.09
2023-01-17	33,910.85	3,990.97

⊛ **Normalizing** the indexes dataframe and saving as `norm_indexes` .

```
norm_indexes = indexes.div(indexes.iloc[0]).mul(100)

head_tail_horz(norm_indexes, 5, 'DJI & GSPC Closing Data Normalized')
```

DJI & GSPC Closing Data Normalized

head(5)

Date	^DJI	^GSPC
2018-01-18	100.00	100.00
2018-01-19	100.21	100.44
2018-01-22	100.76	101.25
2018-01-23	100.74	101.47
2018-01-24	100.90	101.41

tail(5)

Date	^DJI	^GSPC
2023-01-10	129.54	140.07
2023-01-11	130.58	141.87
2023-01-12	131.41	142.36
2023-01-13	131.84	142.93
2023-01-17	130.34	142.63

For indexes, it is always important to include whether the prices reflect only price return or the total return, including dividends of the constituents. Here, **our data only reflects prices changes with no modifications for dividends**.

```
plot_by_df(norm_indexes, sort_param="Date", cmap='cool',
           title="Normalized: DJI vs GSPC (price return only)",
           xlabel = "years", ylabel="normalized prices")
```

Normalized: DJI vs GSPC (price return only) (12 samples & overall plot)

```
GridspecLayout(children=(Output(layout=Layout(border='1px solid black',
grid_area='widget001')), Output(layout…
```

⊛ **Total Return Data**: getting tickers for the two indexes that include total return

```
index_tickers_tr = ['^DJITR', '^SP500TR']
indexes_tr = yf.download(index_tickers_tr, period = '5y').Close
```

```
[**********100%**********]  2 of 2 completed
```

It looks like Yahoo Finance is not the place to get the data on DJI total returns.

```
head_tail_horz(indexes_tr, 5, 'Indexes with Total Return Data')
```

Indexes with Total Return Data

head(5)

Date	^DJITR	^SP500TR
2018-01-18	nan	5,459.50
2018-01-19	nan	5,483.57
2018-01-22	nan	5,527.89
2018-01-23	nan	5,540.13
2018-01-24	nan	5,537.03

tail(5)

Date	^DJITR	^SP500TR
2023-01-10	nan	8,351.68
2023-01-11	nan	8,459.18
2023-01-12	nan	8,489.08
2023-01-13	nan	8,523.32
2023-01-17	81,606.59	8,506.03

Sections: ● Top ● Download & Import ● Historical Data ● Setting Date Range ● High Frenquency ● Splits & Dividends ● Exporting ● Multiple Stocks ● Importing Indexes ● Currency Exchange ● Cryptocurrencies ● Mutual & Exchange Traded Funds ● Treasury Yields ● Ticker Object & Docs ● Stock Fundamentals ● Importing Financials ● Importing Put & Call ● Streaming Real-Time ●

▻ Importing Currency Exchange Rates

```
import_all()
```

⊛ Currencies are **Euros** and **US Dollars**:

- Two tickers: Euro -> USD and USD -> Euro
- `currency=x` is the Yahoo Finance method of exchange rate tickers

```
euro_ticker = 'EURUSD=x'
dollar_ticker = 'USDEUR=x'
```

Head and **Tail** for each currency

```
euro_dollar = yf.download(euro_ticker, period = "5y")
dollar_euro = yf.download(dollar_ticker, period = "5y")
```

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

```
head_tail_horz(euro_dollar, 5, 'Euro to Dollar Exchange Data')
head_tail_horz(dollar_euro, 5, 'Dollar to Euro Exchange Data')
```

Euro to Dollar Exchange Data

head(5)

Date	Open	High	Low	Close	Adj Close	Volume
2018-01-17	1.23	1.23	1.22	1.23	1.23	0
2018-01-18	1.22	1.23	1.22	1.22	1.22	0
2018-01-19	1.22	1.23	1.22	1.22	1.22	0
2018-01-22	1.23	1.23	1.22	1.23	1.23	0
2018-01-23	1.23	1.23	1.22	1.23	1.23	0

tail(5)

Date	Open	High	Low	Close	Adj Close	Volume
2023-01-11	1.07	1.08	1.07	1.07	1.07	0
2023-01-12	1.08	1.08	1.07	1.08	1.08	0
2023-01-13	1.09	1.09	1.08	1.09	1.09	0
2023-01-16	1.08	1.09	1.08	1.08	1.08	0
2023-01-17	1.08	1.09	1.08	1.08	1.08	0

Dollar to Euro Exchange Data

head(5)

Date	Open	High	Low	Close	Adj Close	Volume
2018-01-17	0.81	0.82	0.81	0.81	0.81	0
2018-01-18	0.82	0.82	0.82	0.82	0.82	0
2018-01-19	0.82	0.82	0.81	0.82	0.82	0
2018-01-22	0.82	0.82	0.82	0.82	0.82	0
2018-01-23	0.82	0.82	0.81	0.82	0.82	0

tail(5)

Date	Open	High	Low	Close	Adj Close	Volume
2023-01-11	0.93	0.93	0.93	0.93	0.93	0
2023-01-12	0.93	0.93	0.92	0.93	0.93	0
2023-01-13	0.92	0.93	0.92	0.92	0.92	0
2023-01-16	0.92	0.93	0.92	0.92	0.92	0
2023-01-17	0.92	0.93	0.92	0.93	0.93	0

⊛ US Dollar to GB Pound

```
usd_gbp_ticker = "USDGBP=x"
usd_gbp = yf.download(usd_gbp_ticker, period = '5y')
```

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

```
head_tail_horz(usd_gbp, 5, 'US Dollars to GB Pounds')
```

US Dollars to GB Pounds

head(5)

Date	Open	High	Low	Close	Adj Close	Volume
2018-01-17	0.72	0.73	0.72	0.72	0.72	0
2018-01-18	0.72	0.72	0.72	0.72	0.72	0
2018-01-19	0.72	0.72	0.72	0.72	0.72	0
2018-01-22	0.72	0.72	0.72	0.72	0.72	0
2018-01-23	0.72	0.72	0.71	0.72	0.72	0

tail(5)

Date	Open	High	Low	Close	Adj Close	Volume
2023-01-11	0.82	0.83	0.82	0.82	0.82	0
2023-01-12	0.82	0.83	0.82	0.82	0.82	0
2023-01-13	0.82	0.82	0.82	0.82	0.82	0
2023-01-16	0.82	0.82	0.81	0.82	0.82	0
2023-01-17	0.82	0.82	0.81	0.81	0.81	0

Sections: ● Top ● Download & Import ● Historical Data ● Setting Date Range ● High Frenquency ● Splits & Dividends ● Exporting ● Multiple Stocks ● Importing Indexes ● Currency Exchange ● Cryptocurrencies ● Mutual

▻ Importing Cryptocurrencies

```
import_all()
```

⊛ Working with **Bitcoin** and **Ethereum**

```
crypto_tickers = ['BTC-USD', 'ETH-USD']
head_tail_vert(yf.download(crypto_tickers, period = '5y'),
               5, 'Crypto Historical Data')
```

```
[**********************100%**********************]  2 of 2 completed
```

Crypto Historical Data: head(5)

	Adj Close		Close		High		Low		Open		
	BTC-USD	ETH-USD	BTC-USD	ETH-USD	BTC-USD	ETH-USD	BTC-USD	ETH-USD	BTC-USD	ETH-USD	I
Date											
2018-01-17	11,188.60	1,014.25	11,188.60	1,014.25	11,678.00	1,090.23	9,402.29	780.92	11,431.10	1,061.34	1883
2018-01-18	11,474.90	1,036.28	11,474.90	1,036.28	12,107.30	1,100.31	10,942.50	967.76	11,198.80	1,016.44	1502
2018-01-19	11,607.40	1,039.10	11,607.40	1,039.10	11,992.80	1,093.22	11,172.10	1,003.71	11,429.80	1,028.82	1074
2018-01-20	12,899.20	1,155.15	12,899.20	1,155.15	13,103.00	1,167.11	11,656.20	1,044.95	11,656.20	1,044.95	1180
2018-01-21	11,600.10	1,049.58	11,600.10	1,049.58	12,895.90	1,155.68	11,288.20	1,021.50	12,889.20	1,155.68	993

Crypto Historical Data: tail(5)

	Adj Close		Close		High		Low		Open		
	BTC-USD	ETH-USD	BTC-USD	ETH-USD	BTC-USD	ETH-USD	BTC-USD	ETH-USD	BTC-USD	ETH-USD	I
Date											
2023-01-13	19,909.57	1,451.61	19,909.57	1,451.61	19,964.32	1,461.67	18,753.16	1,404.02	18,868.91	1,417.95	2922
2023-01-14	20,976.30	1,550.71	20,976.30	1,550.71	21,075.14	1,563.74	19,907.83	1,450.99	19,910.54	1,451.43	3896
2023-01-15	20,880.80	1,552.48	20,880.80	1,552.48	20,993.75	1,556.95	20,606.99	1,520.89	20,977.48	1,550.73	1929
2023-01-16	21,169.63	1,576.83	21,169.63	1,576.83	21,360.88	1,594.04	20,715.75	1,529.57	20,882.22	1,552.52	2679
2023-01-17	21,308.52	1,579.51	21,308.52	1,579.51	21,379.99	1,592.95	20,988.30	1,553.88	21,140.80	1,571.58	2448

