yfinance DOC

yfinance DOC

1. Pasta advanced

1.1. Caching – yfinance

O yfinance permite caching para reduzir chamadas repetidas e aliviar o scraping. É possível usar pacotes como `requests_cache` e `requests_ratelimiter` para:

- Criar sessões cacheadas (`requests_cache.CachedSession`)
- Controlar taxa de requisições (`requests_ratelimiter`, `pyrate_limiter`)

Há também um cache persistente de timezones e cookies, em local padrão do sistema (ex: `~/.cache/py-yfinance` em Linux). Para customizar:

```
import yfinance as yf
yf.set_tz_cache_location("custom/cache/location")
```

1.2. EquityQuery – yfinance

```
class yfinance.EquityQuery(operator, operand)
```

Constrói filtros para ações com base em critérios (campos) como região, setor, exchange e peer_group. As operações possíveis incluem:

- `EQ`, `IS-IN`, `BTWN`, `GT`, `LT`, `GTE`, `LTE`
- `AND`, `OR`

Possui atributos como:

- **`valid_fields`**: dicionário que agrupa campos por categoria (price, trading, valuation, etc.)
- **`valid_values`**: mapeia certos campos a valores pré-definidos (p.ex. "exchange")

Exemplo:

```
from yfinance import EquityQuery
query = EquityQuery('and', [
  EquityQuery('is-in', ['exchange', 'NMS', 'NYQ']),
  EquityQuery('lt', ["epsgrowth.lasttwelvemonths", 15])
])
```

Em geral, usada com `yfinance.screen(...)` para execução de screeners.

1.3. Functions and Utilities – yfinance

Descreve funções utilitárias, incluindo:

- `download(...)`: para baixar dados de múltiplos tickers
- `enable_debug_mode()`: ativa modo debug
- `set_tz_cache_location(...)`: define o local de cache de timezone

1.4. FundsData – yfinance

```
class yfinance.scrapers.funds.FundsData(data, symbol, proxy=None)
```

ETF e Mutual Funds Data (módulos consultados: `quoteType`, `summaryProfile`, `fundProfile`, `topHoldings`).

Atributos principais:

- `asset_classes`, `bond_holdings`, `bond_ratings`, `equity_holdings`
- `fund_operations`, `fund_overview`, `sector_weightings`, `top_holdings`

Methods:

- `quote_type()`: Retorna a identificação do tipo ("ETF", "MUTUALFUND", etc.)

(Observação: Havia dois PDFs "FundsData class" e "FundsData" com conteúdo idêntico, unificado aqui.)

1.5. Industry – yfinance

```
class yfinance.Industry(key, session=None, proxy=None)
```

Representa uma indústria dentro de um setor. Parâmetros:

- `key (str)`: identificador da indústria
- `session`, `proxy`: para customizar requisições

Atributos principais:

- `key`, `name`, `overview`, `research_reports`
- `sector_key`, `sector_name`
- `symbol`, `ticker`
- `top_companies`, `top_growth_companies`, `top_performing_companies`

1.6. Logging – yfinance

Por padrão, apenas erros são exibidos. Para habilitar debug:

```
import yfinance as yf
yf.enable_debug_mode()
```

1.7. Multi-Level Column Index – yfinance

Discussão sobre como lidar com DataFrames que possuem colunas de múltiplos níveis. Ao salvar/ler CSV, pode ser necessário manipular o MultiIndex adequadamente.

1.8. PriceHistory class – yfinance

```
class yfinance.scrapers.history.PriceHistory(data, ticker, tz, session=None, proxy=None)
```

Métodos principais:

- `get_actions()`, `get_dividends()`, `get_splits()`, etc.
- `history(...)`: similar a `Ticker.history`.

1.9. Proxy Server – yfinance

Permite realizar downloads via proxy:

```
msft = yf.Ticker("MSFT")
msft.history(..., proxy="PROXY_SERVER")
```

1.10. Screener & Query – yfinance

Descreve classes `EquityQuery`, `FundQuery` (filtros para ações/fundos) e a função `screen(...)` para execução de screeners customizados ou pré-definidos.

1.11. yfinance.download — yfinance

```
yfinance.download(
   tickers,
   ...
) → DataFrame
```

Baixa dados de mercado do Yahoo Finance para um ou mais tickers (multi-index ou single).
Principais parâmetros: `period`, `start`, `end`, `interval`, etc.

