# investpy Documentation

***Release 1.0.8***

**Alvaro Bartolome**

**Jan 24, 2022**

# DOCUMENTATION


investpy

CHAPTER

ONE

# INTRODUCTION

investpy is a Python package to retrieve data from Investing.com, which provides data retrieval from up to 39952 stocks, 82221 funds, 11403 ETFs, 2029 currency crosses, 7797 indices, 688 bonds, 66 commodities, 250 certificates, and 4697 cryptocurrencies.

investpy allows the user to download both recent and historical data from all the financial products indexed at Investing.com. **It includes data from all over the world**, from countries such as United States, France, India, Spain, Russia, or Germany, amongst many others.

investpy seeks to be one of the most complete Python packages when it comes to financial data extraction to stop relying on public/private APIs since investpy is **FREE** and has **NO LIMITATIONS**. These are some of the features that currently lead investpy to be one of the most consistent packages when it comes to financial data retrieval.

---

## 1.1 Data Source

Investing.com is the main data source from which investpy retrieves the data. Investing.com is a global financial portal and Internet brand owned by Fusion Media Ltd. which provides news, analysis, streaming quotes, charts, technical data and financial tools about the global financial markets.

---

## 1.2 Getting Started

TODO

CHAPTER

TWO

# INSTALLATION

To get this package working you will need to **install it via pip** (with a Python 3.6 version or higher) on the terminal by typing:

```
$ pip install investpy
```

Additionally, **if you want to use the latest investpy version instead of the stable one, you can install it from source** with the following command:

```
$ pip install git+https://github.com/alvarobartt/investpy.git@master
```

**The master branch ensures the user that the most updated version will always be working and fully operative** so as not to wait until the the stable release comes out (which eventually may take some time depending on the number of issues to solve).

CHAPTER

THREE

# USAGE

Even though some investpy usage examples are presented on the docs, some basic functionality will be sorted out with sample Python code blocks. Additionally, more usage examples can be found under examples/ directory, which contains a collection of Jupyter Notebooks on how to use investpy and handle its data.

**Note that `investpy.search_quotes` is the only function that ensures that the data is updated and aligned 1:1 with the data provided by Investing.com!**

## 3.1 Recent/Historical Data Retrieval

investpy allows the user to **download both recent and historical data from any financial product indexed** (stocks, funds, ETFs, currency crosses, certificates, bonds, commodities, indices, and cryptos). In the example presented below, the historical data from the past years of a stock is retrieved.

```
import investpy

df = investpy.get_stock_historical_data(stock='AAPL',
                                        country='United States',
                                        from_date='01/01/2010',
                                        to_date='01/01/2020')
print(df.head())
```

```
              Open   High    Low  Close     Volume Currency
Date
2010-01-04  30.49  30.64  30.34  30.57  123432176      USD
2010-01-05  30.66  30.80  30.46  30.63  150476160      USD
2010-01-06  30.63  30.75  30.11  30.14  138039728      USD
2010-01-07  30.25  30.29  29.86  30.08  119282440      USD
2010-01-08  30.04  30.29  29.87  30.28  111969192      USD
```

To get to know all the available recent and historical data extraction functions provided by investpy, and also, parameter tuning, please read the docs.

## 3.2 Search Live Data

**Investing.com search engine is completely integrated** with investpy, which means that any available financial product (quote) can be easily found. The search function allows the user to tune the parameters to adjust the search results to their needs, where both product types and countries from where the products are, can be specified. **All the search functionality can be easily used**, for example, as presented in the following piece of code:

```
import investpy

search_result = investpy.search_quotes(text='apple', products=['stocks'],
                                       countries=['united states'], n_results=1)
print(search_result)
```

```
{"id_": 6408, "name": "Apple Inc", "symbol": "AAPL", "country": "united states", "tag
↪": "/equities/apple-computer-inc", "pair_type": "stocks", "exchange": "NASDAQ"}
```

Retrieved search results will be a `list` of `investpy.utils.search_obj.SearchObj` class instances, unless `n_results` is set to 1, when just a single `investpy.utils.search_obj.SearchObj` class instance will be returned. To get to know which are the available functions and attributes of the returned search results, please read the related documentation at Search Engine Documentation. So on, those search results let the user retrieve both recent and historical data, its information, the technical indicators, the default currency, etc., as presented in the pieces of code below:

### 3.2.1 Recent Data

```
recent_data = search_result.retrieve_recent_data()
print(recent_data.head())
```

```
              Open    High     Low   Close     Volume  Change Pct
Date
2021-05-13  124.58  126.15  124.26  124.97  105861000        1.79
2021-05-14  126.25  127.89  125.85  127.45   81918000        1.98
2021-05-17  126.82  126.93  125.17  126.27   74245000       -0.93
2021-05-18  126.56  126.99  124.78  124.85   63343000       -1.12
2021-05-19  123.16  124.92  122.86  124.69   92612000       -0.13
```

### 3.2.2 Historical Data

```
historical_data = search_result.retrieve_historical_data(from_date='01/01/2019', to_
↪date='01/01/2020')
print(historical_data.head())
```

```
             Open   High    Low  Close     Volume  Change Pct
Date
2020-01-02  74.06  75.15  73.80  75.09  135647008        2.28
2020-01-03  74.29  75.14  74.13  74.36  146536000       -0.97
2020-01-06  73.45  74.99  73.19  74.95  118579000        0.80
2020-01-07  74.96  75.22  74.37  74.60  111511000       -0.47
2020-01-08  74.29  76.11  74.29  75.80  132364000        1.61
```

### 3.2.3 Information

```
information = search_result.retrieve_information()
print(information)
```

```
{"prevClose": 126.11, "dailyRange": "126.1-127.44", "revenue": 325410000000, "open":
↪126.53, "weekRange": "83.14-145.09", "eps": 4.46, "volume": 53522373, "marketCap":
↪2130000000000, "dividend": "0.88(0.70%)", "avgVolume": 88858729, "ratio": 28.58,
↪"beta": 1.2, "oneYearReturn": "50.35%", "sharesOutstanding": 16687631000,
↪"nextEarningDate": "03/08/2021"}
```

### 3.2.4 Currency

```
default_currency = search_result.retrieve_currency()
print(default_currency)
```

```
'USD'
```

### 3.2.5 Technical Indicators

```
technical_indicators = search_result.retrieve_technical_indicators(interval="daily")
print(technical_indicators)
```

```
              indicator           signal     value
0               RSI(14)          neutral   52.1610
1            STOCH(9,6)              buy   63.7110
2          STOCHRSI(14)       overbought  100.0000
3           MACD(12,26)             sell   -0.6700
4               ADX(14)          neutral   21.4750
5          Williams %R              buy  -20.9430
6               CCI(14)              buy   67.1057
7               ATR(14)  less_volatility    1.7871
8       Highs/Lows(14)              buy    0.4279
9   Ultimate Oscillator             sell   47.3620
10                  ROC              buy    1.5150
11  Bull/Bear Power(13)              buy    1.3580
```

## 3.3 Crypto Currencies Data Retrieval

Cryptocurrencies support has recently been included, to let the user retrieve data and information from any available crypto at Investing.com. Please note that some cryptocurrencies do not have available data indexed at Investing.com so that it can not be retrieved using investpy either, even though they are just a few, consider it.

As already presented previously, **historical data retrieval using investpy is really easy**. The piece of code presented below shows how to retrieve the past years of historical data from Bitcoin (BTC).

```
import investpy

data = investpy.get_crypto_historical_data(crypto='bitcoin',
```

(continues on next page)

(continued from previous page)

```
                                        from_date='01/01/2014',
                                        to_date='01/01/2019')

print(data.head())
```

```
             Open    High    Low   Close  Volume Currency
Date
2014-01-01  805.9   829.9  771.0   815.9   10757      USD
2014-01-02  815.9   886.2  810.5   856.9   12812      USD
2014-01-03  856.9   888.2  839.4   884.3    9709      USD
2014-01-04  884.3   932.2  848.3   924.7   14239      USD
2014-01-05  924.7  1029.9  911.4  1014.7   21374      USD
```

CHAPTER

FOUR

# RELATED PROJECTS

Since investpy is intended to retrieve data from different financial products as indexed in Investing.com, the **development of some support modules which implement an additional functionality based on investpy data**, is presented. Note that **anyone can contribute to this section** by creating any package, module, or utility that uses investpy. So on, the ones already created are going to be presented, since they are intended to be used combined with investpy:

- pyrtfolio: is a Python package to generate stock portfolios.
- trendet: is a Python package for trend detection on stock time-series data.

If you developed an interesting/useful project based on investpy data, please open an issue to let me know to include it in this section.

CHAPTER

FIVE

# CONTACT INFORMATION

You can contact me at any of my social network profiles:

- LinkedIn: https://linkedin.com/in/alvarobartt
- Twitter: https://twitter.com/alvarobartt
- GitHub: https://github.com/alvarobartt

Or via email at alvarobartt@yahoo.com, even though this last one is not recommended as mentioned in the FAQs.

CHAPTER

SIX

# CITATION

When citing this repository on your scientific publications please use the following **BibTeX** citation:

```
@misc{investpy,
    author = {Alvaro Bartolome del Canto},
    title = {investpy - Financial Data Extraction from Investing.com with Python},
    year = {2018-2021},
    publisher = {GitHub},
    journal = {GitHub Repository},
    howpublished = {\url{https://github.com/alvarobartt/investpy}},
}
```

When citing this repository on any other social media, please use the following citation:

```
investpy - Financial Data Extraction from Investing.com with Python developed
by Alvaro Bartolome del Canto
```

You should also mention the source from where the data is retrieved, Investing.com; even though it's already included in the package short description title.

CHAPTER

SEVEN

# DISCUSSIONS (Q&A, AMA)

GitHub recently released a new feature named **GitHub Discussions** (still in beta). GitHub Discussions is a collaborative communication forum for the community around an open source project.

Check the investpy GitHub Discussions page at Discussions, and feel free to ask me (ar any developer) anything, share updates, have open-ended conversations, and follow along on decisions affecting the community's way of working.

**Note**. Usually I don't answer emails asking me questions about investpy, as we currently have the GitHub Discussions tab, and I encourage you to use it. GitHub Discussions is the easiest way to contact me about investpy, so that I don't answer the same stuff more than once via email, as anyone can see the opened/answered discussions.

Also, in this section some Frequent Asked Questions are answered, so please read this section before posting a question or openning an issue since duplicates will not be solved or will be referenced to this section. Also, if you think that there are more possible FAQs, consider openning an issue in GitHub so to notify it, since if we all contribute this section can be clear enough so to ease question answering.

**Where can I find the reference of a function and its usage?**

Currently the `docs/` are still missing a lot of information, but they can be clear enough so that users can get to know which functions can be used and how. If you feel that any functionallity or feature is not clear enough, please let me know in the issues tab, so that I can explain it properly for newcomers, so that answers are more general and help more users than just the one asking it. Docs can be found at: Documentation

**What do I do if the financial product I am looking for is not indexed in investpy?**

As it is known, investpy gathers and retrieves data from Investing.com which is a website that contains a lot of financial information. Since investpy relies on Investing.com data, some of it may not be available in Investing, which will mean that it will not be available in investpy either. Anyways, it can be an investpy problem while retrieving data, so on, there is a search function (`investpy.search_quotes(text, products, countries, n_results)`) that can be used for searching financial products that are available in Investing.com but they can not be retrieved using investpy main functions.

**I am having problems while installing the package.**

If you followed the Installation Guide, you should be able to use investpy without having any problem, anyways, if you are stuck on it, open an issue at investpy issues tab so to let the developers know which is your problem in order to solve it as soon as possible. If you were not able to complete the installation, please check that you are running at least Python 3.6 and that you are installing the latest version available, if you are still having problems, open an issue.

**How do I contribute to investpy?**

As this is an open-source project it is **open to contributions, bug reports, bug fixes, documentation improvements, enhancements, and ideas**. There is an open tab of issues where anyone can open new issues if needed or navigate through them to solve them or contribute to its solving. Remember that issues are not threads to describe multiple problems, this does not mean that issues can not be discussed, but so to keep structured project management, the same issue should not describe different problems, just the main one and some nested/related errors that may be found.

CHAPTER

EIGHT

# DISCLAIMER

This Python package has been made for **research purposes** to fit the needs that Investing.com does not cover, so this package works like an Application Programming Interface (API) of Investing.com developed in an **altruistic way**.

Conclude that **investpy is not affiliated in any way to Investing.com or any dependant company**, the only requirement specified by Investing.com to develop this package was to "mention the source where data is retrieved from".