⊛ **Closing data** for **Bitcoin** and **Ethereum**

```
crypto = yf.download(crypto_tickers, period = '5y').Close

head_tail_horz(crypto, 5, "Crypto Close Data")
```

```
[**********************100%**********************]  2 of 2 completed
```

Crypto Close Data

head(5)

Date	BTC-USD	ETH-USD
2018-01-17	11,188.60	1,014.25
2018-01-18	11,474.90	1,036.28
2018-01-19	11,607.40	1,039.10
2018-01-20	12,899.20	1,155.15
2018-01-21	11,600.10	1,049.58

tail(5)

Date	BTC-USD	ETH-USD
2023-01-13	19,909.57	1,451.61
2023-01-14	20,976.30	1,550.71
2023-01-15	20,880.80	1,552.48
2023-01-16	21,169.63	1,576.83
2023-01-17	21,308.52	1,579.51

Converting to other currencies

```
head_tail_vert(yf.download('BTC-EUR', period="5y"), 5, "Bitcoin vs Euro")
```

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

Bitcoin vs Euro: head(5)

Date	Open	High	Low	Close	Adj Close	Volume
2018-01-17	9,316.30	9,523.73	7,688.52	9,186.80	9,186.80	15461532303
2018-01-18	9,195.17	9,894.82	8,958.98	9,380.28	9,380.28	12278590383
2018-01-19	9,343.42	9,786.84	9,112.64	9,498.69	9,498.69	8789198141
2018-01-20	9,538.62	10,722.59	9,538.62	10,555.81	10,555.81	9657692596
2018-01-21	10,547.63	10,553.11	9,199.84	9,464.83	9,464.83	8106380366

Bitcoin vs Euro: tail(5)

Date	Open	High	Low	Close	Adj Close	Volume
2023-01-13	17,374.85	18,406.23	17,274.23	18,355.75	18,355.75	26944191476
2023-01-14	18,356.64	19,435.73	18,354.14	19,344.57	19,344.57	35936519640

Date	Open	High	Low	Close	Adj Close	Volume
2023-01-15	19,345.67	19,360.67	19,003.99	19,293.44	19,293.44	17831342602
2023-01-16	19,294.76	19,744.13	19,136.07	19,544.50	19,544.50	24735714659
2023-01-17	19,517.88	19,740.17	19,390.42	19,740.17	19,740.17	22685894656

```
head_tail_vert(yf.download('ETH-GBP', period="5y"), 5, "Ethereum vs GBP")
```

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

Ethereum vs GBP: head(5)

Date	Open	High	Low	Close	Adj Close	Volume
2018-01-17	769.20	789.62	564.86	733.69	733.69	6181394815
2018-01-18	735.27	795.83	700.21	746.01	746.01	4274940517
2018-01-19	740.64	788.36	721.68	749.87	749.87	2947571659
2018-01-20	754.10	842.25	754.10	833.62	833.62	2868723400
2018-01-21	834.00	834.00	735.21	755.64	755.64	2432030226

Ethereum vs GBP: tail(5)

Date	Open	High	Low	Close	Adj Close	Volume
2023-01-13	1,160.63	1,195.06	1,150.27	1,186.83	1,186.83	6282521158
2023-01-14	1,186.68	1,278.56	1,186.32	1,267.85	1,267.85	12627449006
2023-01-15	1,267.87	1,272.96	1,243.48	1,270.09	1,270.09	5542332445
2023-01-16	1,270.12	1,306.80	1,253.89	1,291.61	1,291.61	6925195834
2023-01-17	1,287.30	1,298.92	1,274.27	1,285.28	1,285.28	6132595712

Bitcoin Versus Etherum

```
plot_by_df(crypto, sort_param="Date", cmap="spring",
           logy=True, title="Bitcoin vs Ethereum (LogY Values)",
          xlabel = "years", ylabel = 'log(price)')
```

Bitcoin vs Ethereum (LogY Values) (12 samples & overall plot)

```
GridspecLayout(children=(Output(layout=Layout(border='1px solid black',
grid_area='widget001')), Output(layout…
```

Sections: ● Top ● Download & Import ● Historical Data ● Setting Date Range ● High Frenquency ● Splits & Dividends ● Exporting ● Multiple Stocks ● Importing Indexes ● Currency Exchange ● Cryptocurrencies ● Mutual & Exchange Traded Funds ● Treasury Yields ● Ticker Object & Docs ● Stock Fundamentals ● Importing Financials ● Importing Put & Call ● Streaming Real-Time ●

▻ Importing Mutual Funds and Exchange Traded Funds

```
import_all()
```

⊛ **ETFs** are how you can invest with a firm that tries to track the index.

- The first ticker below: **iShares 20+ year treasury bond ETF**
  - Tracks the US Treasury 20+ year bond index
  - 95% of assets in US Government bonds
  - Remaining maturity greater than or equal to 20 years
- The second ticker below: **Vivaldi multi-strategy fund** class 1

The existence of **Close** and **Adjusted Close** prices that are not equivalent indicates that these funds' data involve **dividends**.

```
treas_etf_ticker = "TLT"
vivaldi_ticker = "OMOIX"
```

⊛ **iShares** treasury bond ETF

```
ishares = yf.download(treas_etf_ticker, period = '5y')
```

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

```
head_tail_horz(ishares, 5, 'US 20+ Years Treasury Bonds')
```

US 20+ Years Treasury Bonds

head(5)

Date	Open	High	Low	Close	Adj Close	Volume
2018-01-18	123.93	124.27	123.54	123.71	111.56	10,843,900
2018-01-19	123.40	123.56	122.98	123.06	110.97	11,374,300
2018-01-22	123.45	123.71	122.99	123.18	111.08	8,984,700
2018-01-23	123.81	124.20	123.24	123.71	111.56	12,325,700
2018-01-24	122.85	123.13	122.54	123.04	110.95	9,038,900

tail(5)

Date	Open	High	Low	Close	Adj Close	Volume
2023-01-10	104.53	104.82	103.50	103.99	103.99	20,545,200
2023-01-11	104.97	105.72	104.70	105.68	105.68	18,794,000
2023-01-12	105.82	107.81	104.56	107.76	107.76	32,029,600
2023-01-13	107.10	107.80	106.38	106.75	106.75	17,744,800
2023-01-17	105.67	106.66	105.65	106.06	106.06	13,748,235

⊛ **Vivaldi** multi-strategy fund

```
vivaldi = yf.download(vivaldi_ticker, period = '5y')
```

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

```
head_tail_horz(vivaldi, 5, 'Vivaldi Multi-Strategy Fund')
```

Vivaldi Multi-Strategy Fund

head(5)

Date	Open	High	Low	Close	Adj Close	Volume
2018-01-18	26.75	26.75	26.75	26.75	21.05	0
2018-01-19	26.76	26.76	26.76	26.76	21.05	0
2018-01-22	26.79	26.79	26.79	26.79	21.08	0
2018-01-23	26.76	26.76	26.76	26.76	21.05	0
2018-01-24	26.75	26.75	26.75	26.75	21.05	0

tail(5)

Date	Open	High	Low	Close	Adj Close	Volume
2023-01-09	23.78	23.78	23.78	23.78	23.78	0
2023-01-10	23.85	23.85	23.85	23.85	23.85	0
2023-01-11	23.89	23.89	23.89	23.89	23.89	0
2023-01-12	23.94	23.94	23.94	23.94	23.94	0
2023-01-13	23.96	23.96	23.96	23.96	23.96	0

Sections: ● Top ● Download & Import ● Historical Data ● Setting Date Range ● High Frenquency ● Splits & Dividends ● Exporting ● Multiple Stocks ● Importing Indexes ● Currency Exchange ● Cryptocurrencies ● Mutual

▻ Importing Treasury Yields

```
import_all()
```

Importing historical US Treasury yield data.