1.12. yfinance.enable_debug_mode — yfinance

```
yfinance.enable_debug_mode()
```

Ativa modo debug, com logs adicionais.

1.13. yfinance.screen — yfinance

```
yfinance.screen(
   query, offset=None, size=None, ...
)
```

Executa "screener" usando queries (`EquityQuery`, `FundQuery`) ou nomes de queries pré-definidas.

1.14. yfinance.set_tz_cache_location — yfinance

```
yfinance.set_tz_cache_location(cache_dir)
```

Define o diretório de cache de timezone.

---

2. Pasta market

2.1. Market — yfinance

class yfinance.Market(market, session=None, proxy=None, timeout=30)

Permite acessar dados de um "market" específico no Yahoo Finance de modo Pythonic. Há 8 mercados disponíveis:

- US
- GB
- ASIA
- EUROPE
- RATES
- COMMODITIES
- CURRENCIES
- CRYPTOCURRENCIES

Atributos principais (citados em "Market — yfinance2.pdf"):

- `status`
- `summary`

Methods:

- `__init__(market: str, session=None, proxy=None, timeout=30)`

Uso de exemplo:

```
import yfinance as yf

europe = yf.Market("EUROPE")
status = europe.status
summary = europe.summary
```

---

3. Pasta search

3.1. Search — yfinance

```
class yfinance.Search(
    query,
    max_results=8,
    news_count=8,
    ...
)
```

Faz pesquisa no Yahoo Finance, retornando cotações, notícias etc.

3.2. Search & News — yfinance

Mostra como usar `yf.Search()` para buscar quotes e notícias.

Exemplo:

```
import yfinance as yf
quotes = yf.Search("AAPL").quotes
news   = yf.Search("Google").news
```

---

4. Pasta sector

4.1. Sector — yfinance

```
class yfinance.Sector(key, session=None, proxy=None)
```

Permite acessar dados de um setor (ex. "technology"). Atributos: `top_etfs`, `top_mutual_funds`, `industries`, etc.

4.2. Sector and Industry — yfinance

Complementa, mostrando também `Industry`.

---

5. Pasta ticker

5.1. Stock — yfinance

Documento que descreve métodos e atributos do `Ticker` relacionados a splits, dividendos, capital gains etc. Principais referências:

- `get_isin()`, `isin`
- `history(...)`
- `get_history_metadata()`
- `get_dividends()`, `dividends`
- `get_splits()`, `splits`
- `get_actions()`, `actions`
- `get_capital_gains()`, `capital_gains`

- `get_shares_full()`
- `get_info()`, `info`
- `get_fast_info()`, `fast_info`
- `get_news()`, `news`

5.2. Ticker — yfinance

```
class yfinance.Ticker(ticker, session=None, proxy=None)
```

Propriedades e métodos gerais: `actions`, `info`, `news`, `isin`, `dividends`, etc.

- `history(*args, **kwargs)`: retorna DataFrame de preços.
- `get_actions()`, `get_dividends()`, `get_splits()`, `get_info()`, `get_news()`, etc.

5.3. Ticker and Tickers — yfinance

Explica `Ticker(...)` (um ativo) e `Tickers(...)` (múltiplos ativos). Exemplo:

```
import yfinance as yf
msft = yf.Ticker("MSFT")
...
tickers = yf.Tickers("msft aapl goog")
```

5.4. Tickers — yfinance

```
class yfinance.Tickers(tickers, session=None)
```

Gerencia múltiplos tickers em um só objeto, com métodos como `download(...)`, `history(...)`, etc.

5.5. yfinance.Ticker.actions

```
@property Ticker.actions: pandas.DataFrame
```

Combina splits e dividendos em um DataFrame.

5.6. yfinance.Ticker.capital_gains

```
@property Ticker.capital_gains: pandas.Series
```

Série com "capital gains" (usualmente para fundos).

5.7. yfinance.Ticker.dividends

```
@property Ticker.dividends: pandas.Series
```

Série de dividendos, similar a `get_dividends()`.

5.8. yfinance.Ticker.fast_info

```
@property Ticker.fast_info
```

Infos cruciais (preço, volume) rapidamente.

5.9. yfinance.Ticker.get_actions

Ticker.get_actions(proxy=None) → pandas.Series

Combina splits e dividendos em um Series.

5.10. yfinance.Ticker.get_capital_gains

Ticker.get_capital_gains(proxy=None) → pandas.Series

Retorna série de ganhos de capital do ativo.