CHAPTER

NINE

# API REFERENCE

## 9.1 investpy.stocks

investpy.stocks.**get_stock_company_profile**(*stock*, *country='spain'*, *language='english'*)

This function retrieves the company profile of a stock company in the specified language. This function is really useful if NLP techniques want to be applied to stocks, since the company profile is a short description of what the company does and since it is written by the company, it can give the user an overview on what does the company do. The company profile can be retrieved either in english or in spanish, the only thing that changes is the source from where the data is retrieved, but the resulting object will be the same. Note that this functionalliy as described in the docs is just supported for spanish stocks currently, so on, if any other stock from any other country is introduced as parameter, the function will raise an exception.

**Note:** Currently just the spanish company profile can be retrieved from spanish stocks, so if you try to retrieve it in spanish for any other country, this function will raise a ValueError exception.

**Parameters**

- **stock** (str) – symbol of the stock to retrieve its company profile from.
- **country** (str) – name of the country from where the stock is.
- **language** (str, optional) – language in which the company profile is going to be retrieved, can either be english or spanish.

**Returns**

The resulting dict contains the retrieved company profile from the selected source depending on the specified language in the function parameters, which can be either Investing.com (english) or Bolsa de Madrid (spanish); and the URL from where it was retrieved, so to have both the source and the description of the company_profile.

So the resulting dict should look like:

```
company_profile = {
    url: 'https://www.investing.com/equities/bbva-company-profile',
    desc: 'Banco Bilbao Vizcaya Argentaria, S.A. (BBVA) is a ...'
}
```

**Return type** dict - company_profile

**Raises**

- **ValueError** – raised whenever any of the introduced arguments is not valid or errored.
- **FileNotFound** – raised if the *stocks.csv* file was not found or unable to retrieve.

- **IOError** – raised if stocks object/file was not found or unable to retrieve.
- **RuntimeError** – raised if the introduced stock/country was not found or did not match any of the existing ones.
- **ConnectionError** – raised if connection to Investing.com could not be established.

### Examples

```
>>> company_profile = investpy.get_stock_company_profile(stock='bbva', country=
↪'spain', language='english')
>>> company_profile
company_profile = {
    url: 'https://www.investing.com/equities/bbva-company-profile',
    desc: 'Banco Bilbao Vizcaya Argentaria, S.A. (BBVA) is a ...'
}
```

investpy.stocks.**get_stock_countries**()

This function returns a listing with all the available countries from where stocks can be retrieved, so to let the user know which of them are available, since the parameter country is mandatory in every stock retrieval function.

**Returns** The resulting `list` contains all the available countries with stocks as indexed in Investing.com

**Return type** `list` - countries

investpy.stocks.**get_stock_dividends**(*stock*, *country*)

This function retrieves the stock dividends from the introduced stocks, which are token rewards paid to the shareholders for their investment in a company's stock/equity. Dividends data include date of the dividend, dividend value, type, payment date and yield. This information is really useful when it comes to creating portfolios.

**Parameters**

- **stock** (`str`) – symbol of the stock to retrieve its dividends from.
- **country** (`country`) – name of the country from where the stock is from.

**Returns**

Returns a `pandas.DataFrame` containing the retrieved information of stock dividends for every stock symbol introduced as parameter.

So on, the resulting `pandas.DataFrame` will look like:

| | Date | Dividend | Type | Payment Date | Yield |
|---|---|---|---|---|---|
| 0 | 2019-10-11 | 0.2600 | trailing_twelve_months | 2019-10-15 | 5,67% |
| 1 | 2019-04-08 | 0.2600 | trailing_twelve_months | 2019-04-10 | 5,53% |
| 2 | 2018-06-11 | 0.3839 | trailing_twelve_months | 2018-06-13 | 3,96% |
| 3 | 2018-04-06 | 0.2400 | trailing_twelve_months | 2018-04-10 | 4,41% |
| 4 | 2017-10-06 | 0.3786 | trailing_twelve_months | 2017-10-10 | 4,45% |

**Return type** `pandas.DataFrame` - stock_dividends

investpy.stocks.**get_stock_financial_summary**(*stock*, *country*, *summary_type='income_statement'*, *period='annual'*)

This function retrieves the financial summary of the introduced stock (by symbol) from the introduced country, based on the summary_type value this function returns a different type of financial summary, so that the output

format of this function depends on its type. Additionally, the period of the retrieved financial summary type can be specified.

**Parameters**

- **stock** (str) – symbol of the stock to retrieve its financial summary.
- **country** (str) – name of the country from where the introduced stock symbol is.
- **summary_type** (str, optional) – type of the financial summary table to retrieve, default value is *income_statement*, but all the available types are: *income_statement*, *cash_flow_statement* and *balance_sheet*.
- **period** (str, optional) – period range of the financial summary table to rertieve, detault value is *annual*, but all the available periods are: *annual* and *quarterly*.

**Returns**

The resulting pandas.DataFrame contains the table of the requested financial summary from the introduced stock, so the fields/column names may vary, since it depends on the summary_type introduced. So on, the returned table will have the following format/structure:

```
Date || Field 1 | Field 2 | ... | Field N
-----||---------|---------|-----|---------
xxxx || xxxxxxx | xxxxxxx | xxx | xxxxxxx
```

**Return type** pandas.DataFrame - financial_summary

**Raises**

- **ValueError** – raised if any of the introduced parameters is not valid or errored.
- **FileNotFoundError** – raised if the stocks.csv file was not found.
- **IOError** – raised if the stocks.csv file could not be read.
- **ConnectionError** – raised if the connection to Investing.com errored or could not be established.
- **RuntimeError** – raised if any error occurred while running the function.

### Examples

```
>>> data = investpy.get_stock_financial_summary(stock='AAPL', country='United␣
↪States', summary_type='income_statement', period='annual')
>>> data.head()
            Total Revenue  Gross Profit  Operating Income  Net Income
Date
2019-09-28         260174         98392             63930       55256
2018-09-29         265595        101839             70898       59531
2017-09-30         229234         88186             61344       48351
2016-09-24         215639         84263             60024       45687
```

investpy.stocks.**get_stock_historical_data**(*stock*, *country*, *from_date*, *to_date*, *as_json=False*, *order='ascending'*, *interval='Daily'*)

This function retrieves historical data from the introduced stock from Investing.com. So on, the historical data of the introduced stock from the specified country in the specified date range will be retrieved and returned as a pandas.DataFrame if the parameters are valid and the request to Investing.com succeeds. Note that additionally some optional parameters can be specified: as_json and order, which let the user decide if the data

is going to be returned as a `json` or not, and if the historical data is going to be ordered ascending or descending (where the index is the date), respectively.

**Parameters**

- **stock** (`str`) – symbol of the stock to retrieve historical data from.
- **country** (`str`) – name of the country from where the stock is.
- **from_date** (`str`) – date formatted as *dd/mm/yyyy*, since when data is going to be retrieved.
- **to_date** (`str`) – date formatted as *dd/mm/yyyy*, until when data is going to be retrieved.
- **as_json** (`bool`, optional) – to determine the format of the output data, either a `pandas.DataFrame` if False and a `json` if True.
- **order** (`str`, optional) – to define the order of the retrieved data which can either be ascending or descending.
- **interval** (`str`, optional) – value to define the historical data interval to retrieve, by default *Daily*, but it can also be *Weekly* or *Monthly*.

**Returns**

The function can return either a `pandas.DataFrame` or a `json` object, containing the retrieved historical data of the specified stock from the specified country. So on, the resulting dataframe contains the open, high, low, close and volume values for the selected stock on market days and the currency in which those values are presented.

The returned data is case we use default arguments will look like:

```
Date || Open | High | Low | Close | Volume | Currency
-----||------|------|-----|-------|--------|----------
xxxx || xxxx | xxxx | xxx | xxxxx | xxxxxx | xxxxxxxx
```

but if we define *as_json=True*, then the output will be:

```
{
    name: name,
    historical: [
        {
            date: 'dd/mm/yyyy',
            open: x,
            high: x,
            low: x,
            close: x,
            volume: x,
            currency: x
        },
        ...
    ]
}
```

**Return type** `pandas.DataFrame` or `json`

**Raises**

- **ValueError** – raised whenever any of the introduced arguments is not valid or errored.
- **IOError** – raised if stocks object/file was not found or unable to retrieve.

- **RuntimeError** – raised if the introduced stock/country was not found or did not match any of the existing ones.
- **ConnectionError** – raised if connection to Investing.com could not be established.
- **IndexError** – raised if stock historical data was unavailable or not found in Investing.com.

### Examples

```
>>> data = investpy.get_stock_historical_data(stock='bbva', country='spain', from_
↪date='01/01/2010', to_date='01/01/2019')
>>> data.head()
             Open   High    Low  Close  Volume Currency
Date
2010-01-04  12.73  12.96  12.73  12.96       0      EUR
2010-01-05  13.00  13.11  12.97  13.09       0      EUR
2010-01-06  13.03  13.17  13.02  13.12       0      EUR
2010-01-07  13.02  13.11  12.93  13.05       0      EUR
2010-01-08  13.12  13.22  13.04  13.18       0      EUR
```

investpy.stocks.**get_stock_information**(*stock*, *country*, *as_json=False*)

This function retrieves fundamental financial information from the specified stock. The retrieved information from the stock can be valuable as it is additional information that can be used combined with OHLC values, so to determine financial insights from the company which holds the specified stock.

**Parameters**

- **stock** (`str`) – symbol of the stock to retrieve its information from.
- **country** (`country`) – name of the country from where the stock is from.
- **as_json** (`bool`, optional) – optional argument to determine the format of the output data (`dict` or `json`).

**Returns**

The resulting `pandas.DataFrame` contains the information fields retrieved from Investing.com from the specified stock ; it can also be returned as a `dict`, if argument *as_json=True*.

If any of the information fields could not be retrieved, that field/s will be filled with None values. If the retrieval process succeeded, the resulting `dict` will look like:

```
stock_information = {
    "Stock Symbol": "AAPL",
    "Prev. Close": 267.25,
    "Todays Range": "263.45 - 268.25",
    "Revenue": 260170000000.00003,
    "Open": 267.27,
    "52 wk Range": "142 - 268.25",
    "EPS": 11.85,
    "Volume": 23693550.0,
    "Market Cap": 1173730000000.0,
    "Dividend (Yield)": "3.08 (1.15%)",
    "Average Vol. (3m)": 25609925.0,
    "P/E Ratio": 22.29,
    "Beta": 1.23,
    "1-Year Change": "47.92%",
    "Shares Outstanding": 4443236000.0,
```

(continues on next page)

(continued from previous page)

```
    "Next Earnings Date": "04/02/2020"
}
```

**Return type** pandas.DataFrame or dict- stock_information

**Raises**

- **ValueError** – raised if any of the introduced arguments is not valid or errored.
- **FileNotFoundError** – raised if *stocks.csv* file was not found or errored.
- **IOError** – raised if *stocks.csv* file is empty or errored.
- **RuntimeError** – raised if scraping process failed while running.
- **ConnectionError** – raised if the connection to Investing.com errored (did not return HTTP 200)

investpy.stocks.**get_stock_recent_data**(*stock*, *country*, *as_json=False*, *order='ascending'*, *interval='Daily'*)

This function retrieves recent historical data from the introduced stock from Investing.com. So on, the recent data of the introduced stock from the specified country will be retrieved and returned as a pandas.DataFrame if the parameters are valid and the request to Investing.com succeeds. Note that additionally some optional parameters can be specified: as_json and order, which let the user decide if the data is going to be returned as a json or not, and if the historical data is going to be ordered ascending or descending (where the index is the date), respectively.

**Parameters**

- **stock** (str) – symbol of the stock to retrieve recent historical data from.
- **country** (str) – name of the country from where the stock is.
- **as_json** (bool, optional) – to determine the format of the output data, either a pandas.DataFrame if False and a json if True.
- **order** (str, optional) – to define the order of the retrieved data which can either be ascending or descending.
- **interval** (str, optional) – value to define the historical data interval to retrieve, by default *Daily*, but it can also be *Weekly* or *Monthly*.

**Returns**

The function can return either a pandas.DataFrame or a json object, containing the retrieved recent data of the specified stock from the specified country. So on, the resulting dataframe contains the open, high, low, close and volume values for the selected stock on market days and the currency in which those values are presented.