- The US Government borrows money by issuing Treasury or Government bonds with maturities of 5 years, 10 years, and longer
- The **Treasury Yield** indicates how much an investor can earn with the particular bond
- This form of investment is considered to be credit risk-free
- The is based on the likelihood of the US Government repaying its obligations
- They are often used in finance as an approximation of the risk-free rate
- The **risk-free rate** plays an important role in portfolio management and asset pricing

⊛ The following two tickers are for the **10 year Treasury Yield** and the **5 year Treasury Yield**

```
ten_year_treasury = '^TNX'    # Ten year Treasury Yield
five_year_treasury = '^FVX'    # Five year Treasury Yield
treasury_tickers = [ten_year_treasury, five_year_treasury]
```

```
treasury_history = yf.download(treasury_tickers, period = '5y')

head_tail_vert(treasury_history, 5, 'US Treasury 5-Year & 10-Year Yields: Historical Da
```

```
[**********************100%**********************]  2 of 2 completed
```

US Treasury 5-Year & 10-Year Yields: Historical Data: head(5)

	Adj Close		Close		High		Low		Open		Volume	
	^FVX	^TNX	^FVX	^TNX	^FVX	^TNX	^FVX	^TNX	^FVX	^TNX	^FVX	^TNX
Date												
2018-01-17	2.39	2.58	2.39	2.58	2.40	2.58	2.37	2.54	2.38	2.56	0	0
2018-01-18	2.41	2.61	2.41	2.61	2.42	2.62	2.40	2.60	2.42	2.61	0	0
2018-01-19	2.44	2.64	2.44	2.64	2.44	2.65	2.41	2.61	2.42	2.63	0	0
2018-01-22	2.46	2.66	2.46	2.66	2.46	2.66	2.43	2.63	2.45	2.66	0	0
2018-01-23	2.42	2.62	2.42	2.62	2.43	2.64	2.41	2.61	2.43	2.63	0	0

US Treasury 5-Year & 10-Year Yields: Historical Data: tail(5)

	Adj Close		Close		High		Low		Open		Volume	

	^FVX Adj Close	^TNX Adj Close	^FVX Close	^TNX Close	^FVX High	^TNX High	^FVX Low	^TNX Low	^FVX Open	^TNX Open	^FVX Volume	^TNX Volume
Date	^FVX	^TNX	^FVX	^TNX	^FVX	^TNX	^FVX	^TNX	^FVX	^TNX	^FVX	^TNX
Date												
2023-01-10	3.73	3.62	3.73	3.62	3.76	3.64	3.68	3.57	3.72	3.58	0	0
2023-01-11	3.67	3.55	3.67	3.55	3.71	3.60	3.66	3.55	3.69	3.57	0	0
2023-01-12	3.55	3.45	3.55	3.45	3.73	3.59	3.52	3.43	3.63	3.50	0	0
2023-01-13	3.61	3.51	3.61	3.51	3.62	3.52	3.53	3.43	3.57	3.49	0	0
2023-01-17	3.61	3.53	3.61	3.53	3.67	3.58	3.58	3.50	3.66	3.58	0	0

Dataframe for Treasury Yields - closing data only

```
treasury_yields = yf.download(treasury_tickers, period = '5y').Close
```

```
[**********************100%**********************]  2 of 2 completed
```

⊛ **Understanding Treasury Yields**: If on the first day below, 2018-01-16, an investor invested in the five or ten year bonds, they could expect the listed average percent increase per year.

```
head_tail_horz(treasury_yields, 5, "US Treasury Yields: Last 5 Years")
```

US Treasury Yields: Last 5 Years

head(5)

Date	^FVX	^TNX
2018-01-17	2.39	2.58
2018-01-18	2.41	2.61
2018-01-19	2.44	2.64
2018-01-22	2.46	2.66
2018-01-23	2.42	2.62

tail(5)

Date	^FVX	^TNX
2023-01-10	3.73	3.62
2023-01-11	3.67	3.55
2023-01-12	3.55	3.45
2023-01-13	3.61	3.51
2023-01-17	3.61	3.53

```
plot_by_df(treasury_yields, sort_param="Date", num_records = 10,
           title = "5 and 10 Year Treasury Bond Yields",
          xlabel = "years", ylabel = "yearly percentage profit", legend_loc = 3,
           cmap = 'spring')
```

5 and 10 Year Treasury Bond Yields (10 samples & overall plot)

```
GridspecLayout(children=(Output(layout=Layout(border='1px solid black',
grid_area='widget001')), Output(layout…
```

▻ The Ticker Object

```
import_all()
```

Aside from the method of using `yfinance.download()`, it is also possible to get historical stock data using a **ticker object** using `yf.Ticker()`. With this method, it is possible to also retrieve:

- fundamental stock data
- performance metrics
- balance sheets
- profit and loss statements
- cashflow statements

Using **Disney** stock for this section:

```
ticker = "DIS"
disney = yf.Ticker(ticker)
disney
```

```
yfinance.Ticker object <DIS>
```

⊛ `yf.Ticker.history()` does not contain the adjusted close data, because the **here, close prices are adjusted for dividends and stock splits by default**.
`yf.download()` is the better option for getting simple historical price data.

```
head_tail_horz(disney.history(), 5, "Historical Disney Prices")
```

Historical Disney Prices

head(5)

Date	Open	High	Low	Close	Volume	Dividends	Stock Splits
2022-12-19	89.44	89.46	85.41	85.78	19,225,300	0.00	0.00
2022-12-20	86.08	87.84	85.76	87.02	14,918,400	0.00	0.00
2022-12-21	87.19	88.10	86.48	86.92	11,004,800	0.00	0.00
2022-12-22	86.03	86.73	84.69	86.67	15,487,400	0.00	0.00
2022-12-23	86.06	88.07	85.77	88.01	11,171,600	0.00	0.00

tail(5)

Date	Open	High	Low	Close	Volume	Dividends	Stock Splits
2023-01-10	94.28	95.64	93.82	95.56	6,914,600	0.00	0.00
2023-01-11	95.92	96.55	95.05	96.33	8,757,300	0.00	0.00
2023-01-12	98.50	100.63	97.57	99.81	19,764,300	0.00	0.00
2023-01-13	99.38	99.60	98.01	99.40	12,220,500	0.00	0.00
2023-01-17	100.32	100.99	99.00	99.91	12,181,349	0.00	0.00

More `yf.Ticker` functions along with others from documentation:

```
msft = yf.Ticker("MSFT")
```

.history()

```
# get historical market data
hist = msft.history(period="max")
head_tail_vert(hist, 5, 'microsoft history')
```

microsoft history: head(5)

Date	Open	High	Low	Close	Volume	Dividends	Stock Splits
1986-03-13	0.06	0.06	0.06	0.06	1031788800	0.00	0.00
1986-03-14	0.06	0.06	0.06	0.06	308160000	0.00	0.00
1986-03-17	0.06	0.06	0.06	0.06	133171200	0.00	0.00
1986-03-18	0.06	0.06	0.06	0.06	67766400	0.00	0.00
1986-03-19	0.06	0.06	0.06	0.06	47894400	0.00	0.00

microsoft history: tail(5)

Date	Open	High	Low	Close	Volume	Dividends	Stock Splits
2023-01-10	227.76	231.31	227.33	228.85	27033900	0.00	0.00
2023-01-11	231.29	235.95	231.11	235.77	28669300	0.00	0.00
2023-01-12	235.26	239.90	233.56	238.51	27269500	0.00	0.00
2023-01-13	237.00	239.37	234.92	239.23	21317700	0.00	0.00
2023-01-17	237.97	240.91	237.09	240.35	29736926	0.00	0.00

.history_metadata()

```
# show meta information about the history (requires history() to be called first)
msft.history_metadata
```


```
{'currency': 'USD',
 'symbol': 'MSFT',
 'exchangeName': 'NMS',
 'instrumentType': 'EQUITY',
 'firstTradeDate': 511108200,
 'regularMarketTime': 1673989204,
 'gmtoffset': -18000,
 'timezone': 'EST',
 'exchangeTimezoneName': 'America/New_York',
 'regularMarketPrice': 240.35,
 'chartPreviousClose': 0.097,
 'priceHint': 2,
 'currentTradingPeriod': {'pre': {'timezone': 'EST',
   'start': 1673946000,
   'end': 1673965800,
   'gmtoffset': -18000},
  'regular': {'timezone': 'EST',
   'start': 1673965800,
   'end': 1673989200,
   'gmtoffset': -18000},
  'post': {'timezone': 'EST',
   'start': 1673989200,
   'end': 1674003600,
   'gmtoffset': -18000}},
 'dataGranularity': '1d',
 'range': '',
 'validRanges': ['1d',
  '5d',
  '1mo',
  '3mo',
  '6mo',
  '1y',
  '2y',
  '5y',
  '10y',
  'ytd',
  'max']}
```


.actions

```
# show actions (dividends, splits, capital gains)
see(msft.actions.sample(10).sort_values("Date"), 'msft.actions.sample(10).sort_values("
```

msft.actions.sample(10).sort_values("Date")

Date	Dividends	Stock Splits
1998-02-23	0.00	2.00
2003-02-18	0.00	2.00
2006-02-15	0.09	0.00
2013-08-13	0.23	0.00
2013-11-19	0.28	0.00
2015-02-17	0.31	0.00
2016-11-15	0.39	0.00
2018-08-15	0.42	0.00
2019-05-15	0.46	0.00
2021-08-18	0.56	0.00

.analyst_price_target

```
see(msft.analyst_price_target, 'msft.analyst_price_target')
```

msft.analyst_price_target

	0
targetLowPrice	234.00
currentPrice	240.35
targetMeanPrice	293.77
targetHighPrice	411.00
numberOfAnalystOpinions	47.00

.balance_sheet

```
see(msft.balance_sheet.head(10), 'msft.balance_sheet.head(10)')
```

msft.balance_sheet.head(10)

	2022-06-30 00:00:00	2021-06-30 00:00:00	2020-06-30 00:00:00	2019-06-30 00:00:00
Total Assets	364,840,000,000.00	333,779,000,000.00	301,311,000,000.00	286,556,000,000.00
Current Assets	169,684,000,000.00	184,406,000,000.00	181,915,000,000.00	175,552,000,000.00
Cash Cash Equivalents And Short Term Investments	104,749,000,000.00	130,334,000,000.00	136,527,000,000.00	133,819,000,000.00
Cash And Cash Equivalents	13,931,000,000.00	14,224,000,000.00	13,576,000,000.00	11,356,000,000.00
Cash Financial	8,258,000,000.00	7,272,000,000.00	NaN	NaN
Cash Equivalents	5,673,000,000.00	6,952,000,000.00	NaN	NaN
Other Short Term Investments	90,818,000,000.00	116,110,000,000.00	122,951,000,000.00	122,463,000,000.00