5.11. yfinance.Ticker.get_dividends

Ticker.get_dividends(proxy=None) → pandas.Series

Retorna histórico de dividendos.

5.12. yfinance.Ticker.get_fast_info

Ticker.get_fast_info(proxy=None)

Dados rápidos do ativo (preço, capitalização, etc.).

5.13. yfinance.Ticker.get_history_metadata

Ticker.get_history_metadata(proxy=None) → dict

Retorna metadados sobre histórico de preços (timezone, etc.).

5.14. yfinance.Ticker.get_info

Ticker.get_info(proxy=None) → dict

Semelhante à propriedade `info`, mas obtido via método.

5.15. yfinance.Ticker.get_isin

Ticker.get_isin(proxy=None) → str | None

Obtém ISIN do ativo, se disponível.

5.16. yfinance.Ticker.get_news

Ticker.get_news(count=10, tab='news', proxy=None) → list

Retorna lista de notícias ("news", "all" ou "press releases").

5.17. yfinance.Ticker.get_shares_full

Ticker.get_shares_full(start=None, end=None, proxy=None)

Infos sobre ações em circulação; possivelmente histórico se suportado.

5.18. yfinance.Ticker.get_splits

```
Ticker.get_splits(proxy=None) → pandas.Series
```

Retorna splits do ativo.

5.19. yfinance.Ticker.history

```
Ticker.history(*args, **kwargs) → pandas.DataFrame
```

Histórico de preços, OHLCV.

5.20. yfinance.Ticker.info

```
@property Ticker.info: dict
```

Propriedade que devolve um dicionário com dados diversos do ativo.

5.21. yfinance.Ticker.isin

```
@property Ticker.isin
```

Retorna o ISIN, se disponível.

5.22. yfinance.Ticker.news

```
@property Ticker.news: list
```

Lista de notícias sobre o ativo (sem parâmetros).

5.23. yfinance.Ticker.splits

```
@property Ticker.splits: pandas.Series
```

Retorna splits do ativo no formato Series.

Documento atualizado com Market — yfinance e Market — yfinance2.

---

yfinance – Documento Unificado Organizado por Pastas

A seguir está o documento completo com todos os PDFs integrados, agora adicionando a pasta **market** conforme solicitado.

---

1. Pasta advanced

1.1. Caching – yfinance

O yfinance permite caching para reduzir chamadas repetidas e aliviar o scraping. É possível usar pacotes como `requests_cache` e `requests_ratelimiter` para:

- Criar sessões cacheadas (`requests_cache.CachedSession`)
- Controlar taxa de requisições (`requests_ratelimiter`, `pyrate_limiter`)

Há também um cache persistente de timezones e cookies, em local padrão do sistema (ex: `~/.cache/py-yfinance` em Linux). Para customizar:

```
import yfinance as yf
yf.set_tz_cache_location("custom/cache/location")
```

1.2. EquityQuery — yfinance

class yfinance.EquityQuery(operator, operand)

Constrói filtros para ações com base em critérios (campos) como região, setor, exchange e peer_group. As operações possíveis incluem:

- `EQ`, `IS-IN`, `BTWN`, `GT`, `LT`, `GTE`, `LTE`
- `AND`, `OR`

Possui atributos como:

- **`valid_fields`**: dicionário que agrupa campos por categoria (price, trading, valuation, etc.)
- **`valid_values`**: mapeia certos campos a valores pré-definidos (p.ex. "exchange")

Exemplo:

```
from yfinance import EquityQuery
query = EquityQuery('and', [
  EquityQuery('is-in', ['exchange', 'NMS', 'NYQ']),
  EquityQuery('lt', ["epsgrowth.lasttwelvemonths", 15])
])
```

Em geral, usada com `yfinance.screen(...)` para execução de screeners.

1.3. Functions and Utilities — yfinance

Descreve funções utilitárias, incluindo:

- `download(...)`: para baixar dados de múltiplos tickers
- `enable_debug_mode()`: ativa modo debug
- `set_tz_cache_location(...)`: define o local de cache de timezone

1.4. FundsData — yfinance

class yfinance.scrapers.funds.FundsData(data, symbol, proxy=None)

ETF e Mutual Funds Data (módulos consultados: `quoteType`, `summaryProfile`, `fundProfile`, `topHoldings`).