The resulting recent data, in case that the default parameters were applied, will look like:

```
Date || Open | High | Low | Close | Volume | Currency
-----||------|------|-----|-------|--------|----------
xxxx || xxxx | xxxx | xxx | xxxxx | xxxxxx | xxxxxxxx
```

but in case that as_json parameter was defined as True, then the output will be:

```
{
    name: name,
    recent: [
        {
```

(continues on next page)

(continued from previous page)

```
            date: 'dd/mm/yyyy',
            open: x,
            high: x,
            low: x,
            close: x,
            volume: x,
            currency: x
        },
        ...
    ]
}
```

**Return type** `pandas.DataFrame` or `json`

**Raises**

- **`ValueError`** – raised whenever any of the introduced arguments is not valid or errored.
- **`IOError`** – raised if stocks object/file was not found or unable to retrieve.
- **`RuntimeError`** – raised if the introduced stock/country was not found or did not match any of the existing ones.
- **`ConnectionError`** – raised if connection to Investing.com could not be established.
- **`IndexError`** – raised if stock recent data was unavailable or not found in Investing.com.

### Examples

```
>>> data = investpy.get_stock_recent_data(stock='bbva', country='spain')
>>> data.head()
             Open   High    Low  Close    Volume Currency
Date
2019-08-13  4.263  4.395  4.230  4.353  27250000      EUR
2019-08-14  4.322  4.325  4.215  4.244  36890000      EUR
2019-08-15  4.281  4.298  4.187  4.234  21340000      EUR
2019-08-16  4.234  4.375  4.208  4.365  46080000      EUR
2019-08-19  4.396  4.425  4.269  4.269  18950000      EUR
```

`investpy.stocks.`**`get_stocks`**(*country=None*)

This function retrieves all the stock data stored in *stocks.csv* file, which previously was retrieved from Investing.com. Since the resulting object is a matrix of data, the stock data is properly structured in rows and columns, where columns are the stock data attribute names. Additionally, country filtering can be specified, which will make this function return not all the stored stock data, but just the stock data of the stocks from the introduced country.

**Parameters** **`country`** (`str`, optional) – name of the country to retrieve all its available stocks from.

**Returns**

The resulting `pandas.DataFrame` contains all the stock data from the introduced country if specified, or from every country if None was specified, as indexed in Investing.com from the information previously retrieved by investpy and stored on a csv file.

So on, the resulting `pandas.DataFrame` will look like:

```
country | name | full name | isin | currency | symbol
--------|------|-----------|------|----------|--------
xxxxxxx | xxxx | xxxxxxxxx | xxxx | xxxxxxxx | xxxxxx
```

**Return type** pandas.DataFrame - stocks_df

**Raises**

- **ValueError** – raised whenever any of the introduced arguments is not valid.
- **FileNotFoundError** – raised if *stocks.csv* file was not found.
- **IOError** – raised when *stocks.csv* file is missing or empty.

investpy.stocks.**get_stocks_dict**(*country=None*, *columns=None*, *as_json=False*)

This function retrieves all the stock information stored in the *stocks.csv* file and formats it as a Python dictionary which contains the same information as the file, but every row is a dict and all of them are contained in a list. Note that the dictionary structure is the same one as the JSON structure. Some optional paramaters can be specified such as the country, columns or as_json, which are a filtering by country so not to return all the stocks but just the ones from the introduced country, the column names that want to be retrieved in case of needing just some columns to avoid unnecessary information load, and whether the information wants to be returned as a JSON object or as a dictionary; respectively.

**Parameters**

- **country** (str, optional) – name of the country to retrieve all its available stocks from.
- **columns** (list, optional) – column names of the stock data to retrieve, can be: <country, name, full_name, isin, currency, symbol>
- **as_json** (bool, optional) – if True the returned data will be a json object, if False, a list of dict.

**Returns**

The resulting list of dict contains the retrieved data from every stock as indexed in Investing.com from the information previously retrieved by investpy and stored on a csv file.

In case the information was successfully retrieved, the list of dict will look like:

```
stocks_dict = {
    'country': country,
    'name': name,
    'full_name': full_name,
    'isin': isin,
    'currency': currency,
    'symbol': symbol,
}
```

**Return type** list of dict OR json - stocks_dict

**Raises**

- **ValueError** – raised whenever any of the introduced arguments is not valid.
- **FileNotFoundError** – raised if *stocks.csv* file was not found.
- **IOError** – raised when *stocks.csv* file is missing or empty.

investpy.stocks.**get_stocks_list**(*country=None*)

This function retrieves all the stock symbols stored in *stocks.csv* file, which contains all the data from the stocks as previously retrieved from Investing.com. So on, this function will just return the stock symbols which will be one of the input parameters when it comes to stock data retrieval functions from investpy. Additionally, note

that the country filtering can be applied, which is really useful since this function just returns the symbols and in stock data retrieval functions both the symbol and the country must be specified and they must match.

**Parameters** **country** (str, optional) – name of the country to retrieve all its available stocks from.

**Returns**

The resulting list contains the all the stock symbols from the introduced country if specified, or from every country if None was specified, as indexed in Investing.com from the information previously retrieved by investpy and stored on a csv file.

In case the information was successfully retrieved, the list of stock symbols will look like:

```
stocks_list = ['TS', 'APBR', 'GGAL', 'TXAR', 'PAMP', ...]
```

**Return type** list - stocks_list

**Raises**

- **ValueError** – raised whenever any of the introduced arguments is not valid.
- **FileNotFoundError** – raised if *stocks.csv* file was not found.
- **IOError** – raised when *stocks.csv* file is missing or empty.

investpy.stocks.**get_stocks_overview**(*country*, *as_json=False*, *n_results=100*)

This function retrieves an overview containing all the real time data available for the main stocks from a country, such as the names, symbols, current value, etc. as indexed in Investing.com. So on, the main usage of this function is to get an overview on the main stocks from a country, so to get a general view. Note that since this function is retrieving a lot of information at once, by default just the overview of the Top 100 stocks is being retrieved, but an additional parameter called n_results can be specified so to retrieve N results.

**Parameters**

- **country** (str) – name of the country to retrieve the stocks overview from.
- **as_json** (bool, optional) – optional argument to determine the format of the output data (pandas.DataFrame or json).
- **n_results** (int, optional) – number of results to be displayed on the overview table (0-1000).

**Returns**

The resulting pandas.DataFrame contains all the data available in Investing.com of the main stocks from a country in order to get an overview of it.

If the retrieval process succeeded, the resulting pandas.DataFrame should look like:

```
country | name | symbol | last | high | low | change | change_
↪percentage | turnover | currency
--------|------|--------|------|------|-----|--------|----------------
↪---|----------|----------
xxxxxxx | xxxx | xxxxxx | xxxx | xxxx | xxx | xxxxxx |␣
↪xxxxxxxxxxxxxxxx | xxxxxxxx | xxxxxxxx
```

**Return type** pandas.DataFrame - stocks_overview

**Raises**

- **ValueError** – raised if any of the introduced arguments errored.
- **FileNotFoundError** – raised when *stocks.csv* file is missing.

- **IOError** – raised if data could not be retrieved due to file error.
- **RuntimeError** – raised either if the introduced country does not match any of the listed ones or if no overview results could be retrieved from Investing.com.
- **ConnectionError** – raised if GET requests does not return 200 status code.

investpy.stocks.**search_stocks**(*by*, *value*)

This function searches stocks by the introduced value for the specified field. This means that this function is going to search if there is a value that matches the introduced one for the specified field which is the *stocks.csv* column name to search in. Available fields to search stocks are 'name', 'full_name' and 'isin'.

**Parameters**

- **by** (str) – name of the field to search for, which is the column name which can be: 'name', 'full_name' or 'isin'.
- **value** (str) – value of the field to search for, which is the value that is going to be searched.

**Returns** The resulting pandas.DataFrame contains the search results from the given query, which is any match of the specified value in the specified field. If there are no results for the given query, an error will be raised, but otherwise the resulting pandas.DataFrame will contain all the available stocks that match the introduced query.

**Return type** pandas.DataFrame - search_result

**Raises**

- **ValueError** – raised if any of the introduced parameters is not valid or errored.
- **FileNotFoundError** – raised if *stocks.csv* file is missing.
- **IOError** – raised if data could not be retrieved due to file error.
- **RuntimeError** – raised if no results were found for the introduced value in the introduced field.

## 9.2 investpy.funds

investpy.funds.**get_fund_countries**()

This function retrieves all the country names indexed in Investing.com with available funds to retrieve data from, via reading the *fund_countries.csv* file from the resources directory. So on, this function will display a listing containing a set of countries, in order to let the user know which countries are taken into account and also the return listing from this function can be used for country param check if needed.

**Returns** The resulting list contains all the available countries with funds as indexed in Investing.com

**Return type** list - countries

**Raises**

- **FileNotFoundError** – raised when the *fund_countries.csv* file was not found.
- **IndexError** – raised if *fund_countries.csv* file was unavailable or not found.

investpy.funds.**get_fund_historical_data**(*fund*, *country*, *from_date*, *to_date*, *as_json=False*, *order='ascending'*, *interval='Daily'*)

This function retrieves historical data from the introduced *fund* from Investing via Web Scraping on the introduced date range. The resulting data can it either be stored in a pandas.DataFrame or in a json object with *ascending* or *descending* order.

**Parameters**

- **fund** (str) – name of the fund to retrieve recent historical data from.
- **country** (str) – name of the country from where the introduced fund is.
- **from_date** (str) – date as *str* formatted as *dd/mm/yyyy*, from where data is going to be retrieved.
- **to_date** (str) – date as *str* formatted as *dd/mm/yyyy*, until where data is going to be retrieved.
- **as_json** (bool, optional) – to determine the format of the output data (pandas.DataFrame or json).
- **order** (str, optional) – optional argument to define the order of the retrieved data (*ascending*, *asc* or *descending*, *desc*).
- **interval** (str, optional) – value to define the historical data interval to retrieve, by default *Daily*, but it can also be *Weekly* or *Monthly*.

**Returns**

The function returns a either a pandas.DataFrame or a json file containing the retrieved recent data from the specified fund via argument. The dataset contains the open, high, low and close values for the selected fund on market days.

The returned data is case we use default arguments will look like:

```
Date || Open | High | Low | Close | Currency
-----||------|------|-----|-------|----------
xxxx || xxxx | xxxx | xxx | xxxxx | xxxxxxxx
```

but if we define *as_json=True*, then the output will be:

```
{
    name: name,
    historical: [
        {
            date: dd/mm/yyyy,
            open: x,
            high: x,
            low: x,
            close: x,
            currency: x
        },
        ...
    ]
}
```

**Return type** pandas.DataFrame or json

**Raises**

- **ValueError** – argument error.
- **IOError** – funds object/file not found or unable to retrieve.
- **RuntimeError** – introduced fund does not match any of the indexed ones.
- **ConnectionError** – if GET requests does not return 200 status code.
- **IndexError** – if fund information was unavailable or not found.

### Examples

```
>>> data = investpy.get_fund_historical_data(fund='bbva multiactivo conservador pp
↪', country='spain', from_date='01/01/2010', to_date='01/01/2019')
>>> data.head()
             Open   High    Low  Close Currency
Date
2018-02-15  1.105  1.105  1.105  1.105      EUR
2018-02-16  1.113  1.113  1.113  1.113      EUR
2018-02-17  1.113  1.113  1.113  1.113      EUR
2018-02-18  1.113  1.113  1.113  1.113      EUR
2018-02-19  1.111  1.111  1.111  1.111      EUR
```

investpy.funds.**get_fund_information**(*fund*, *country*, *as_json=False*)

This function retrieves basic financial information from the specified fund. Retrieved information from the fund can be valuable as it is additional information that can be used combined with OHLC values, so to determine financial insights from the company which holds the specified fund.

**Parameters**

- **fund** (str) – name of the fund to retrieve the financial information from.
- **country** (str) – name of the country from where the introduced fund is.
- **as_json** (bool, optional) – optional argument to determine the format of the output data (dict or json).

**Returns**

The resulting pandas.DataFrame contains the information fields retrieved from Investing.com from the specified fund; it can also be returned as a dict, if argument *as_json=True*.

If any of the information fields could not be retrieved, that field/s will be filled with None values. If the retrieval process succeeded, the resulting dict will look like:

```
fund_information = {
    'Fund Name': fund_name,
    'Rating': rating,
    '1-Year Change': year_change,
    'Previous Close': prev_close,
    'Risk Rating': risk_rating,
    'TTM Yield': ttm_yield,
    'ROE': roe,
    'Issuer': issuer,
    'Turnover': turnover,
    'ROA': row,
    'Inception Date': inception_date,
    'Total Assets': total_assets,
    'Expenses': expenses,
    'Min Investment': min_investment,
    'Market Cap': market_cap,
    'Category': category
}
```

**Return type** pandas.DataFrame or dict- fund_information

investpy.funds.**get_fund_recent_data**(*fund*, *country*, *as_json=False*, *order='ascending'*, *interval='Daily'*)

This function retrieves recent historical data from the introduced *fund* from Investing via Web Scraping. The

resulting data can it either be stored in a `pandas.DataFrame` or in a `json` file, with *ascending* or *descending* order.