	2022-06-30 00:00:00	2021-06-30 00:00:00	2020-06-30 00:00:00	2019-06-30 00:00:00
Receivables	44,261,000,000.00	38,043,000,000.00	32,011,000,000.00	29,524,000,000.00
Accounts Receivable	44,261,000,000.00	38,043,000,000.00	32,011,000,000.00	29,524,000,000.00
Gross Accounts Receivable	44,894,000,000.00	38,794,000,000.00	32,799,000,000.00	29,935,000,000.00

.calendar

```
see(msft.calendar, 'msft.calendar')
```

msft.calendar

	Value
Earnings Date	2023-01-24 21:00:00
Earnings Average	2.31
Earnings Low	2.23
Earnings High	2.91
Revenue Average	52986600000
Revenue Low	52389000000
Revenue High	53871800000

.cash_flow

```
see(msft.cash_flow.head(5),'msft.cash_flow.head(5)')
```

msft.cash_flow.head(5)

	2022-06-30 00:00:00	2021-06-30 00:00:00	2020-06-30 00:00:00	2019-06-30 00:00:00
Operating Cash Flow	89,035,000,000.00	76,740,000,000.00	60,675,000,000.00	52,185,000,000.00
Cash Flow From Continuing Operating Activities	89,035,000,000.00	76,740,000,000.00	60,675,000,000.00	52,185,000,000.00
Net Income From Continuing Operations	72,738,000,000.00	61,271,000,000.00	44,281,000,000.00	39,240,000,000.00
Operating Gains Losses	-409,000,000.00	-1,249,000,000.00	-219,000,000.00	-792,000,000.00
Gain Loss On Investment Securities	NaN	NaN	-219,000,000.00	-792,000,000.00

.dividends

```
# show dividends
head_tail_vert(msft.dividends, 5, 'msft.dividends')
```

msft.dividends: head(5)

Date	Dividends
2003-02-19	0.08
2003-10-15	0.16
2004-08-23	0.08
2004-11-15	3.08
2005-02-15	0.08

msft.dividends: tail(5)

Date	Dividends
2021-11-17	0.62
2022-02-16	0.62
2022-05-18	0.62
2022-08-17	0.62
2022-11-16	0.68

.earnings

```
msft.earnings
```

Year	Revenue	Earnings
2019	125843000000	39240000000
2020	143015000000	44281000000
2021	168088000000	61271000000
2022	198270000000	72738000000

.earnings_dates

```
see(msft.earnings_dates, intraday=True, title="msft.earnings_dates")
```

msft.earnings_dates

Earnings Date	EPS Estimate	Reported EPS	Surprise(%)
2023-10-23 06:00:00-04:00	NaN	NaN	NaN
2023-07-24 06:00:00-04:00	NaN	NaN	NaN
2023-04-24 06:00:00-04:00	NaN	NaN	NaN

Earnings Date	EPS Estimate	Reported EPS	Surprise(%)
2023-01-24 17:00:00-05:00	2.31	NaN	NaN
2023-01-24 16:00:00-05:00	2.31	NaN	NaN
2022-10-25 12:00:00-04:00	2.30	2.35	0.02
2022-07-26 12:00:00-04:00	2.29	2.23	-0.03
2022-04-26 12:00:00-04:00	2.19	2.22	0.02
2022-01-25 11:00:00-05:00	2.31	2.48	0.07
2021-10-26 12:00:00-04:00	2.08	2.27	0.09
2021-07-27 12:00:00-04:00	1.92	2.17	0.13
2021-04-27 12:00:00-04:00	1.78	2.03	0.14

.earnings_forecasts

```
see(msft.earnings_forecasts, "msft.earnings_forecasts")
```

msft.earnings_forecasts

	avg	low	high	yearAgoEps	numberOfAnalysts	growth	period	endDate
0	2.31	2.23	2.91	2.48	35.00	-0.07	0q	2022-12-31
1	2.33	1.95	2.50	2.22	34.00	0.05	+1q	2023-03-31
2	9.60	9.26	11.21	9.21	46.00	0.04	0y	2023-06-30
3	11.16	10.21	12.48	9.60	45.00	0.16	+1y	2024-06-30
4	NaN	NaN	NaN	NaN	NaN	NaN	+5y	None
5	NaN	NaN	NaN	NaN	NaN	NaN	-5y	None

.earnings_trend

```
msft.earnings_trend
```

Period	Max Age	End Date	Growth	Earnings Estimate Avg	Earnings Estimate Low	Earnings Estimate High	Earnings Estimate Year Ago Eps	Earnings Estimate Number Of Analysts	Earnings Estimate Growth	Revenue Estimate Avg	...	Re Es G
0Q	1	2022-12-31	-0.07	2.31	2.23	2.91	2.48	35.00	-0.07	52,986,600,000.00	...	
+1Q	1	2023-03-31	0.05	2.33	1.95	2.50	2.22	34.00	0.05	52,516,800,000.00	...	
0Y	1	2023-06-30	0.04	9.60	9.26	11.21	9.21	46.00	0.04	212,964,000,000.00	...	
+1Y	1	2024-06-30	0.16	11.16	10.21	12.48	9.60	45.00	0.16	239,627,000,000.00	...	

Period	Max Age	End Date	Growth	Earnings Estimate Avg	Earnings Estimate Low	Earnings Estimate High	Earnings Estimate Year Ago Eps	Earnings Estimate Number Of Analysts	Earnings Estimate Growth	Revenue Estimate Avg	...	Re Es G
+5Y	1	NaT	0.13	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	
-5Y	1	NaT	0.24	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	

6 rows × 24 columns

.financials

```
see(msft.financials.head(5), 'msft.financials.head(5)')
```

msft.financials.head(5)

	2022-06-30 00:00:00	2021-06-30 00:00:00	2020-06-30 00:00:00	2019-06-30 00:00:00
Total Revenue	198,270,000,000.00	168,088,000,000.00	143,015,000,000.00	125,843,000,000.00
Operating Revenue	198,270,000,000.00	168,088,000,000.00	143,015,000,000.00	125,843,000,000.00
Cost Of Revenue	62,650,000,000.00	52,232,000,000.00	46,078,000,000.00	42,910,000,000.00
Gross Profit	135,620,000,000.00	115,856,000,000.00	96,937,000,000.00	82,933,000,000.00
Operating Expense	52,237,000,000.00	45,940,000,000.00	43,978,000,000.00	39,974,000,000.00

.income_stmt

```
see(msft.income_stmt.head(5), 'msft.income_stmt.head(5)')
```

msft.income_stmt.head(5)

	2022-06-30 00:00:00	2021-06-30 00:00:00	2020-06-30 00:00:00	2019-06-30 00:00:00
Total Revenue	198,270,000,000.00	168,088,000,000.00	143,015,000,000.00	125,843,000,000.00
Operating Revenue	198,270,000,000.00	168,088,000,000.00	143,015,000,000.00	125,843,000,000.00
Cost Of Revenue	62,650,000,000.00	52,232,000,000.00	46,078,000,000.00	42,910,000,000.00
Gross Profit	135,620,000,000.00	115,856,000,000.00	96,937,000,000.00	82,933,000,000.00
Operating Expense	52,237,000,000.00	45,940,000,000.00	43,978,000,000.00	39,974,000,000.00

.quarterly_balance_sheet

```
see(msft.quarterly_balance_sheet.head(5),"msft.quarterly_balance_sheet.head(5)")
```

msft.quarterly_balance_sheet.head(5)

	2022-09-30 00:00:00	2022-06-30 00:00:00	2022-03-31 00:00:00	2021-12-31 00:00:00
Total Assets	359,784,000,000.00	364,840,000,000.00	344,607,000,000.00	340,389,000,000.00

	2022-09-30 00:00:00	2022-06-30 00:00:00	2022-03-31 00:00:00	2021-12-31 00:00:00
Current Assets	160,812,000,000.00	169,684,000,000.00	153,922,000,000.00	174,188,000,000.00
Cash Cash Equivalents And Short Term Investments	107,244,000,000.00	104,749,000,000.00	104,660,000,000.00	125,348,000,000.00
Cash And Cash Equivalents	22,884,000,000.00	13,931,000,000.00	12,498,000,000.00	20,604,000,000.00
Cash Financial	7,237,000,000.00	8,258,000,000.00	7,456,000,000.00	6,255,000,000.00

.quarterly_cash_flow

```
see(msft.quarterly_cash_flow.head(5),"msft.quarterly_cash_flow.head(5)")
```

msft.quarterly_cash_flow.head(5)

	2022-09-30 00:00:00	2022-06-30 00:00:00	2022-03-31 00:00:00	2021-12-31 00:00:00
Operating Cash Flow	23,198,000,000.00	24,629,000,000.00	25,386,000,000.00	14,480,000,000.00
Cash Flow From Continuing Operating Activities	23,198,000,000.00	24,629,000,000.00	25,386,000,000.00	14,480,000,000.00
Net Income From Continuing Operations	17,556,000,000.00	16,740,000,000.00	16,728,000,000.00	18,765,000,000.00
Operating Gains Losses	-22,000,000.00	157,000,000.00	105,000,000.00	-307,000,000.00
Depreciation Amortization Depletion	2,790,000,000.00	3,979,000,000.00	3,773,000,000.00	3,496,000,000.00