Atributos principais:

- `asset_classes`, `bond_holdings`, `bond_ratings`, `equity_holdings`
- `fund_operations`, `fund_overview`, `sector_weightings`, `top_holdings`

Methods:

- `quote_type()`: Retorna a identificação do tipo ("ETF", "MUTUALFUND", etc.)

(Observação: Havia dois PDFs "FundsData class" e "FundsData" com conteúdo idêntico, unificado aqui.)

1.5. Industry — yfinance

class yfinance.Industry(key, session=None, proxy=None)

Representa uma indústria dentro de um setor. Parâmetros:

- `key (str)`: identificador da indústria
- `session`, `proxy`: para customizar requisições

Atributos principais:

- `key`, `name`, `overview`, `research_reports`

- `sector_key`, `sector_name`
- `symbol`, `ticker`
- `top_companies`, `top_growth_companies`, `top_performing_companies`

1.6. Logging – yfinance

Por padrão, apenas erros são exibidos. Para habilitar debug:

```
import yfinance as yf
yf.enable_debug_mode()
```

1.7. Multi-Level Column Index – yfinance

Discussão sobre como lidar com DataFrames que possuem colunas de múltiplos níveis. Ao salvar/ler CSV, pode ser necessário manipular o MultiIndex adequadamente.

1.8. PriceHistory class – yfinance

```
class yfinance.scrapers.history.PriceHistory(data, ticker, tz, session=None, proxy=None)
```

Métodos principais:

- `get_actions()`, `get_dividends()`, `get_splits()`, etc.
- `history(...)`: similar a `Ticker.history`.

1.9. Proxy Server – yfinance

Permite realizar downloads via proxy:

```
msft = yf.Ticker("MSFT")
msft.history(..., proxy="PROXY_SERVER")
```

1.10. Screener & Query – yfinance

Descreve classes `EquityQuery`, `FundQuery` (filtros para ações/fundos) e a função `screen(...)` para execução de screeners customizados ou pré-definidos.

1.11. yfinance.download – yfinance

```
yfinance.download(
    tickers,
    ...
) → DataFrame
```

Baixa dados de mercado do Yahoo Finance para um ou mais tickers (multi-index ou single). Principais parâmetros: `period`, `start`, `end`, `interval`, etc.

1.12. yfinance.enable_debug_mode – yfinance

```
yfinance.enable_debug_mode()
```

Ativa modo debug, com logs adicionais.

1.13. yfinance.screen – yfinance

```
yfinance.screen(
    query, offset=None, size=None, ...
```

```
)
```

Executa "screener" usando queries (`EquityQuery`, `FundQuery`) ou nomes de queries pré-definidas.

1.14. yfinance.set_tz_cache_location — yfinance

```
yfinance.set_tz_cache_location(cache_dir)
```

Define o diretório de cache de timezone.

---

2. Pasta market

2.1. Market — yfinance

class yfinance.Market(market, session=None, proxy=None, timeout=30)

Permite acessar dados de um "market" específico no Yahoo Finance de modo Pythonic. Há 8 mercados disponíveis:

- US
- GB
- ASIA
- EUROPE
- RATES
- COMMODITIES
- CURRENCIES
- CRYPTOCURRENCIES

Atributos principais (citados em "Market — yfinance2.pdf"):

- `status`
- `summary`

Methods:

- `__init__(market: str, session=None, proxy=None, timeout=30)`

Uso de exemplo:

```
import yfinance as yf

europe = yf.Market("EUROPE")
status = europe.status
summary = europe.summary
```

---

3. Pasta search

3.1. Search — yfinance

```
class yfinance.Search(
    query,
    max_results=8,
    news_count=8,
    ...
)
```

Faz pesquisa no Yahoo Finance, retornando cotações, notícias etc.

3.2. Search & News — yfinance

Mostra como usar `yf.Search()` para buscar quotes e notícias.

Exemplo:

```
import yfinance as yf
quotes = yf.Search("AAPL").quotes
news   = yf.Search("Google").news
```

---

4. Pasta sector

4.1. Sector — yfinance

```
class yfinance.Sector(key, session=None, proxy=None)
```

Permite acessar dados de um setor (ex. "technology"). Atributos: `top_etfs`, `top_mutual_funds`, `industries`, etc.

4.2. Sector and Industry — yfinance

Complementa, mostrando também `Industry`.