**Parameters**

- **fund** (`str`) – name of the fund to retrieve recent historical data from.
- **country** (`str`) – name of the country from where the introduced fund is.
- **as_json** (`bool`, optional) – optional argument to determine the format of the output data (`pandas.DataFrame` or `json`).
- **order** (`str`, optional) – optional argument to define the order of the retrieved data (*ascending*, *asc* or *descending*, *desc*).
- **interval** (`str`, optional) – value to define the historical data interval to retrieve, by default *Daily*, but it can also be *Weekly* or *Monthly*.

**Returns**

The function returns a either a `pandas.DataFrame` or a `json` file containing the retrieved recent data from the specified fund via argument. The dataset contains the open, high, low and close values for the selected fund on market days.

The returned data is case we use default arguments will look like:

```
Date || Open | High | Low | Close | Currency
-----||------|------|-----|-------|----------
xxxx || xxxx | xxxx | xxx | xxxxx | xxxxxxxx
```

but if we define *as_json=True*, then the output will be:

```
{
    name: name,
    recent: [
        {
            date: dd/mm/yyyy,
            open: x,
            high: x,
            low: x,
            close: x,
            currency: x
        },
        ...
    ]
}
```

**Return type** `pandas.DataFrame` or `json`

**Raises**

- **ValueError** – argument error.
- **IOError** – funds object/file not found or unable to retrieve.
- **RuntimeError** – introduced fund does not match any of the indexed ones.
- **ConnectionError** – if GET requests does not return 200 status code.
- **IndexError** – if fund information was unavailable or not found.

### Examples

```
>>> data = investpy.get_fund_recent_data(fund='bbva multiactivo conservador pp',␣
↪country='spain')
>>> data.head()
             Open   High    Low  Close Currency
Date
2019-08-13  1.110  1.110  1.110  1.110      EUR
2019-08-16  1.109  1.109  1.109  1.109      EUR
2019-08-19  1.114  1.114  1.114  1.114      EUR
2019-08-20  1.112  1.112  1.112  1.112      EUR
2019-08-21  1.115  1.115  1.115  1.115      EUR
```

`investpy.funds.get_funds(country=None)`

This function retrieves all the available *funds* from Investing.com and returns them as a `pandas.DataFrame`, which contains not just the fund names, but all the fields contained on the *funds.csv* file. All the available funds can be found at: https://www.investing.com/funds/

**Parameters** **country** (`str`, optional) – name of the country to retrieve all its available funds from.

**Returns**

The resulting `pandas.DataFrame` contains all the funds basic information retrieved from Investing.com, some of which is not useful for the user, but for the inner package functions, such as the *id* field, for example.

In case the information was successfully retrieved, the `pandas.DataFrame` will look like:

```
country | name | symbol | issuer | isin | asset_class | currency |␣
↪underlying
--------|------|--------|--------|------|-------------|----------|----
↪--------
xxxxxxx | xxxx | xxxxxx | xxxxxx | xxxx | xxxxxxxxxxx | xxxxxxxx |␣
↪xxxxxxxxxx
```

**Return type** `pandas.DataFrame` - funds_df

**Raises**

- **ValueError** – raised whenever any of the introduced arguments is not valid or errored.
- **FileNotFoundError** – raised when the *funds.csv* file was not found.
- **IOError** – raised if the *funds.csv* file is missing or errored.

`investpy.funds.get_funds_dict(country=None, columns=None, as_json=False)`

This function retrieves all the available funds on Investing.com and returns them as a `dict` containing the country, name, symbol, tag, id, issuer, isin, asset_class, currency and underlying data. All the available funds can be found at: https://www.investing.com/funds/

**Parameters**

- **country** (`str`, optional) – name of the country to retrieve all its available funds from.
- **columns** (`list` of `str`, optional) – description a `list` containing the column names from which the data is going to be retrieved.
- **as_json** (`bool`, optional) – description value to determine the format of the output data (`dict` or `json`).

**Returns**

The resulting `dict` contains the retrieved data if found, if not, the corresponding fields are filled with *None* values.

In case the information was successfully retrieved, the `dict` will look like:

```
{
    'country': country,
    'name': name,
    'symbol': symbol,
    'issuer': issuer,
    'isin': isin,
    'asset_class': asset_class,
    'currency': currency,
    'underlying': underlying
}
```

**Return type** `dict` or `json` - funds_dict

**Raises**

- **ValueError** – raised whenever any of the introduced arguments is not valid or errored.
- **FileNotFoundError** – raised when the *funds.csv* file was not found.
- **IOError** – raised if the *funds.csv* file is missing or errored.

investpy.funds.**get_funds_list**(*country=None*)

This function retrieves all the available funds and returns a list of each one of them. All the available funds can be found at: https://www.investing.com/funds/

**Parameters** **country** (`str`, optional) – name of the country to retrieve all its available funds from.

**Returns**

The resulting list contains the retrieved data, which corresponds to the fund names of every fund listed on Investing.com.

In case the information was successfully retrieved from the CSV file, the `list` will look like:

```
funds = [
    'Blackrock Global Funds - Global Allocation Fund E2',
    'Quality Inversión Conservadora Fi',
    'Nordea 1 - Stable Return Fund E Eur',
    ...
]
```

**Return type** `list` - funds_list

**Raises**

- **ValueError** – raised whenever any of the introduced arguments is not valid or errored.
- **FileNotFoundError** – raised when the *funds.csv* file was not found.
- **IOError** – raised if the *funds.csv* file is missing or errored.

investpy.funds.**get_funds_overview**(*country*, *as_json=False*, *n_results=100*)

This function retrieves an overview containing all the real time data available for the main funds from a country, such as the names, symbols, current value, etc. as indexed in Investing.com. So on, the main usage of this function is to get an overview on the main funds from a country, so to get a general view. Note that since this function is retrieving a lot of information at once, by default just the overview of the Top 100 funds is being retrieved, but an additional parameter called n_results can be specified so to retrieve N results.

**Parameters**

- **country** (str) – name of the country to retrieve the funds overview from.
- **as_json** (bool, optional) – optional argument to determine the format of the output data (pandas.DataFrame or json).
- **n_results** (int, optional) – number of results to be displayed on the overview table (0-1000).

**Returns**

The resulting pandas.DataFrame contains all the data available in Investing.com of the main ETFs from a country in order to get an overview of it.

If the retrieval process succeeded, the resulting pandas.DataFrame should look like:

```
country | name | symbol | last | change | total_assets
--------|------|--------|------|--------|--------------
xxxxxxx | xxxx | xxxxxx | xxxx | xxxxxx | xxxxxxxxxxxx
```

**Return type** pandas.DataFrame - funds_overview

**Raises**

- **ValueError** – raised if there was any argument error.
- **FileNotFoundError** – raised when *funds.csv* file is missing.
- **IOError** – raised if data could not be retrieved due to file error.
- **RuntimeError** – raised either if the introduced country does not match any of the listed ones or if no overview results could be retrieved from Investing.com.
- **ConnectionError** – raised if GET requests does not return 200 status code.

investpy.funds.**search_funds**(*by*, *value*)

This function searches funds by the introduced value for the specified field. This means that this function is going to search if there is a value that matches the introduced value for the specified field which is the *funds.csv* column name to search in. Available fields to search funds are 'name', 'symbol', 'issuer' and 'isin'.

**Parameters**

- **by** (str) – name of the field to search for, which is the column name ('name', 'symbol', 'issuer' or 'isin').
- **value** (str) – value of the field to search for, which is the str that is going to be searched.

**Returns** The resulting *pandas.DataFrame* contains the search results from the given query (the specified value in the specified field). If there are no results and error will be raised, but otherwise this *pandas.DataFrame* will contain all the available field values that match the introduced query.

**Return type** pandas.DataFrame - search_result

**Raises**

- **ValueError** – raised if any of the introduced params is not valid or errored.
- **FileNotFoundError** – raised if *funds.csv* file is missing.
- **IOError** – raised if data could not be retrieved due to file error.
- **RuntimeError** – raised if no results were found for the introduced value in the introduced field.

## 9.3 investpy.etfs

investpy.etfs.**get_etf_countries**()

This function retrieves all the available countries to retrieve etfs from, as the listed countries are the ones indexed on Investing.com. The purpose of this function is to list the countries which have available etfs according to Investing.com data, so to ease the etf retrieval process of a particular country.

**Returns**

The resulting `list` contains all the countries listed on Investing.com with etfs available to retrieve data from.

In the case that the file reading of *etf_countries.csv* which contains the names and codes of the countries with etfs was successfully completed, the resulting `list` will look like:

```
countries = ['australia', 'austria', 'belgium', 'brazil', ...]
```

**Return type** `list` - countries

**Raises** **FileNotFoundError** – raised when *etf_countries.csv* file was not found.

investpy.etfs.**get_etf_historical_data**(*etf*, *country*, *from_date*, *to_date*, *stock_exchange=None*, *as_json=False*, *order='ascending'*, *interval='Daily'*)

This function retrieves historical data from the introduced *etf* from Investing.com via Web Scraping on the introduced date range. The resulting data can it either be stored in a `pandas.DataFrame` or in a `json` object with *ascending* or *descending* order.

**Parameters**

- **etf** (`str`) – name of the etf to retrieve recent historical data from.
- **country** (`str`) – name of the country from where the etf is.
- **from_date** (`str`) – date as *str* formatted as *dd/mm/yyyy*, from where data is going to be retrieved.
- **to_date** (`str`) – date as *str* formatted as *dd/mm/yyyy*, until where data is going to be retrieved.
- **as_json** (`bool`, optional) – to determine the format of the output data (`pandas.DataFrame` or `json`).
- **order** (`str`, optional) – optional argument to define the order of the retrieved data (*ascending*, *asc* or *descending*, *desc*).
- **interval** (`str`, optional) – value to define the historical data interval to retrieve, by default *Daily*, but it can also be *Weekly* or *Monthly*.

**Returns**

The function returns either a `pandas.DataFrame` or a `json` file containing the retrieved recent data from the specified etf via argument. The dataset contains the open, high, low and close values for the selected etf on market days.

The returned data is case we use default arguments will look like:

```
Date || Open | High | Low | Close | Volume | Currency | Exchange
-----||------|------|-----|-------|--------|----------|---------
xxxx || xxxx | xxxx | xxx | xxxxx | xxxxxx | xxxxxxxx | xxxxxxxx
```

but if we define *as_json=True*, then the output will be:

```
{
    name: name,
    historical: [
        {
            date: dd/mm/yyyy,
            open: x,
            high: x,
            low: x,
            close: x,
            volume: x,
            currency: x,
            exchange: x
        },
        ...
    ]
}
```

**Return type** pandas.DataFrame or json

**Raises**

- **ValueError** – raised whenever any of the arguments is not valid or errored.
- **IOError** – raised if etfs object/file not found or unable to retrieve.
- **RuntimeError** – raised if the introduced etf does not match any of the indexed ones.
- **ConnectionError** – raised if GET requests does not return 200 status code.
- **IndexError** – raised if etf information was unavailable or not found.

### Examples

```
>>> data = investpy.get_etf_historical_data(etf='bbva accion dj eurostoxx 50',␣
↪country='spain', from_date='01/01/2010', to_date='01/01/2019')
>>> data.head()
              Open   High    Low  Close  Volume Currency Exchange
Date
2011-12-07  23.70  23.70  23.70  23.62    2000      EUR   Madrid
2011-12-08  23.53  23.60  23.15  23.04     599      EUR   Madrid
2011-12-09  23.36  23.60  23.36  23.62    2379      EUR   Madrid
2011-12-12  23.15  23.26  23.00  22.88   10695      EUR   Madrid
2011-12-13  22.88  22.88  22.88  22.80      15      EUR   Madrid
```

investpy.etfs.**get_etf_information**(*etf*, *country*, *as_json=False*)

This function retrieves fundamental financial information from the specified ETF. The retrieved information from the ETF can be valuable as it is additional information that can be used combined with OHLC values, so to determine financial insights from the company which holds the specified ETF.

**Parameters**

- **etf** (str) – name of the ETF to retrieve recent historical data from.
- **country** (str) – name of the country from where the ETF is.
- **as_json** (bool, optional) – optional argument to determine the format of the output data (dict or json).