.quarterly_earnings

```
see(msft.quarterly_earnings,"msft.quarterly_earnings")
```

msft.quarterly_earnings

Quarter	Revenue	Earnings
4Q2021	51728000000	18765000000
1Q2022	49360000000	16728000000
2Q2022	51865000000	16740000000
3Q2022	50122000000	17556000000

.quarterly_financials

```
see(msft.quarterly_financials.head(5),"msft.quarterly_financials.head(5)")
```

msft.quarterly_financials.head(5)

	2022-09-30 00:00:00	2022-06-30 00:00:00	2022-03-31 00:00:00	2021-12-31 00:00:00
Total Revenue	50,122,000,000.00	51,865,000,000.00	49,360,000,000.00	51,728,000,000.00

	2022-09-30 00:00:00	2022-06-30 00:00:00	2022-03-31 00:00:00	2021-12-31 00:00:00
Operating Revenue	50,122,000,000.00	51,865,000,000.00	49,360,000,000.00	51,728,000,000.00
Cost Of Revenue	15,452,000,000.00	16,429,000,000.00	15,615,000,000.00	16,960,000,000.00
Gross Profit	34,670,000,000.00	35,436,000,000.00	33,745,000,000.00	34,768,000,000.00
Operating Expense	13,152,000,000.00	14,902,000,000.00	13,381,000,000.00	12,521,000,000.00

.quarterly_income_stmt

```
see(msft.quarterly_income_stmt.head(5),"msft.quarterly_income_stmt.head(5)")
```

msft.quarterly_income_stmt.head(5)

	2022-09-30 00:00:00	2022-06-30 00:00:00	2022-03-31 00:00:00	2021-12-31 00:00:00
Total Revenue	50,122,000,000.00	51,865,000,000.00	49,360,000,000.00	51,728,000,000.00
Operating Revenue	50,122,000,000.00	51,865,000,000.00	49,360,000,000.00	51,728,000,000.00
Cost Of Revenue	15,452,000,000.00	16,429,000,000.00	15,615,000,000.00	16,960,000,000.00
Gross Profit	34,670,000,000.00	35,436,000,000.00	33,745,000,000.00	34,768,000,000.00
Operating Expense	13,152,000,000.00	14,902,000,000.00	13,381,000,000.00	12,521,000,000.00

.recommendations

```
see(msft.recommendations.head(5),"msft.recommendations.head(5)")
```

msft.recommendations.head(5)

Date	Firm	To Grade	From Grade	Action
2012-03-16	Argus Research	Buy		up
2012-03-19	Hilliard Lyons	Long-Term Buy		main
2012-03-22	Morgan Stanley	Overweight		main
2012-04-03	UBS	Buy		main
2012-04-20	Deutsche Bank	Buy		main

.recommendations_summary

```
see(msft.recommendations_summary.head(5),"msft.recommendations_summary.head(5)")
```

msft.recommendations_summary.head(5)

Date	Firm	To Grade	From Grade	Action
2012-03-16	Argus Research	Buy		up

Date	Firm	To Grade	From Grade	Action
2012-03-19	Hilliard Lyons	Long-Term Buy		main
2012-03-22	Morgan Stanley	Overweight		main
2012-04-03	UBS	Buy		main
2012-04-20	Deutsche Bank	Buy		main

.revenue_forecasts

```
see(msft.revenue_forecasts,"msft.revenue_forecasts")
```

msft.revenue_forecasts

	avg	low	high	numberOfAnalysts	yearAgoRevenue	growth	period
0	52,986,600,000.00	52,389,000,000.00	53,871,800,000.00	33.00	51,728,000,000.00	0.02	0q
1	52,516,800,000.00	49,186,000,000.00	54,999,400,000.00	32.00	49,360,000,000.00	0.06	+1q
2	212,964,000,000.00	205,383,000,000.00	241,257,000,000.00	47.00	198,270,000,000.00	0.07	0y
3	239,627,000,000.00	215,371,000,000.00	259,099,000,000.00	46.00	212,964,000,000.00	0.12	+1y
4	NaN	NaN	NaN	NaN	NaN	NaN	+5y
5	NaN	NaN	NaN	NaN	NaN	NaN	-5y

.shares

```
see(msft.shares,"msft.shares")
```

msft.shares

Year	BasicShares
2019	7673000000
2020	7610000000
2021	7547000000
2022	7496000000

.sustainability

```
see(msft.sustainability,"msft.sustainability")
```

msft.sustainability

	Value
2022-8	
palmOil	False
controversialWeapons	False
gambling	False
socialScore	8.39
nuclear	False
furLeather	False
alcoholic	False
gmo	False
catholic	False
socialPercentile	None
peerCount	105
governanceScore	5.33
environmentPercentile	None
animalTesting	False
tobacco	False
totalEsg	15.24
highestControversy	3
esgPerformance	UNDER_PERF
coal	False
pesticides	False
adult	False
percentile	10.73
peerGroup	Software & Services
smallArms	False
environmentScore	1.53
governancePercentile	None
militaryContract	False

.splits

```
# show splits
see(msft.splits,"msft.splits")
```

msft.splits

	Stock Splits
Date	
1987-09-21	2.00
1990-04-16	2.00

	Stock Splits
Date	
1991-06-27	1.50
1992-06-15	1.50
1994-05-23	2.00
1996-12-09	2.00
1998-02-23	2.00
1999-03-29	2.00
2003-02-18	2.00

.major_holders

```
# show major holders
see(msft.major_holders,"msft.major_holders")
```

msft.major_holders

	0	1
0	0.06%	% of Shares Held by All Insider
1	73.35%	% of Shares Held by Institutions
2	73.39%	% of Float Held by Institutions
3	5960	Number of Institutions Holding Shares

.institutional_holders

```
# show institutional holders
msft.institutional_holders['Date Reported'].dt.strftime('%Y-%m-%d')
```

```
0    2022-09-29
1    2022-09-29
2    2022-09-29
3    2022-09-29
4    2022-09-29
5    2022-09-29
6    2022-09-29
7    2022-09-29
8    2022-09-29
9    2022-09-29
Name: Date Reported, dtype: object
```

.mutualfundholders

```
# show mutualfund holders
see(msft.mutualfund_holders,"msft.mutualfund_holders")
```

msft.mutualfund_holders

	Holder	Shares	Date Reported	% Out	Value
0	Vanguard Total Stock Market Index Fund	222313847	2022-09-29	0.03	53433134483
1	Vanguard 500 Index Fund	169062762	2022-09-29	0.02	40634235878
2	SPDR S&P 500 ETF Trust	82389656	2022-11-29	0.01	19802354322
3	Fidelity 500 Index Fund	80870977	2022-11-29	0.01	19437339815
4	iShares Core S&P 500 ETF	66940817	2022-12-30	0.01	16089225774
5	Invesco ETF Tr-Invesco QQQ Tr, Series 1 ETF	65747097	2022-11-29	0.01	15802315165
6	Vanguard Growth Index Fund	61426549	2022-09-29	0.01	14763871427
7	Vanguard Institutional Index Fund-Institutional Index Fund	54149972	2022-09-29	0.01	13014946100
8	Growth Fund Of America Inc	49038474	2022-09-29	0.01	11786397525
9	Select Sector SPDR Fund-Technology	35233656	2022-11-29	0.00	8468409434

.options

```
# show options expirations
msft.options
```

```
('2023-01-20',
 '2023-01-27',
 '2023-02-03',
 '2023-02-10',
 '2023-02-17',
 '2023-02-24',
 '2023-03-03',
 '2023-03-17',
 '2023-04-21',
 '2023-06-16',
 '2023-07-21',
 '2023-08-18',
 '2023-09-15',
 '2024-01-19',
 '2024-06-21',
 '2025-01-17',
 '2025-06-20')
```

.news

```
# show news
msft.news[0]
```

```
{'uuid': 'b914a73e-e1f9-388a-a649-ba4f8b2574c0',
 'title': 'Market Rally Pauses, Tesla Keeps Rebounding; United Airlines, Moderna, Micros
 'publisher': "Investor's Business Daily",
 'link': 'https://finance.yahoo.com/m/b914a73e-e1f9-388a-a649-ba4f8b2574c0/market-rally-
```

```
 'providerPublishTime': 1673994096,
 'type': 'STORY',
 'thumbnail': {'resolutions': [{'url': 'https://s.yimg.com/uu/api/res/1.2/meY_pPOZbC99Si
~B/aD01NjM7dz0xMDAwO2FwcGlkPXl0YWNoeW9u/https://media.zenfs.com/en/ibd.com/9080cd15e60ec
    'width': 1000,
    'height': 563,
    'tag': 'original'},
   {'url': 'https://s.yimg.com/uu/api/res/1.2/IFQnoqX1cTefxkf8VyAxTA--
~B/Zmk9ZmlsbDtoPTE0MDtweW9mZj0wO3c9MTQwO2FwcGlkPXl0YWNoeW9u/https://media.zenfs.com/en/i
    'width': 140,
    'height': 140,
    'tag': '140x140'}]},
 'relatedTickers': ['UAL',
  '^GSPC',
  'TSLA',
  'MSFT',
  '^DJI',
  'COMP',
  '^RUT',
  'VRTX',
  'MRNA']}
```