---

5. Pasta ticker

5.1. Stock — yfinance

Documento que descreve métodos e atributos do `Ticker` relacionados a splits, dividendos, capital gains etc. Principais referências:

- `get_isin()`, `isin`
- `history(...)`
- `get_history_metadata()`
- `get_dividends()`, `dividends`
- `get_splits()`, `splits`
- `get_actions()`, `actions`
- `get_capital_gains()`, `capital_gains`
- `get_shares_full()`
- `get_info()`, `info`
- `get_fast_info()`, `fast_info`
- `get_news()`, `news`

5.2. Ticker — yfinance

```
class yfinance.Ticker(ticker, session=None, proxy=None)
```

Propriedades e métodos gerais: `actions`, `info`, `news`, `isin`, `dividends`, etc.

- `history(*args, **kwargs)`: retorna DataFrame de preços.
- `get_actions()`, `get_dividends()`, `get_splits()`, `get_info()`, `get_news()`, etc.

5.3. Ticker and Tickers — yfinance

Explica `Ticker(...)` (um ativo) e `Tickers(...)` (múltiplos ativos). Exemplo:

```
import yfinance as yf
msft = yf.Ticker("MSFT")
...
tickers = yf.Tickers("msft aapl goog")
```

5.4. Tickers — yfinance

```
class yfinance.Tickers(tickers, session=None)
```

Gerencia múltiplos tickers em um só objeto, com métodos como `download(...)`, `history(...)`, etc.

5.5. yfinance.Ticker.actions

```
@property Ticker.actions: pandas.DataFrame
```

Combina splits e dividendos em um DataFrame.

5.6. yfinance.Ticker.capital_gains

```
@property Ticker.capital_gains: pandas.Series
```

Série com "capital gains" (usualmente para fundos).

5.7. yfinance.Ticker.dividends

```
@property Ticker.dividends: pandas.Series
```

Série de dividendos, similar a `get_dividends()`.

5.8. yfinance.Ticker.fast_info

```
@property Ticker.fast_info
```

Infos cruciais (preço, volume) rapidamente.

5.9. yfinance.Ticker.get_actions

```
Ticker.get_actions(proxy=None) → pandas.Series
```

Combina splits e dividendos em um Series.

5.10. yfinance.Ticker.get_capital_gains

```
Ticker.get_capital_gains(proxy=None) → pandas.Series
```

Retorna série de ganhos de capital do ativo.

5.11. yfinance.Ticker.get_dividends

```
Ticker.get_dividends(proxy=None) → pandas.Series
```

Retorna histórico de dividendos.

5.12. yfinance.Ticker.get_fast_info

```
Ticker.get_fast_info(proxy=None)
```

Dados rápidos do ativo (preço, capitalização, etc.).

5.13. yfinance.Ticker.get_history_metadata

```
Ticker.get_history_metadata(proxy=None) → dict
```

Retorna metadados sobre histórico de preços (timezone, etc.).

5.14. yfinance.Ticker.get_info

```
Ticker.get_info(proxy=None) → dict
```

Semelhante à propriedade `info`, mas obtido via método.

5.15. yfinance.Ticker.get_isin

```
Ticker.get_isin(proxy=None) → str | None
```

Obtém ISIN do ativo, se disponível.

5.16. yfinance.Ticker.get_news

```
Ticker.get_news(count=10, tab='news', proxy=None) → list
```

Retorna lista de notícias ("news", "all" ou "press releases").

5.17. yfinance.Ticker.get_shares_full

```
Ticker.get_shares_full(start=None, end=None, proxy=None)
```

Infos sobre ações em circulação; possivelmente histórico se suportado.

5.18. yfinance.Ticker.get_splits

```
Ticker.get_splits(proxy=None) → pandas.Series
```

Retorna splits do ativo.

5.19. yfinance.Ticker.history

```
Ticker.history(*args, **kwargs) → pandas.DataFrame
```

Histórico de preços, OHLCV.

5.20. yfinance.Ticker.info

```
@property Ticker.info: dict
```

Propriedade que devolve um dicionário com dados diversos do ativo.

5.21. yfinance.Ticker.isin

```
@property Ticker.isin
```

Retorna o ISIN, se disponível.

5.22. yfinance.Ticker.news

```
@property Ticker.news: list
```

Lista de notícias sobre o ativo (sem parâmetros).

5.23. yfinance.Ticker.splits

```
@property Ticker.splits: pandas.Series
```

Retorna splits do ativo no formato Series.