**Returns**

The resulting `pandas.DataFrame` contains the information fields retrieved from Investing.com from the specified ETF; it can also be returned as a `dict`, if argument *as_json=True*.

If any of the information fields could not be retrieved, that field/s will be filled with None values. If the retrieval process succeeded, the resulting `dict` will look like:

```
etf_information = {
    "1-Year Change": "21.83%",
    "52 wk Range": "233.76 - 320.06",
    "Asset Class": "Equity",
    "Average Vol. (3m)": 59658771.0,
    "Beta": 1.01,
    "Dividend Yield": "1.73%",
    "Dividends (TTM)": 4.03,
    "ETF Name": "SPDR S&P 500",
    "Market Cap": 296440000000.0,
    "Open": 319.25,
    "Prev. Close": 317.27,
    "ROI (TTM)": "- 0.46%",
    "Shares Outstanding": 934132116.0,
    "Todays Range": "319.18 - 320.06",
    "Total Assets": 167650000000.0,
    "Volume": 27928710.0
}
```

**Return type** `pandas.DataFrame` or `dict`- etf_information

**Raises**

- **`ValueError`** – raised if any of the introduced arguments is not valid or errored.
- **`FileNotFoundError`** – raised if *etfs.csv* file was not found or errored.
- **`IOError`** – raised if *etfs.csv* file is empty or errored.
- **`RuntimeError`** – raised if scraping process failed while running.
- **`ConnectionError`** – raised if the connection to Investing.com errored (did not return HTTP 200)

`investpy.etfs.`**`get_etf_recent_data`**(*etf*, *country*, *stock_exchange=None*, *as_json=False*, *order='ascending'*, *interval='Daily'*)

This function retrieves recent historical data from the introduced *etf* from Investing via Web Scraping. The resulting data can it either be stored in a `pandas.DataFrame` or in a `json` file, with *ascending* or *descending* order.

**Parameters**

- **`etf`** (`str`) – name of the etf to retrieve recent historical data from.
- **`country`** (`str`) – name of the country from where the etf is.
- **`as_json`** (`bool`, optional) – optional argument to determine the format of the output data (`pandas.DataFrame` or `json`).
- **`order`** (`str`, optional) – optional argument to define the order of the retrieved data (*ascending*, *asc* or *descending*, *desc*).
- **`interval`** (`str`, optional) – value to define the historical data interval to retrieve, by default *Daily*, but it can also be *Weekly* or *Monthly*.

**Returns**

The function returns either a `pandas.DataFrame` or a `json` file containing the retrieved recent data from the specified etf via argument. The dataset contains the open, high, low and close values for the selected etf on market days.

The returned data is case we use default arguments will look like:

```
Date || Open | High | Low | Close | Volume | Currency | Exchange
-----||------|------|-----|-------|--------|----------|---------
xxxx || xxxx | xxxx | xxx | xxxxx | xxxxxx | xxxxxxxx | xxxxxxxx
```

but if we define *as_json=True*, then the output will be:

```
{
    name: name,
    recent: [
        {
            date: dd/mm/yyyy,
            open: x,
            high: x,
            low: x,
            close: x,
            volume: x,
            currency: x,
            exchange: x
        },
        ...
    ]
}
```

**Return type** `pandas.DataFrame` or `json`

**Raises**

- **`ValueError`** – raised whenever any of the arguments is not valid or errored.
- **`IOError`** – raised if etfs object/file not found or unable to retrieve.
- **`RuntimeError`** – raised if the introduced etf does not match any of the indexed ones.
- **`ConnectionError`** – raised if GET requests does not return 200 status code.
- **`IndexError`** – raised if etf information was unavailable or not found.

### Examples

```
>>> data = investpy.get_etf_recent_data(etf='bbva accion dj eurostoxx 50',␣
↪country='spain')
>>> data.head()
              Open    High    Low   Close  Volume Currency Exchange
Date
2020-04-09  28.890  29.155  28.40  28.945   20651      EUR   Madrid
2020-04-14  29.345  30.235  28.94  29.280   14709      EUR   Madrid
2020-04-15  29.125  29.125  28.11  28.130   14344      EUR   Madrid
2020-04-16  28.505  28.590  28.08  28.225   17662      EUR   Madrid
2020-04-17  29.000  29.325  28.80  28.895   19578      EUR   Madrid
```

`investpy.etfs.`**`get_etfs`**(*country=None*)

This function retrieves all the available etfs indexed on Investing.com, already stored on *etfs.csv*. This function

also allows the users to specify which country do they want to retrieve data from or if they want to retrieve it from every listed country; so on, all the indexed etfs will be returned.

**Parameters** **country** (str, optional) – name of the country to retrieve all its available etfs from.

**Returns**

The resulting pandas.DataFrame contains all the etfs basic information stored on *etfs.csv*, since it was previously retrieved by investpy. Unless the country is specified, all the available etfs indexed on Investing.com is returned, but if it is specified, just the etfs from that country are returned.

In the case that the file reading of *etfs.csv* or the retrieval process from Investing.com was successfully completed, the resulting pandas.DataFrame will look like:

```
country | name | full_name | symbol | isin | asset_class | currency |␣
↪stock_exchange | def_stock_exchange
--------|------|-----------|--------|------|-------------|----------|-
↪---------------|--------------------
xxxxxxx | xxxx | xxxxxxxxx | xxxxxx | xxxx | xxxxxxxxxxx | xxxxxxxx |␣
↪xxxxxxxxxxxxxx | xxxxxxxxxxxxxxxxxx
```

**Return type** pandas.DataFrame - etfs

**Raises**

- **ValueError** – raised when any of the input arguments is not valid.
- **FileNotFoundError** – raised when *etfs.csv* file was not found.
- **IOError** – raised when *etfs.csv* file is missing.

investpy.etfs.**get_etfs_dict**(*country=None*, *columns=None*, *as_json=False*)

This function retrieves all the available etfs indexed on Investing.com, already stored on *etfs.csv*. This function also allows the user to specify which country do they want to retrieve data from, or from every listed country; the columns which the user wants to be included on the resulting dict; and the output of the function will either be a dict or a json.

**Parameters**

- **country** (str, optional) – name of the country to retrieve all its available etfs from.
- **columns** (list, optional) – names of the columns of the etf data to retrieve <country, name, full_name, symbol, isin, asset_class, currency, stock_exchange>
- **as_json** (bool, optional) – value to determine the format of the output data which can either be a dict or a json.

**Returns**

The resulting dict contains the retrieved data if found, if not, the corresponding fields are filled with *None* values.

In case the information was successfully retrieved, the dict will look like:

```
etfs_dict = {
    "country": country,
    "name": name,
    "full_name": full_name,
    "symbol": symbol,
    "isin": isin,
    "asset_class": asset_class,
    "currency": currency,
```

(continues on next page)

(continued from previous page)

```
    "stock_exchange": stock_exchange,
    "def_stock_exchange": def_stock_exchange
}
```

**Return type** dict or json - etfs_dict

**Raises**

- **ValueError** – raised when any of the input arguments is not valid.
- **FileNotFoundError** – raised when *etfs.csv* file was not found.
- **IOError** – raised when *etfs.csv* file is missing.

investpy.etfs.**get_etfs_list**(*country=None*)

This function retrieves all the available etfs indexed on Investing.com, already stored on *etfs.csv*. This function also allows the users to specify which country do they want to retrieve data from or if they want to retrieve it from every listed country; so on, a listing of etfs will be returned. This function helps the user to get to know which etfs are available on Investing.com.

**Parameters** **country** (str, optional) – name of the country to retrieve all its available etfs from.

**Returns**

The resulting list contains the retrieved data from the *etfs.csv* file, which is a listing of the names of the etfs listed on Investing.com, which is the input for data retrieval functions as the name of the etf to retrieve data from needs to be specified.

In case the listing was successfully retrieved, the list will look like:

```
etfs_list = [
    'Betashares U.S. Equities Strong Bear Currency Hedg',
    'Betashares Active Australian Hybrids',
    'Australian High Interest Cash', ...
]
```

**Return type** list - etfs_list

**Raises**

- **ValueError** – raised when any of the input arguments is not valid.
- **FileNotFoundError** – raised when *etfs.csv* file was not found.
- **IOError** – raised when *etfs.csv* file is missing.

investpy.etfs.**get_etfs_overview**(*country*, *as_json=False*, *n_results=100*)

This function retrieves an overview containing all the real time data available for the main ETFs from a country, such as the ETF names, symbols, current value, etc. as indexed in Investing.com. So on, the main usage of this function is to get an overview on the main ETFs from a country, so to get a general view. Note that since this function is retrieving a lot of information at once, by default just the overview of the Top 100 ETFs is being retrieved, but an additional parameter called n_results can be specified so to retrieve N results.

**Parameters**

- **country** (str) – name of the country to retrieve the ETFs overview from.
- **as_json** (bool, optional) – optional argument to determine the format of the output data (pandas.DataFrame or json).
- **n_results** (int, optional) – number of results to be displayed on the overview table (0-1000).

**Returns**

The resulting `pandas.DataFrame` contains all the data available in Investing.com of the main ETFs from a country in order to get an overview of it.

If the retrieval process succeeded, the resulting `pandas.DataFrame` should look like:

```
country | name | full_name | symbol | last | change | turnover
--------|------|-----------|--------|------|--------|----------
xxxxxxx | xxxx | xxxxxxxxx | xxxxxx | xxxx | xxxxxx | xxxxxxxx
```

**Return type** `pandas.DataFrame` - etfs_overview

**Raises**

- **ValueError** – raised if there was any argument error.
- **FileNotFoundError** – raised when *etfs.csv* file is missing.
- **IOError** – raised if data could not be retrieved due to file error.
- **RuntimeError** – raised either if the introduced country does not match any of the listed ones or if no overview results could be retrieved from Investing.com.
- **ConnectionError** – raised if GET requests does not return 200 status code.

`investpy.etfs.`**search_etfs**(*by*, *value*)

This function searches etfs by the introduced value for the specified field. This means that this function is going to search if there is a value that matches the introduced value for the specified field which is the *etfs.csv* column name to search in. Available fields to search etfs are 'name', 'full_name' and 'symbol'.

**Parameters**

- **by** (`str`) – name of the field to search for, which is the column name ('name', 'full_name' or 'symbol').
- **value** (`str`) – value of the field to search for, which is the str that is going to be searched.

**Returns** The resulting *pandas.DataFrame* contains the search results from the given query (the specified value in the specified field). If there are no results and error will be raised, but otherwise this *pandas.DataFrame* will contain all the available field values that match the introduced query.

**Return type** `pandas.DataFrame` - search_result

**Raises**

- **ValueError** – raised if any of the introduced params is not valid or errored.
- **FileNotFoundError** – raised if *etfs.csv* file is missing.
- **IOError** – raised if data could not be retrieved due to file error.
- **RuntimeError** – raised if no results were found for the introduced value in the introduced field.

## 9.4 investpy.indices

investpy.indices.**get_index_countries**()

This function retrieves all the country names indexed in Investing.com with available indices to retrieve data from, via reading the *indices.csv* file from the resources directory. So on, this function will display a listing containing a set of countries, in order to let the user know which countries are available for indices data retrieval.

**Returns** The resulting `list` contains all the available countries with indices as indexed in Investing.com

**Return type** `list` - countries

**Raises**

- **FileNotFoundError** – raised if the *indices.csv* file was not found.
- **IOError** – raised if the *indices.csv* file is missing or errored.

investpy.indices.**get_index_historical_data**(*index*, *country*, *from_date*, *to_date*, *as_json=False*, *order='ascending'*, *interval='Daily'*)

This function retrieves historical data of the introduced *index* (from the specified country, note that both index and country should match since if the introduced index is not listed in the indices of that country, the function will raise an error). The retrieved historical data are the OHLC values plus the Volume and the Currency in which those values are specified, from the introduced date range if valid. So on, the resulting data can it either be stored in a `pandas.DataFrame` or in a `json` file.

**Parameters**

- **index** (`str`) – name of the index to retrieve recent historical data from.
- **country** (`str`) – name of the country from where the index is.
- **from_date** (`str`) – date as *str* formatted as *dd/mm/yyyy*, from where data is going to be retrieved.
- **to_date** (`str`) – date as *str* formatted as *dd/mm/yyyy*, until where data is going to be retrieved.
- **as_json** (`bool`, optional) – optional argument to determine the format of the output data (`pandas.DataFrame` or `json`).
- **interval** (`str`, optional) – value to define the historical data interval to retrieve, by default *Daily*, but it can also be *Weekly* or *Monthly*.