.option_chain()

```
# get option chain for specific expiration
opt = msft.option_chain('2025-06-20')
type(opt)
```

```
yfinance.ticker.Options
```

Cached Session

```
import requests_cache
session = requests_cache.CachedSession('yfinance.cache')
session.headers['User-agent'] = 'my-program/1.0'
ticker = yf.Ticker('msft', session=session)
# The scraped response will be stored in the cache
ticker.actions
```

Date	Dividends	Stock Splits
1987-09-21 00:00:00-04:00	0.00	2.00
1990-04-16 00:00:00-04:00	0.00	2.00
1991-06-27 00:00:00-04:00	0.00	1.50
1992-06-15 00:00:00-04:00	0.00	1.50
1994-05-23 00:00:00-04:00	0.00	2.00
...	...	...
2021-11-17 00:00:00-05:00	0.62	0.00

Date	Dividends	Stock Splits
2022-02-16 00:00:00-05:00	0.62	0.00
2022-05-18 00:00:00-04:00	0.62	0.00
2022-08-17 00:00:00-04:00	0.62	0.00
2022-11-16 00:00:00-05:00	0.68	0.00

85 rows × 2 columns

yf.download() parameters

```
data = yf.download(  # or pdr.get_data_yahoo(...
        # tickers list or string as well
        tickers = "SPY AAPL MSFT",

        # use "period" instead of start/end
        # valid periods: 1d,5d,1mo,3mo,6mo,1y,2y,5y,10y,ytd,max
        # (optional, default is '1mo')
        period = "ytd",

        # fetch data by interval (including intraday if period < 60 days)
        # valid intervals: 1m,2m,5m,15m,30m,60m,90m,1h,1d,5d,1wk,1mo,3mo
        # (optional, default is '1d')
        interval = "5d",

        # Whether to ignore timezone when aligning ticker data from
        # different timezones. Default is False.
        ignore_tz = False,

        # group by ticker (to access via data['SPY'])
        # (optional, default is 'column')
        group_by = 'ticker',

        # adjust all OHLC automatically
        # (optional, default is False)
        auto_adjust = True,

        # attempt repair of missing data or currency mixups e.g. $/cents
        repair = False,

        # download pre/post regular market hours data
        # (optional, default is False)
        prepost = True,

        # use threads for mass downloading? (True/False/Integer)
        # (optional, default is True)
        threads = True,

        # proxy URL scheme use use when downloading?
        # (optional, default is None)
```

```
        proxy = None
    )
```

```
[**********************100%**********************]  3 of 3 completed
```

```
head_tail_vert(data, 5, 'Example Data')
```

Example Data: head(5)

	SPY					AAPL							
	Open	High	Low	Close	Volume	Open	High	Low	Close	Volume	Open	High	
Date													
2023-01-03	384.37	386.43	377.83	380.82	74850700	130.28	130.90	124.17	125.07	112117500	243.08	245.75	2
2023-01-13	393.62	400.23	393.34	397.77	125557011	132.03	137.29	131.66	135.94	121211444	237.00	240.91	2

Example Data: tail(5)

	SPY					AAPL							
	Open	High	Low	Close	Volume	Open	High	Low	Close	Volume	Open	High	
Date													
2023-01-03	384.37	386.43	377.83	380.82	74850700	130.28	130.90	124.17	125.07	112117500	243.08	245.75	2
2023-01-13	393.62	400.23	393.34	397.77	125557011	132.03	137.29	131.66	135.94	121211444	237.00	240.91	2

Sections: ● Top ● Download & Import ● Historical Data ● Setting Date Range ● High Frenquency ● Splits & Dividends ● Exporting ● Multiple Stocks ● Importing Indexes ● Currency Exchange ● Cryptocurrencies ● Mutual & Exchange Traded Funds ● Treasury Yields ● Ticker Object & Docs ● Stock Fundamentals ● Importing Financials ● Importing Put & Call ● Streaming Real-Time ●

▻ Stock Fundamentals, Metadata, Performance Tricks

```
import_all()
```

⊛ **info**: This shows all the fields available with the `info` method

Example of one of the fields in the info

```
disney.info['longBusinessSummary']
```

```
'The Walt Disney Company, together with its subsidiaries, operates as an entertainment company worldwide. It operates through two segments, Disney Media and Entertainment
```

Distribution; and Disney Parks, Experiences and Products. The company engages in the film and episodic television content production and distribution activities, as well as operates television networks under the ABC, Disney, ESPN, Freeform, FX, Fox, National Geographic, and Star brands; and studios that produces films under the Walt Disney Pictures, Twentieth Century Studios, Marvel, Lucasfilm, Pixar, and Searchlight Pictures banners. It also offers direct-to-consumer streaming services through Disney+, Disney+ Hotstar, ESPN+, Hulu, and Star+; sale/licensing of film and television content to third-party television and subscription video-on-demand services; theatrical, home entertainment, and music distribution services; staging and licensing of live entertainment events; and post-production services by Industrial Light & Magic and Skywalker Sound. In addition, the company operates theme parks and resorts, such as Walt Disney World Resort in Florida; Disneyland Resort in California; Disneyland Paris; Hong Kong Disneyland Resort; and Shanghai Disney Resort; Disney Cruise Line, Disney Vacation Club, National Geographic Expeditions, and Adventures by Disney, as well as Aulani, a Disney resort and spa in Hawaii. Further, it licenses its intellectual property to a third party for the operations of the Tokyo Disney Resort; provides consumer products, including licensing of trade names, characters, visual, literary, and other IP for use on merchandise, published materials, and games; operates a direct-to-home satellite distribution platform; sells branded merchandise through retail, online, and wholesale businesses; and develops and publishes books, comic books, and magazines. The Walt Disney Company was founded in 1923 and is based in Burbank, California.'

Another field

```
disney.info['twoHundredDayAverage']
```

```
104.43735
```

Converting the info to a Series

```
pd.Series(disney.info)
```

```
zip                                                 91521
sector                                 Communication Services
fullTimeEmployees                                  171600
longBusinessSummary    The Walt Disney Company, together with its sub...
city                                              Burbank
                                       ...
coinMarketCapLink                                   None
regularMarketPrice                                 99.91
preMarketPrice                                    100.37
logo_url               https://logo.clearbit.com/thewaltdisneycompany...
trailingPegRatio                                    0.80
Length: 154, dtype: object
```

for-loop to add Microsoft and Facebook to the df:

```
df = pd.Series(disney.info, name = 'DIS').to_frame().T

tickers = ['MSFT', 'META']

for company in tickers:
    df.loc[f'{company}'] = pd.Series(yf.Ticker(company).info)

df
```

	zip	sector	fullTimeEmployees	longBusinessSummary	city	phone	state	country	company
DIS	91521	Communication Services	171600	The Walt Disney Company, together with its sub...	Burbank	818 560 1000	CA	United States	
MSFT	98052-6399	Technology	221000	Microsoft Corporation develops, licenses, and ...	Redmond	425 882 8080	WA	United States	
META	94025	Communication Services	87314	Meta Platforms, Inc. develops products that en...	Menlo Park	650 543 4800	CA	United States	

3 rows × 154 columns

⊛ Create a dataframe of info for any ticker list:

```
def company_info_dataframer(ticker_list):
    df = pd.DataFrame(columns = list(yf.Ticker(ticker_list[0]).info.keys()))
    for company in ticker_list:
        df.loc[f'{company}'] = pd.Series(yf.Ticker(company).info)

    return df
```

⊛ Getting "**trending tickers**" from the YFinance site:

- Changing the % Change column to float so that the table can be sorted descending by the % change
- The **top ten positive percent change companies** will be added to the dataframe
- Removing the indexes from the possible stocks to get info about

```
ticker_table = pd.read_html("https://finance.yahoo.com/lookup/")[0]
ticker_table["% Change"] = pd.Series([float(x[:-1]) for x in ticker_table['% Change']])
see(ticker_table, "Yahoo Finance: 30 Trending Stocks")
```

Yahoo Finance: 30 Trending Stocks

	Symbol	Name	Last Price	Change	% Change
0	TSLA	Tesla, Inc.	131.49	9.09	7.43
1	UAL	United Airlines Holdings, Inc.	51.20	-0.45	-0.87
2	AMC	AMC Entertainment Holdings, Inc.	6.07	1.01	19.96
3	GS	The Goldman Sachs Group, Inc.	349.92	-24.08	-6.44