**Returns**

The function returns either a `pandas.DataFrame` or a `json` file containing the retrieved historical data from the specified index via argument. The dataset contains the open, high, low, close and volume values for the selected index on market days, additionally the currency in which those values are specified is returned.

The returned data is case we use default arguments will look like:

```
Date || Open | High | Low | Close | Volume | Currency
-----||------|------|-----|-------|--------|----------
xxxx || xxxx | xxxx | xxx | xxxxx | xxxxxx | xxxxxxxx
```

but if we define *as_json=True*, then the output will be:

```
{
    name: name,
    historical: [
        {
            date: dd/mm/yyyy,
            open: x,
            high: x,
            low: x,
            close: x,
            volume: x,
            currency: x
        },
        ...
    ]
}
```

**Return type** pandas.DataFrame or json

**Raises**

- **ValueError** – raised if there was an argument error.
- **IOError** – raised if indices object/file was not found or unable to retrieve.
- **RuntimeError** – raised if the introduced index does not match any of the indexed ones.
- **ConnectionError** – raised if GET requests does not return 200 status code.
- **IndexError** – raised if index information was unavailable or not found.

### Examples

```
>>> data = investpy.get_index_historical_data(index='ibex 35', country='spain',␣
↪from_date='01/01/2018', to_date='01/01/2019')
>>> data.head()
               Open     High      Low    Close    Volume Currency
Date
2018-01-02  15128.2  15136.7  14996.6  15096.8  10340000      EUR
2018-01-03  15145.0  15186.9  15091.9  15106.9  12800000      EUR
2018-01-04  15105.5  15368.7  15103.7  15368.7  17070000      EUR
2018-01-05  15353.9  15407.5  15348.6  15398.9  11180000      EUR
2018-01-08  15437.1  15448.7  15344.0  15373.3  12890000      EUR
```

investpy.indices.**get_index_information**(*index*, *country*, *as_json=False*)

This function retrieves fundamental financial information from the specified index. The retrieved information from the index can be valuable as it is additional information that can be used combined with OHLC values, so to determine financial insights from the company which holds the specified index.

**Parameters**

- **index** (str) – name of the index to retrieve recent historical data from.
- **country** (str) – name of the country from where the index is.
- **as_json** (bool, optional) – optional argument to determine the format of the output data (dict or json).

**Returns**

The resulting pandas.DataFrame contains the information fields retrieved from Investing.com from the specified index; it can also be returned as a dict, if argument *as_json=True*.

If any of the information fields could not be retrieved, that field/s will be filled with None values. If the retrieval process succeeded, the resulting dict will look like:

```
index_information = {
    "Index Name": "S&P Merval",
    "Prev. Close": 36769.59,
    "Volume": None,
    "Todays Range": "36,769.59 - 37,894.32",
    "Open": 36769.59,
    "Average Vol. (3m)": None,
    "52 wk Range": "22,484.4 - 44,470.76",
    "1-Year Change": "18.19%"
}
```

**Return type** pandas.DataFrame or dict- index_information

**Raises**

- **ValueError** – raised if any of the introduced arguments is not valid or errored.
- **FileNotFoundError** – raised if *indices.csv* file was not found or errored.
- **IOError** – raised if *indices.csv* file is empty or errored.
- **RuntimeError** – raised if scraping process failed while running.
- **ConnectionError** – raised if the connection to Investing.com errored (did not return HTTP 200)

investpy.indices.**get_index_recent_data**(*index*, *country*, *as_json=False*, *order='ascending'*, *interval='Daily'*)

This function retrieves recent historical data from the introduced *index* from Investing via Web Scraping. The resulting data can it either be stored in a pandas.DataFrame or in a json file, with *ascending* or *descending* order.

**Parameters**

- **index** (str) – name of the index to retrieve recent historical data from.
- **country** (str) – name of the country from where the index is.
- **as_json** (bool, optional) – optional argument to determine the format of the output data (pandas.DataFrame or json).
- **order** (str, optional) – optional argument to define the order of the retrieved data (*ascending*, *asc* or *descending*, *desc*).
- **interval** (str, optional) – value to define the historical data interval to retrieve, by default *Daily*, but it can also be *Weekly* or *Monthly*.

**Returns**

The function returns either a pandas.DataFrame or a json file containing the retrieved recent data from the specified index via argument. The dataset contains the open, high, low, close and volume values for the selected index on market days, additionally the currency value is returned.

The returned data is case we use default arguments will look like:

```
Date || Open | High | Low | Close | Volume | Currency
-----||------|------|-----|-------|--------|----------
xxxx || xxxx | xxxx | xxx | xxxxx | xxxxxx | xxxxxxxx
```

but if we define *as_json=True*, then the output will be:

```
{
    name: name,
    recent: [
        {
            date: dd/mm/yyyy,
            open: x,
            high: x,
            low: x,
            close: x,
            volume: x,
            currency: x
        },
        ...
    ]
}
```

**Return type** `pandas.DataFrame` or `json`

**Raises**

- **`ValueError`** – raised if there was an argument error.
- **`IOError`** – raised if indices object/file was not found or unable to retrieve.
- **`RuntimeError`** – raised if the introduced index does not match any of the indexed ones.
- **`ConnectionError`** – raised if GET requests does not return 200 status code.
- **`IndexError`** – raised if index information was unavailable or not found.

### Examples

```
>>> data = investpy.get_index_recent_data(index='ibex 35', country='spain')
>>> data.head()
               Open     High      Low    Close   Volume Currency
Date
2019-08-26  12604.7  12646.3  12510.4  12621.3  4770000      EUR
2019-08-27  12618.3  12723.3  12593.6  12683.8  8230000      EUR
2019-08-28  12657.2  12697.2  12585.1  12642.5  7300000      EUR
2019-08-29  12637.2  12806.6  12633.8  12806.6  5650000      EUR
2019-08-30  12767.6  12905.9  12756.9  12821.6  6040000      EUR
```

`investpy.indices.`**`get_indices`**(*country=None*)

This function retrieves all the available *indices* from Investing.com as previously listed in investpy, and returns them as a `pandas.DataFrame` with all the information of every available index. If the country filtering is applied, just the indices from the introduced country are going to be returned. All the available indices can be found at: https://www.investing.com/indices/world-indices and at https://www.investing.com/indices/world-indices, since both world and global indices are retrieved.

**Parameters** **`country`** (`str`, optional) – name of the country to retrieve all its available indices from.

**Returns**

The resulting `pandas.DataFrame` contains all the indices information retrieved from Investing.com, as previously listed by investpy.

In case the information was successfully retrieved, the `pandas.DataFrame` will look like:

```
country | name | full_name | symbol | currency | class | market
--------|------|-----------|--------|----------|-------|--------
xxxxxxx | xxxx | xxxxxxxxx | xxxxxx | xxxxxxxx | xxxxx | xxxxxx
```

**Return type** `pandas.DataFrame` - indices_df

**Raises**

- **`ValueError`** – raised if any of the introduced parameters is missing or errored.
- **`FileNotFoundError`** – raised if the *indices.csv* file was not found.
- **`IOError`** – raised if the *indices.csv* file from *investpy* is missing or errored.

`investpy.indices.`**`get_indices_dict`**(*country=None*, *columns=None*, *as_json=False*)

This function retrieves all the available *indices* from Investing.com as previously listed in investpy, and returns them as a `dict` with all the information of every available index. If the country filtering is applied, just the indices from the introduced country are going to be returned. Additionally, the columns to retrieve data from can be specified as a parameter formatted as a `list`. All the available indices can be found at: https://www.investing.com/indices/world-indices and at https://www.investing.com/indices/world-indices, since both world and global indices are retrieved.

**Parameters**

- **`country`** (`str`, optional) – name of the country to retrieve all its available indices from.
- **`columns`** (`list` of `str`, optional) – description a `list` containing the column names from which the data is going to be retrieved.
- **`as_json`** (`bool`, optional) – description value to determine the format of the output data (`dict` or `json`).

**Returns**

The resulting `dict` contains the retrieved data if found, if not, the corresponding fields are filled with *None* values.

In case the information was successfully retrieved, the `dict` will look like:

```
indices_dict = {
    'country': country,
    'name': name,
    'full_name': full_name,
    'symbol': symbol,
    'currency': currency,
    'class': class,
    'market': market
}
```

**Return type** `dict` or `json` - indices_dict

**Raises**

- **`ValueError`** – raised whenever any of the introduced arguments is not valid or errored.
- **`FileNotFoundError`** – raised if the *indices.csv* file was not found.

- **IOError** – raised if the *indices.csv* file is missing or errored.

investpy.indices.**get_indices_list**(*country=None*)

This function retrieves all the available *indices* from Investing.com as previously listed in investpy, and returns them as a `list` with the names of every available index. If the country filtering is applied, just the indices from the introduced country are going to be returned. All the available indices can be found at: https://www.investing.com/indices/world-indices and at https://www.investing.com/indices/world-indices, since both world and global indices are retrieved.

**Parameters** **country** (`str`, optional) – name of the country to retrieve all its available indices from.

**Returns**

The resulting `list` contains the retrieved data, which corresponds to the index names of every index listed in Investing.com.

In case the information was successfully retrieved, the `list` will look like:

```
indices = ['S&P Merval', 'S&P Merval Argentina', 'S&P/BYMA Argentina␣
↪General', ...]
```

**Return type** `list` - indices_list

**Raises**

- **ValueError** – raised whenever any of the introduced arguments is not valid or errored.
- **FileNotFoundError** – raised if the *indices.csv* file was not found.
- **IOError** – raised if the *indices.csv* file is missing or errored.

investpy.indices.**get_indices_overview**(*country*, *as_json=False*, *n_results=100*)

This function retrieves an overview containing all the real time data available for the main indices from a country, such as the names, symbols, current value, etc. as indexed in Investing.com. So on, the main usage of this function is to get an overview on the main indices from a country, so to get a general view. Note that since this function is retrieving a lot of information at once, by default just the overview of the Top 100 indices is being retrieved, but an additional parameter called n_results can be specified so to retrieve N results.

**Parameters**

- **country** (`str`) – name of the country to retrieve the indices overview from.
- **as_json** (`bool`, optional) – optional argument to determine the format of the output data (`pandas.DataFrame` or `json`).
- **n_results** (`int`, optional) – number of results to be displayed on the overview table (0-1000).

**Returns**

The resulting `pandas.DataFrame` contains all the data available in Investing.com of the main indices from a country in order to get an overview of it.

If the retrieval process succeeded, the resulting `pandas.DataFrame` should look like:

```
country | name | last | high | low | change | change_percentage |␣
↪currency
--------|------|------|------|-----|--------|-------------------|-----
↪-----
xxxxxxx | xxxx | xxxx | xxxx | xxx | xxxxxx | xxxxxxxxxxxxxxxxx |␣
↪xxxxxxxx
```

**Return type** pandas.DataFrame - indices_overview

**Raises**

- **ValueError** – raised if any of the introduced arguments is not valid or errored.
- **FileNotFoundError** – raised when *indices.csv* file is missing.
- **IOError** – raised if data could not be retrieved due to file error.
- **RuntimeError** – raised either if the introduced country does not match any of the listed ones or if no overview results could be retrieved from Investing.com.
- **ConnectionError** – raised if GET requests does not return 200 status code.

investpy.indices.**search_indices**(*by*, *value*)

This function searches indices by the introduced value for the specified field. This means that this function is going to search if there is a value that matches the introduced value for the specified field which is the *indices.csv* column name to search in. Available fields to search indices are 'name', 'full_name' and 'symbol'.

**Parameters**

- **by** (str) – name of the field to search for, which is the column name ('name', 'full_name' or 'symbol').
- **value** (str) – value of the field to search for, which is the str that is going to be searched.

**Returns** The resulting *pandas.DataFrame* contains the search results from the given query (the specified value in the specified field). If there are no results and error will be raised, but otherwise this *pandas.DataFrame* will contain all the available field values that match the introduced query.

**Return type** pandas.DataFrame - search_result

**Raises**

- **ValueError** – raised if any of the introduced params is not valid or errored.
- **FileNotFoundError** – raised if *indices.csv* file is missing.
- **IOError** – raised if data could not be retrieved due to file error.
- **RuntimeError** – raised if no results were found for the introduced value in the introduced field.

## 9.5 investpy.currency_crosses

investpy.currency_crosses.**get_available_currencies**()

This function retrieves a listing with all the available currencies with indexed currency crosses in order to get to know which are the available currencies. The currencies listed in this function, so on, can be used to search currency crosses and used the retrieved data to get historical data of those currency crosses, so to determine which is the value of one base currency in the second currency.

**Returns**

The resulting list contains all the available currencies with currency crosses being either the base or the second value of the cross, as listed in Investing.com.

In case the listing was successfully retrieved, the list will look like:

```
available_currencies = [
    'AED', 'AFN', 'ALL', 'AMD', 'ANG', ...
]
```

**Return type** `list` - available_currencies

**Raises**

- **FileNotFoundError** – raised if *currency_crosses.csv* file was not found.
- **IOError** – raised if currency crosses retrieval failed, both for missing file or empty file.

`investpy.currency_crosses.`**get_currency_cross_historical_data**(*currency_cross*, *from_date*, *to_date*, *as_json=False*, *order='ascending'*, *interval='Daily'*)

This function retrieves recent historical data from the introduced *currency_cross* from Investing via Web Scrapping. The resulting data can it either be stored in a `pandas.DataFrame` or in a `json` file, with *ascending* or *descending* order.

**Parameters**

- **currency_cross** (`str`) – name of the currency cross to retrieve recent historical data from.
- **from_date** (`str`) – date as *str* formatted as *dd/mm/yyyy*, from where data is going to be retrieved.
- **to_date** (`str`) – date as *str* formatted as *dd/mm/yyyy*, until where data is going to be retrieved.
- **as_json** (`bool`, optional) – optional argument to determine the format of the output data (`pandas.DataFrame` or `json`).
- **order** (`str`, optional) – optional argument to define the order of the retrieved data (*ascending*, *asc* or *descending*, *desc*).
- **interval** (`str`, optional) – value to define the historical data interval to retrieve, by default *Daily*, but it can also be *Weekly* or *Monthly*.

**Returns**

The function returns a either a `pandas.DataFrame` or a `json` file containing the retrieved recent data from the specified currency_cross via argument. The dataset contains the open, high, low, close and volume values for the selected currency_cross on market days.

The return data is case we use default arguments will look like:

```
Date || Open | High | Low | Close | Currency
-----||------|------|-----|-------|---------
xxxx || xxxx | xxxx | xxx | xxxxx | xxxxxxxx
```

but if we define *as_json=True*, then the output will be:

```
{
    name: name,
    historical: [
        dd/mm/yyyy: {
            'open': x,
            'high': x,
            'low': x,
            'close': x,
            'currency' : x
        },
        ...
```

(continues on next page)

(continued from previous page)

```
    ]
}
```

**Return type** pandas.DataFrame or json

**Raises**

- **ValueError** – argument error.
- **IOError** – stocks object/file not found or unable to retrieve.
- **RuntimeError** – introduced currency_cross does not match any of the indexed ones.
- **ConnectionError** – if GET requests does not return 200 status code.
- **IndexError** – if currency_cross information was unavailable or not found.

### Examples

```
>>> data = investpy.get_currency_cross_historical_data(currency_cross='EUR/USD',␣
↪from_date='01/01/2018', to_date='01/01/2019')
>>> data.head()
              Open    High     Low   Close Currency
Date
2018-01-01  1.2003  1.2014  1.1995  1.2010      USD
2018-01-02  1.2013  1.2084  1.2003  1.2059      USD
2018-01-03  1.2058  1.2070  1.2001  1.2014      USD
2018-01-04  1.2015  1.2090  1.2004  1.2068      USD
2018-01-05  1.2068  1.2085  1.2021  1.2030      USD
```

investpy.currency_crosses.**get_currency_cross_information**(*currency_cross*, *as_json=False*)

This function retrieves fundamental financial information from the specified currency cross. The retrieved information from the currency cross can be valuable as it is additional information that can be used combined with OHLC values, so to determine financial insights from the company which holds the specified currency cross.

**Parameters**

- **currency_cross** (str) – name of the currency_cross to retrieve recent historical data from.
- **as_json** (bool, optional) – optional argument to determine the format of the output data (dict or json).

**Returns**

The resulting pandas.DataFrame contains the information fields retrieved from Investing.com from the specified currency cross; it can also be returned as a dict, if argument *as_json=True*.

If any of the information fields could not be retrieved, that field/s will be filled with None values. If the retrieval process succeeded, the resulting dict will look like:

```
currency_cross_information = {
    "1-Year Change": "- 1.61%",
    "52 wk Range": "1.0879 - 1.1572",
    "Ask": 1.1144,
    "Bid": 1.114,
    "Currency Cross": "EUR/USD",
```

(continues on next page)

(continued from previous page)

```
    "Open": 1.1121,
    "Prev. Close": 1.1119,
    "Todays Range": "1.1123 - 1.1159"
}
```

**Return type** `pandas.DataFrame` or `dict`- currency cross_information

**Raises**

- **`ValueError`** – raised if any of the introduced arguments is not valid or errored.
- **`FileNotFoundError`** – raised if *currency_crosses.csv* file was not found.
- **`IOError`** – raised if *currency_crosses.csv* file is empty or errored.
- **`RuntimeError`** – raised if scraping process failed while running.
- **`ConnectionError`** – raised if the connection to Investing.com errored (did not return HTTP 200)

`investpy.currency_crosses.`**`get_currency_cross_recent_data`**(*currency_cross*, *as_json=False*, *order='ascending'*, *interval='Daily'*)

This function retrieves recent historical data from the introduced *currency_cross* as indexed in Investing.com via Web Scraping. The resulting data can it either be stored in a `pandas.DataFrame` or in a `json` file, with *ascending* or *descending* order.

**Parameters**

- **`currency_cross`** (`str`) – name of the currency_cross to retrieve recent historical data from.
- **`as_json`** (`bool`, optional) – optional argument to determine the format of the output data (`pandas.DataFrame` or `json`).
- **`order`** (`str`, optional) – optional argument to define the order of the retrieved data (*ascending*, *asc* or *descending*, *desc*).
- **`interval`** (`str`, optional) – value to define the historical data interval to retrieve, by default *Daily*, but it can also be *Weekly* or *Monthly*.

**Returns**

The function returns a either a `pandas.DataFrame` or a `json` file containing the retrieved recent data from the specified currency_cross via argument. The dataset contains the open, high, low, close, volume and currency values for the selected currency_cross on market days.

The return data is in case we use default arguments will look like:

```
Date || Open | High | Low | Close | Currency
-----||------|------|-----|-------|---------
xxxx || xxxx | xxxx | xxx | xxxxx | xxxxxxxx
```

but if we define *as_json=True*, then the output will be:

```
{
    name: name,
    recent: [
        dd/mm/yyyy: {
            'open': x,
```

(continues on next page)

(continued from previous page)

```
            'high': x,
            'low': x,
            'close': x,
            'currency' : x
        },
        ...
    ]
}
```

**Return type** pandas.DataFrame or json

**Raises**

- **ValueError** – raised if any of the introduced arguments was not valid or errored.
- **IOError** – raised if currency_crosses object/file not found or unable to retrieve.
- **RuntimeError** – raised introduced currency_cross does not match any of the indexed ones.
- **ConnectionError** – raised if GET request did not return 200 status code.
- **IndexError** – raised if currency_cross information was unavailable or not found.

### Examples

```
>>> data = investpy.get_currency_cross_recent_data(currency_cross='EUR/USD')
>>> data.head()
              Open    High     Low   Close Currency
Date
2019-08-27  1.1101  1.1116  1.1084  1.1091      USD
2019-08-28  1.1090  1.1099  1.1072  1.1078      USD
2019-08-29  1.1078  1.1093  1.1042  1.1057      USD
2019-08-30  1.1058  1.1062  1.0963  1.0991      USD
2019-09-02  1.0990  1.1000  1.0958  1.0968      USD
```

investpy.currency_crosses.**get_currency_crosses**(*base=None*, *second=None*)

This function retrieves all the available currency crosses from Investing.com and returns them as a pandas.DataFrame, which contains not just the currency crosses names, but all the fields contained on the currency_crosses file. Note that the filtering params are both base and second, which mean the base and the second currency of the currency cross, for example, in the currency cross *EUR/USD* the base currency is EUR and the second currency is USD. These are optional parameters, so specifying one of them means that all the currency crosses where the introduced currency is either base or second will be returned; if both are specified, just the introduced currency cross will be returned if it exists. All the available currency crosses can be found at: https://www.investing.com/currencies/

**Parameters**

- **base** (str, optional) – symbol of the base currency of the currency cross, this will return a pandas.DataFrame containing all the currency crosses where the base currency matches the introduced one.
- **second** (str) – symbol of the second currency of the currency cross, this will return a pandas.DataFrame containing all the currency crosses where the second currency matches the introduced one.

**Returns**

The resulting `pandas.DataFrame` contains all the currency crosses basic information retrieved from Investing.com.

In case the information was successfully retrieved, the resulting `pandas.DataFrame` will look like:

```
name | full_name | base | second | base_name | second_name
-----|-----------|------|--------|-----------|-------------
xxxx | xxxxxxxxx | xxxx | xxxxxx | xxxxxxxxx | xxxxxxxxxxx
```

**Return type** `pandas.DataFrame` - currency_crosses_df

**Raises**

- **`ValueError`** – raised if any of the introduced arguments is not valid or errored.
- **`FileNotFoundError`** – raised if *currency_crosses.csv* file was not found.
- **`IOError`** – raised if currency crosses retrieval failed, both for missing file or empty file.

`investpy.currency_crosses.`**`get_currency_crosses_dict`**(*base=None*, *second=None*, *columns=None*, *as_json=False*)

This function retrieves all the available currency crosses from Investing.com and returns them as a `dict`, which contains not just the currency crosses names, but all the fields contained on the currency_crosses file is columns is None, otherwise, just the specified column values will be returned. Note that the filtering params are both base and second, which mean the base and the second currency of the currency cross, for example, in the currency cross *EUR/USD* the base currency is EUR and the second currency is USD. These are optional parameters, so specifying one of them means that all the currency crosses where the introduced currency is either base or second will be returned; if both are specified, just the introduced currency cross will be returned if it exists. All the available currency crosses can be found at: https://www.investing.com/currencies/

**Parameters**

- **`base`** (`str`, optional) – symbol of the base currency of the currency cross, this will return a `pandas.DataFrame` containing all the currency crosses where the base currency matches the introduced one.
- **`second`** (`str`) – symbol of the second currency of the currency cross, this will return a `pandas.DataFrame` containing all the currency crosses where the second currency matches the introduced one.
- **`columns`** (`list`, optional) – names of the columns of the currency crosses data to retrieve <name, full_name, base, base_name, second, second_name>
- **`as_json`** (`bool`, optional) – value to determine the format of the output data which can either be a `dict` or a `json`.

**Returns**

The resulting `dict` contains the retrieved data if found, if not, the corresponding fields are filled with *None* values.

In case the information was successfully retrieved, the `dict` will look like:

```
{
    'name': name,
    'full_name': full_name,
    'base': base,
    'base_name': base_name,
    'second': second,
    'second_name': second_name
}
```

**Return type** `dict` or `json` - currency_crosses_dict

**Raises**

- **ValueError** – raised if any of the introduced arguments is not valid or errored.
- **FileNotFoundError** – raised if *currency_crosses.csv* file was not found.
- **IOError** – raised if currency crosses retrieval failed, both for missing file or empty file.

`investpy.currency_crosses.`**get_currency_crosses_list**(*base=None*, *second=None*)

This function retrieves all the available currency crosses from Investing.com and returns them as a `dict`, which contains not just the currency crosses names, but all the fields contained on the currency_crosses file is columns is None, otherwise, just the specified column values will be returned. Note that the filtering params are both base and second, which mean the base and the second currency of the currency cross, for example, in the currency cross *EUR/USD* the base currency is EUR and the second currency is USD. These are optional parameters, so specifying one of them means that all the currency crosses where the introduced currency is either base or second will be returned; if both are specified, just the introduced currency cross will be returned if it exists. All the available currency crosses can be found at: https://www.investing.com/currencies/

**Parameters**

- **base** (`str`, optional) – symbol of the base currency of the currency cross, this will return a `pandas.DataFrame` containing all the currency crosses where the base currency matches the introduced one.
- **second** (`str`) – symbol of the second currency of the currency cross, this will return a `pandas.DataFrame` containing all the currency crosses where the second currency matches the introduced one.

**Returns**

The resulting `list` contains the retrieved data from the *currency_crosses.csv* file, which is a listing of the names of the currency crosses listed in Investing.com, which is the input for data retrieval functions as the name of the currency cross to retrieve data from needs to be specified.