	Symbol	Name	Last Price	Change	% Change
4	NVDA	NVIDIA Corporation	177.02	8.03	4.75
5	BBBY	Bed Bath & Beyond Inc.	4.14	0.48	13.11
6	MRNA	Moderna, Inc.	190.69	-1.31	-0.68
7	GME	GameStop Corp.	21.80	1.31	6.39
8	APE	AMC Entertainment Holdings, Inc.	1.70	0.18	11.84
9	RBLX	Roblox Corporation	37.12	3.91	11.77
10	MS	Morgan Stanley	97.08	5.42	5.91
11	AAL	American Airlines Group Inc.	17.08	0.06	0.35
12	XELA	Exela Technologies, Inc.	0.09	0.02	29.59
13	SPY	SPDR S&P 500 ETF Trust	397.77	-0.73	-0.18
14	ZOM	Zomedica Corp.	0.29	0.04	17.01
15	DAL	Delta Air Lines, Inc.	38.26	0.06	0.16
16	SPCE	Virgin Galactic Holdings, Inc.	5.59	0.40	7.71
17	PFE	Pfizer Inc.	46.08	-1.77	-3.70
18	ATER	Aterian, Inc.	1.33	0.36	37.35
19	RIG	Transocean Ltd.	6.07	0.41	7.24
20	COIN	Coinbase Global, Inc.	54.14	4.16	8.32
21	SOFI	SoFi Technologies, Inc.	5.64	0.20	3.68
22	MSFT	Microsoft Corporation	240.35	1.12	0.47
23	MULN	Mullen Automotive, Inc.	0.30	-0.06	-15.97
24	SNAP	Snap Inc.	9.67	-0.13	-1.33
25	IQ	iQIYI, Inc.	5.59	-1.10	-16.44
26	XPEV	XPeng Inc.	9.36	-0.63	-6.31
27	FUV	Arcimoto, Inc.	6.12	1.46	31.33
28	CCL	Carnival Corporation & plc	10.88	0.37	3.52
29	SQ	Block, Inc.	75.10	3.45	4.82

Creating list of the trending tickers and passing to the dataframe making function:

```
trending_tickers = ticker_table[~ticker_table.Symbol.str.contains('\^')]\
                                .sort_values("% Change", ascending = False).head(10)

trending_tickers = list(trending_tickers.Symbol)
trending_tickers_info = company_info_dataframer(trending_tickers)
```

Choosing a few main columns to display:

```
current_cols = ['shortName', 'sector', 'market', 'fullTimeEmployees',
                'city', 'country', 'currentPrice']
```

```
see(trending_tickers_info[current_cols].sort_values("currentPrice", ascending=False),
    "Top 10 Daily Trending Stocks: Company Information")
```

Top 10 Daily Trending Stocks: Company Information

	shortName	sector	market	fullTimeEmployees	city	country	currentPrice
COIN	Coinbase Global, Inc.	Technology	us_market	3730	Wilmington	United States	54.14
RBLX	Roblox Corporation	Communication Services	us_market	1600	San Mateo	United States	37.12
FUV	Arcimoto, Inc.	Consumer Cyclical	us_market	325	Eugene	United States	6.12
AMC	AMC Entertainment Holdings, Inc	Communication Services	us_market	3046	Leawood	United States	6.07
SPCE	Virgin Galactic Holdings, Inc.	Industrials	us_market	804	Tustin	United States	5.59
BBBY	Bed Bath & Beyond Inc.	Consumer Cyclical	us_market	32000	Union	United States	4.14
APE	AMC Entertainment Holdings, Inc	Communication Services	us_market	3046	Leawood	United States	1.70
ATER	Aterian, Inc.	Consumer Cyclical	us_market	156	New York	United States	1.33
ZOM	Zomedica Corp.	Healthcare	us_market	47	Ann Arbor	United States	0.29
XELA	Exela Technologies, Inc.	Technology	us_market	16500	Irving	United States	0.09

Sections: ● Top ● Download & Import ● Historical Data ● Setting Date Range ● High Frequency ● Splits & Dividends ● Exporting ● Multiple Stocks ● Importing Indexes ● Currency Exchange ● Cryptocurrencies ● Mutual & Exchange Traded Funds ● Treasury Yields ● Ticker Object & Docs ● Stock Fundamentals ● Importing Financials ● Importing Put & Call ● Streaming Real-Time ●

▻ Importing Financials: Balance Sheets, Profit and Loss, Cashflow

```
import_all()
```

Must start with a ticker object:

```
disney
```

```
yfinance.Ticker object <DIS>
```

⊛ Balance Sheets

```
see(disney.balance_sheet.head(10), 'disney.balance_sheet.head(10)')
```

disney.balance_sheet.head(10)

	2022-09-30 00:00:00	2021-09-30 00:00:00	2020-09-30 00:00:00	2019-09-30 00:00:00
Total Assets	203,631,000,000.00	203,609,000,000.00	201,549,000,000.00	193,984,000,000.00
Current Assets	29,098,000,000.00	33,657,000,000.00	35,251,000,000.00	28,124,000,000.00
Cash Cash Equivalents And Short Term Investments	11,615,000,000.00	15,959,000,000.00	17,914,000,000.00	5,418,000,000.00
Cash And Cash Equivalents	11,615,000,000.00	15,959,000,000.00	17,914,000,000.00	5,418,000,000.00
Receivables	12,652,000,000.00	13,367,000,000.00	12,708,000,000.00	15,481,000,000.00
Accounts Receivable	10,811,000,000.00	11,177,000,000.00	11,299,000,000.00	12,882,000,000.00
Other Receivables	1,999,000,000.00	2,360,000,000.00	1,835,000,000.00	2,894,000,000.00
Receivables Adjustments Allowances	-158,000,000.00	-170,000,000.00	-426,000,000.00	-295,000,000.00
Inventory	1,742,000,000.00	1,331,000,000.00	1,583,000,000.00	1,649,000,000.00
Prepaid Assets	1,890,000,000.00	2,183,000,000.00	2,171,000,000.00	4,597,000,000.00

⊛ Financials: Profit and Loss Statement

```
see(disney.financials.head(10), 'disney.financials.head(10)')
```

disney.financials.head(10)

	2022-09-30 00:00:00	2021-09-30 00:00:00	2020-09-30 00:00:00	2019-09-30 00:00:00
Total Revenue	82,722,000,000.00	67,418,000,000.00	65,388,000,000.00	69,570,000,000.00
Operating Revenue	82,722,000,000.00	67,418,000,000.00	65,388,000,000.00	69,570,000,000.00
Cost Of Revenue	54,401,000,000.00	45,131,000,000.00	43,880,000,000.00	42,018,000,000.00
Gross Profit	28,321,000,000.00	22,287,000,000.00	21,508,000,000.00	27,552,000,000.00
Operating Expense	21,551,000,000.00	18,628,000,000.00	17,714,000,000.00	15,701,000,000.00
Selling General And Administration	16,388,000,000.00	13,517,000,000.00	12,369,000,000.00	11,541,000,000.00
Depreciation Amortization Depletion Income Statement	5,163,000,000.00	5,111,000,000.00	5,345,000,000.00	4,160,000,000.00
Depreciation And Amortization In Income Statement	5,163,000,000.00	5,111,000,000.00	5,345,000,000.00	4,160,000,000.00
Operating Income	6,770,000,000.00	3,659,000,000.00	3,794,000,000.00	11,851,000,000.00
Net Non Operating Interest Income Expense	-1,397,000,000.00	-1,406,000,000.00	-1,491,000,000.00	-978,000,000.00

⊛ Cashflow Statement

```
see(disney.cashflow.head(10), 'disney.cashflow.head(10)')
```

disney.cashflow.head(10)

	2022-09-30 00:00:00	2021-09-30 00:00:00	2020-09-30 00:00:00	2019-09-30 00:00:00
Operating Cash Flow	6,010,000,000.00	5,567,000,000.00	7,618,000,000.00	6,606,000,000.00

	2022-09-30 00:00:00	2021-09-30 00:00:00	2020-09-30 00:00:00	2019-09-30 00:00:00
Cash Flow From Continuing Operating Activities	6,002,000,000.00	5,566,000,000.00	7,616,000,000.00	5,984,000,000.00
Net Income From Continuing Operations	3,553,000,000.00	2,536,000,000.00	-2,442,000,000.00	10,913,000,000.00
Operating Gains Losses	518,000,000.00	-277,000,000.00	-1,571,000,000.00	-4,691,000,000.00
Gain Loss On Sale Of Business	NaN	NaN	NaN	-4,794,000,000.00
Gain Loss On Investment Securities	714,000,000.00	-332,000,000.00	-920,000,000.00	NaN
Earnings Losses From Equity Investments	-816,000,000.00	-761,000,000.00	-651,000,000.00	103,000,000.00
Pension And Employee Benefit Expense	620,000,000.00	816,000,000.00	NaN	NaN
Depreciation Amortization Depletion	5,163,000,000.00	5,111,000,000.00	5,345,000,000.00	4,160,000,000.00
Depreciation And Amortization	5,163,000,000.00	5,111,000,000.00	5,345,000,000.00	4,160,000,000.00

Creating CSV files of **financials** (profits and losses) from a list of tickers:

```
trending_tickers
```

```
['ATER', 'FUV', 'XELA', 'AMC', 'ZOM', 'BBBY', 'APE', 'RBLX', 'COIN', 'SPCE']
```

```
for ticker in trending_tickers[0:2]:
    yf.Ticker(ticker).financials.to_csv(f"{ticker}.csv")
```

⊛ Creating a combined **most recent financials** dataframe from a list of tickers:

```
def create_financials_df(ticker_list):
    recent_financials = pd.DataFrame(columns = ticker_list)

    for ticker in ticker_list:
        ticker_df = yf.Ticker(ticker).financials
        column_name = str(ticker_df.columns[0])
        recent_financials[f"{ticker}"] = ticker_df[column_name]

    return recent_financials
```

Using the function above on the first tickers in the trending list:

```
trending_financials_df = create_financials_df(trending_tickers[0:3])
```

```
see(trending_financials_df.head(5), 'trending_financials_df.head(5)')
```

trending_financials_df.head(5)

	ATER	FUV	XELA
Total Revenue	247,767,000.00	4,386,222.00	1,166,606,000.00
Operating Revenue	247,767,000.00	4,386,222.00	1,166,606,000.00

	ATER	FUV	XELA
Cost Of Revenue	125,904,000.00	17,148,948.00	889,095,000.00
Gross Profit	121,863,000.00	-12,762,726.00	277,511,000.00
Operating Expense	182,305,000.00	32,054,635.00	256,122,000.00

Sections: ● Top ● Download & Import ● Historical Data ● Setting Date Range ● High Frenquency ● Splits & Dividends ● Exporting ● Multiple Stocks ● Importing Indexes ● Currency Exchange ● Cryptocurrencies ● Mutual & Exchange Traded Funds ● Treasury Yields ● Ticker Object & Docs ● Stock Fundamentals ● Importing Financials ● Importing Put & Call ● Streaming Real-Time ●

▻ Importing Put & Call Options (Stocks)

- These are complicated financial derivatives

```
import_all()
```

Must start with a ticker object:

```
disney
```

```
yfinance.Ticker object <DIS>
```

⊛ **.option_chain()** returns a tuple with two dataframes, to first being call and second put options

⊛ Disney Call Options:

```
disney_calls = disney.option_chain()[0]
head_tail_vert(disney_calls, 5, "Disney Call Options")
```

Disney Call Options: head(5)

	contractSymbol	lastTradeDate	strike	lastPrice	bid	ask	change	percentChange	volume	openInterest
0	DIS230120C00050000	2023-01-17	50.00	50.25	49.70	50.15	12.90	34.54	1.00	128
1	DIS230120C00055000	2022-12-12	55.00	39.30	41.10	41.65	0.00	0.00	4.00	36
2	DIS230120C00060000	2023-01-17	60.00	39.85	39.80	40.60	0.05	0.13	2.00	425
3	DIS230120C00065000	2023-01-10	65.00	30.15	34.75	35.25	0.00	0.00	2.00	291
4	DIS230120C00070000	2023-01-17	70.00	30.00	29.80	30.25	0.95	3.27	9.00	503

Disney Call Options: tail(5)

	contractSymbol	lastTradeDate	strike	lastPrice	bid	ask	change	percentChange	volume	openInterest

	contractSymbol	lastTradeDate	strike	lastPrice	bid	ask	change	percentChange	volume	openInterest
67	DIS230120C00250000	2023-01-17	250.00	0.01	0.00	0.01	0.00	0.00	10.00	7351
68	DIS230120C00260000	2023-01-17	260.00	0.01	0.00	0.01	0.00	0.00	8.00	2405
69	DIS230120C00270000	2022-11-03	270.00	0.01	0.00	0.01	0.00	0.00	100.00	1742
70	DIS230120C00280000	2023-01-17	280.00	0.01	0.00	0.01	0.00	0.00	20.00	2972
71	DIS230120C00290000	2023-01-17	290.00	0.01	0.00	0.00	0.00	0.00	1.00	9007

⊛ Disney Put Options:

```
disney_puts = disney.option_chain()[1]
head_tail_vert(disney_puts, 5, "Disney Put Options")
```

Disney Put Options: head(5)

	contractSymbol	lastTradeDate	strike	lastPrice	bid	ask	change	percentChange	volume	openInterest	i
0	DIS230120P00050000	2023-01-11	50.00	0.01	0.00	0.01	0.00	0.00	14	4204	
1	DIS230120P00055000	2023-01-04	55.00	0.01	0.00	0.01	0.00	0.00	10	1133	
2	DIS230120P00060000	2023-01-11	60.00	0.01	0.00	0.01	0.00	0.00	2	2617	
3	DIS230120P00065000	2023-01-13	65.00	0.01	0.00	0.01	0.00	0.00	417	9615	
4	DIS230120P00070000	2023-01-17	70.00	0.01	0.00	0.01	0.00	0.00	1	8822	

Disney Put Options: tail(5)

	contractSymbol	lastTradeDate	strike	lastPrice	bid	ask	change	percentChange	volume	openInt
61	DIS230120P00250000	2022-12-01	250.00	153.10	162.70	163.55	0.00	0.00	2	
62	DIS230120P00260000	2022-09-14	260.00	148.15	164.85	165.55	0.00	0.00	125	
63	DIS230120P00270000	2022-12-01	270.00	173.10	182.55	183.60	0.00	0.00	15	
64	DIS230120P00280000	2022-11-17	280.00	188.80	189.10	190.35	0.00	0.00	34	
65	DIS230120P00290000	2023-01-13	290.00	190.85	189.15	191.05	0.00	0.00	2	

Sections: ● Top ● Download & Import ● Historical Data ● Setting Date Range ● High Frenquency ● Splits & Dividends ● Exporting ● Multiple Stocks ● Importing Indexes ● Currency Exchange ● Cryptocurrencies ● Mutual Funds & Exchange Traded Funds ● Treasury Yields ● Ticker Object & Docs ● Stock Fundamentals ● Importing Financials ● Importing Put & Call ● Streaming Real-Time

---

▻ Streaming Real-Time Data

```
import_all()
import time
```

⊛ Creating a simple **live-stream for Euro-USD**

```
yf.download("EURUSD=X", interval = "1m", period = "1d").tail(5)
```

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

Datetime	Open	High	Low	Close	Adj Close	Volume
2023-01-17 22:34:00+00:00	1.08	1.08	1.08	1.08	1.08	0
2023-01-17 22:35:00+00:00	1.08	1.08	1.08	1.08	1.08	0
2023-01-17 22:36:00+00:00	1.08	1.08	1.08	1.08	1.08	0
2023-01-17 22:37:00+00:00	1.08	1.08	1.08	1.08	1.08	0
2023-01-17 22:38:00+00:00	1.08	1.08	1.08	1.08	1.08	0

```
stream_eurusd = yf.download("EURUSD=X", interval = "1m", period = "1d")
```

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

Getting the most recent data:

- last row of data, close column

⊛ **Automating** with a while-loop

- the 5 passed to `time()` denotes how many seconds to wait before running the code again

```
# while True:
#   time.sleep(5)
#   stream_eurusd = yf.download("EURUSD=X", interval = "1m", period = "1d")
#   see((stream_eurusd.index[-1],stream_eurusd.iloc[-1, 3]))
```

⊛ **Function to get live-stream any ticker**:

```
def live_stream(ticker, delay=60, iterations=10,
                currency="dollars", month="word"):

    def timestamp_to_local(timestamp):
        timestamp = pd.to_datetime(timestamp, utc=True).tz_convert('US/Eastern')
        return timestamp

    see(f"The following is the data for {ticker} every {delay} seconds for {delay * ite

    for iteration in range(iterations):
        time.sleep(delay)
        stream = yf.download(ticker, interval = "1m", period = "1d")

        time_stamp = timestamp_to_local(stream.index[-1])
```

```
        month_num, day, year, hour, minute = (time_stamp.month,
                                              time_stamp.day,
                                              time_stamp.year,
                                              time_stamp.hour,
                                              time_stamp.minute)

        if currency == "dollars":
            price = f"${stream.iloc[-1, 3]:.4f}"
        else:
            price = f"{stream.iloc[-1, 3]:.4f}"

        if month == "word":
            month = time_stamp.month_name()[:3]
            see(f"{ticker}: {price} {currency} at {hour}:{minute:02d} on {month} {day},
        elif isinstance(month, str):
            month = time_stamp.month_name()[:3]
            see(f"{ticker}: {price} {currency} at {hour}:{minute:02d} on {month} {day},
        else:
            month = month_num
            see(f"{ticker}: {price} {currency} at {hour}:{minute:02d} on {month}/{day}/
```

```
live_stream("NVDA", 5, 10, month="number")
```

The following is the data for NVDA every 5 seconds for 50 seconds.

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

NVDA: $177.0200 dollars at 16:00 on Jan 17, 2023.

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

NVDA: $177.0200 dollars at 16:00 on Jan 17, 2023.

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

NVDA: $177.0200 dollars at 16:00 on Jan 17, 2023.

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

NVDA: $177.0200 dollars at 16:00 on Jan 17, 2023.

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

NVDA: $177.0200 dollars at 16:00 on Jan 17, 2023.

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

NVDA: $177.0200 dollars at 16:00 on Jan 17, 2023.

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

NVDA: $177.0200 dollars at 16:00 on Jan 17, 2023.

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

NVDA: $177.0200 dollars at 16:00 on Jan 17, 2023.

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

NVDA: $177.0200 dollars at 16:00 on Jan 17, 2023.

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

NVDA: $177.0200 dollars at 16:00 on Jan 17, 2023.

Sections: ● Top ● Download & Import ● Historical Data ● Setting Date Range ● High Frenquency ● Splits & Dividends ● Exporting ● Multiple Stocks ● Importing Indexes ● Currency Exchange ● Cryptocurrencies ● Mutual Funds & Exchange Traded Funds ● Treasury Yields ● Ticker Object & Docs ● Stock Fundamentals ● Importing Financials ● Importing Put & Call ● Streaming Real-Time

---