In case the listing was successfully retrieved, the `list` will look like:

```
currency_crosses_list = [
    'USD/BRLT', 'CAD/CHF', 'CHF/CAD', 'CAD/PLN', 'PLN/CAD', ...
]
```

**Return type** `list` - currency_crosses_list

**Raises**

- **ValueError** – raised if any of the introduced arguments is not valid or errored.
- **FileNotFoundError** – raised if *currency_crosses.csv* file was not found.
- **IOError** – raised if currency crosses retrieval failed, both for missing file or empty file.

`investpy.currency_crosses.`**get_currency_crosses_overview**(*currency*, *as_json=False*, *n_results=100*)

This function retrieves an overview containing all the real time data available for the main stocks from a country, such as the names, symbols, current value, etc. as indexed in Investing.com. So on, the main usage of this function is to get an overview on the main stocks from a country, so to get a general view. Note that since this function is retrieving a lot of information at once, by default just the overview of the Top 100 stocks is being retrieved, but an additional parameter called n_results can be specified so to retrieve N results.

**Parameters**

- **currency** (`str`) – name of the currency to retrieve the currency crosses overview from.

- **as_json** (bool, optional) – optional argument to determine the format of the output data (pandas.DataFrame or json).
- **n_results** (int, optional) – number of results to be displayed on the overview table (0-1000).

**Returns**

The resulting pandas.DataFrame contains all the data available in Investing.com of the main currency crosses from a given currency in order to get an overview of them.

If the retrieval process succeeded, the resulting pandas.DataFrame should look like:

```
symbol | name | bid | ask | high | low | change | change_percentage
-------|------|-----|-----|------|-----|--------|-------------------
xxxxxx | xxxx | xxx | xxx | xxxx | xxx | xxxxxx | xxxxxxxxxxxxxxxxx
```

**Return type** pandas.DataFrame - stocks_overview

**Raises**

- **ValueError** – raised if any of the introduced arguments errored.
- **FileNotFoundError** – raised if *currencies.csv* file is missing.
- **IOError** – raised if data could not be retrieved due to file error.
- **RuntimeError** – raised either if the introduced currency does not match any of the listed ones or if no overview results could be retrieved from Investing.com.
- **ConnectionError** – raised if GET requests does not return 200 status code.

investpy.currency_crosses.**search_currency_crosses**(*by*, *value*)

This function searches currency crosses by the introduced value for the specified field. This means that this function is going to search if there is a value that matches the introduced value for the specified field which is the *currency_crosses.csv* column name to search in. Available fields to search indices are 'name', 'full_name', 'base', 'second', 'base_name' and 'second_name'.

**Parameters**

- **by** (str) – name of the field to search for, which is the column name ('name', 'full_name', 'base', 'second', 'base_name' or 'second_name').
- **value** (str) – value of the field to search for, which is the str that is going to be searched.

**Returns** The resulting *pandas.DataFrame* contains the search results from the given query (the specified value in the specified field). If there are no results and error will be raised, but otherwise this *pandas.DataFrame* will contain all the available field values that match the introduced query.

**Return type** pandas.DataFrame - search_result

**Raises**

- **ValueError** – raised if any of the introduced params is not valid or errored.
- **FileNotFoundError** – raised if *currency_crosses.csv* file is missing.
- **IOError** – raised if data could not be retrieved due to file error.
- **RuntimeError** – raised if no results were found for the introduced value in the introduced field.

## 9.6 investpy.bonds

investpy.bonds.**get_bond_countries**()

This function returns a listing with all the available countries from where bonds can be retrieved, so to let the user know which of them are available, since the parameter country is mandatory in every bond retrieval function. Also, not just the available countries, but the required name is provided since Investing.com has a certain country name standard and countries should be specified the same way they are in Investing.com.

**Returns** The resulting `list` contains all the available countries with government bonds as indexed in Investing.com

**Return type** `list` - countries

**Raises**

- **FileNotFoundError** – raised when bond countries file was not found.
- **IOError** – raised when bond countries file is missing or empty.

investpy.bonds.**get_bond_historical_data**(*bond*, *from_date*, *to_date*, *as_json=False*, *order='ascending'*, *interval='Daily'*)

This function retrieves historical data from the introduced bond from Investing.com. So on, the historical data of the introduced bond in the specified date range will be retrieved and returned as a `pandas.DataFrame` if the parameters are valid and the request to Investing.com succeeds. Note that additionally some optional parameters can be specified: as_json and order, which let the user decide if the data is going to be returned as a `json` or not, and if the historical data is going to be ordered ascending or descending (where the index is the date), respectively.

**Parameters**

- **bond** (`str`) – name of the bond to retrieve historical data from.
- **from_date** (`str`) – date formatted as *dd/mm/yyyy*, since when data is going to be retrieved.
- **to_date** (`str`) – date formatted as *dd/mm/yyyy*, until when data is going to be retrieved.
- **as_json** (`bool`, optional) – to determine the format of the output data, either a `pandas.DataFrame` if False and a `json` if True.
- **order** (`str`, optional) – to define the order of the retrieved data which can either be ascending or descending.
- **interval** (`str`, optional) – value to define the historical data interval to retrieve, by default *Daily*, but it can also be *Weekly* or *Monthly*.

**Returns**

The function returns a either a `pandas.DataFrame` or a `json` file containing the retrieved recent data from the specified bond via argument. The dataset contains the open, high, low and close for the selected bond on market days.

The resulting recent data, in case that the default parameters were applied, will look like:

```
Date || Open | High | Low | Close
-----||------|------|-----|-------
xxxx || xxxx | xxxx | xxx | xxxxx
```

but in case that as_json parameter was defined as True, then the output will be:

```
{
    name: name,
    historical: [
        {
            date: 'dd/mm/yyyy',
            open: x,
            high: x,
            low: x,
            close: x
        },
        ...
    ]
}
```

**Return type** pandas.DataFrame or json

**Raises**

- **ValueError** – raised whenever any of the introduced arguments is not valid or errored.
- **IOError** – raised if bonds object/file was not found or unable to retrieve.
- **RuntimeError** – raised if the introduced bond was not found or did not match any of the existing ones.
- **ConnectionError** – raised if connection to Investing.com could not be established.
- **IndexError** – raised if bond historical data was unavailable or not found in Investing.com.

### Examples

```
>>> data = investpy.get_bond_historical_data(bond='Argentina 3Y', from_date='01/
↪01/2010', to_date='01/01/2019')
>>> data.head()
            Open  High   Low  Close
Date
2011-01-03  4.15  4.15  4.15   5.15
2011-01-04  4.07  4.07  4.07   5.45
2011-01-05  4.27  4.27  4.27   5.71
2011-01-10  4.74  4.74  4.74   6.27
2011-01-11  4.30  4.30  4.30   6.56
```

investpy.bonds.**get_bond_information**(*bond*, *as_json=False*)

This function retrieves fundamental financial information from the specified bond. The retrieved information from the bond can be valuable as it is additional information that can be used combined with OHLC values, so to determine financial insights from the company which holds the specified bond.

**Parameters**

- **bond** (str) – name of the bond to retrieve information from.
- **as_json** (bool, optional) – optional argument to determine the format of the output data (dict or json).

**Returns**

The resulting pandas.DataFrame contains the information fields retrieved from Investing.com from the specified bond; it can also be returned as a dict, if argument *as_json=True*.

If any of the information fields could not be retrieved, that field/s will be filled with None values. If the retrieval process succeeded, the resulting `dict` will look like:

```
bond_information = {
    "1-Year Change": "46.91%",
    "52 wk Range": "-0.575 - 0.01",
    "Bond Name": "Spain 1Y",
    "Coupon": "None",
    "Maturity Date": "04/12/2020",
    "Prev. Close": -0.425,
    "Price": 100.417,
    "Price Open": 100.416,
    "Price Range": -100.481,
    "Todays Range": "-0.49 - -0.424"
}
```

**Return type** `pandas.DataFrame` or `dict`- bond_information

**Raises**

- **`ValueError`** – raised if any of the introduced arguments is not valid or errored.
- **`FileNotFoundError`** – raised if bonds.csv file was not found or errored.
- **`IOError`** – raised if bonds.csv file is empty or errored.
- **`RuntimeError`** – raised if scraping process failed while running.
- **`ConnectionError`** – raised if the connection to Investing.com errored (did not return HTTP 200)

`investpy.bonds.`**`get_bond_recent_data`**(*bond*, *as_json=False*, *order='ascending'*, *interval='Daily'*)

This function retrieves recent historical data from the introduced bond from Investing.com. So on, the recent data of the introduced bond will be retrieved and returned as a `pandas.DataFrame` if the parameters are valid and the request to Investing.com succeeds. Note that additionally some optional parameters can be specified: as_json and order, which let the user decide if the data is going to be returned as a `json` or not, and if the recent data is going to be ordered ascending or descending (where the index is the date), respectively.

**Parameters**

- **`bond`** (`str`) – name of the bond to retrieve recent historical data from.
- **`as_json`** (`bool`, optional) – to determine the format of the output data, either a `pandas.DataFrame` if False and a `json` if True.
- **`order`** (`str`, optional) – to define the order of the retrieved data which can either be ascending or descending.
- **`interval`** (`str`, optional) – value to define the historical data interval to retrieve, by default *Daily*, but it can also be *Weekly* or *Monthly*.

**Returns**

The function can return either a `pandas.DataFrame` or a `json` object, containing the retrieved recent data of the specified bond. So on, the resulting dataframe contains the open, high, low and close values for the selected bond on market days.

The resulting recent data, in case that the default parameters were applied, will look like:

```
Date || Open | High | Low | Close
-----||------|------|-----|-------
xxxx || xxxx | xxxx | xxx | xxxxx
```

but in case that as_json parameter was defined as True, then the output will be:

```
{
    name: name,
    recent: [
        {
            date: 'dd/mm/yyyy',
            open: x,
            high: x,
            low: x,
            close: x
        },
        ...
    ]
}
```

**Return type** pandas.DataFrame or json

**Raises**

- **ValueError** – raised whenever any of the introduced arguments is not valid or errored.
- **IOError** – raised if bonds object/file was not found or unable to retrieve.
- **RuntimeError** – raised if the introduced bond was not found or did not match any of the existing ones.
- **ConnectionError** – raised if connection to Investing.com could not be established.
- **IndexError** – raised if bond historical data was unavailable or not found in Investing.com.

### Examples

```
>>> data = investpy.get_bond_recent_data(bond='Argentina 3Y')
>>> data.head()
              Open    High     Low   Close
Date
2019-09-23  52.214  52.214  52.214  52.214
2019-09-24  52.323  52.323  52.323  52.323
2019-09-25  52.432  52.432  52.432  52.432
2019-09-26  52.765  52.765  52.765  52.765
2019-09-27  52.876  52.876  52.876  52.876
```

investpy.bonds.**get_bonds**(*country=None*)

This function retrieves all the bonds data stored in *bonds.csv* file, which previously was retrieved from Investing.com. Since the resulting object is a matrix of data, the bonds data is properly structured in rows and columns, where columns are the bond data attribute names. Additionally, country filtering can be specified, which will make this function return not all the stored bond data, but just the data of the bonds from the introduced country.

**Parameters** **country** (str, optional) – name of the country to retrieve all its available bonds from.

**Returns**

The resulting pandas.DataFrame contains all the bond data from the introduced country if specified, or from every country if None was specified, as indexed in Investing.com from the information previously retrieved by investpy and stored on a csv file.

So on, the resulting pandas.DataFrame will look like:

```
country | name | full name
--------|------|-----------
xxxxxxx | xxxx | xxxxxxxxx
```

**Return type** pandas.DataFrame - bonds_df

**Raises**

- **ValueError** – raised whenever any of the introduced arguments is not valid.
- **FileNotFoundError** – raised when bonds file was not found.
- **IOError** – raised when bond countries file is missing or empty.

investpy.bonds.**get_bonds_dict**(*country=None*, *columns=None*, *as_json=False*)

This function retrieves all the bonds information stored in the *bonds.csv* file and formats it as a Python dictionary which contains the same information as the file, but every row is a dict and all of them are contained in a list. Note that the dictionary structure is the same one as the JSON structure. Some optional paramaters can be specified such as the country, columns or as_json, which are a filtering by country so not to return all the bonds but just the ones from the introduced country, the column names that want to be retrieved in case of needing just some columns to avoid unnecessary information load, and whether the information wants to be returned as a JSON object or as a dictionary; respectively.

**Parameters**

- **country** (str, optional) – name of the country to retrieve all its available bonds from.
- **columns** (list, optional) – column names of the bonds data to retrieve, can be: <country, name, full_name>
- **as_json** (bool, optional) – if True the returned data will be a json object, if False, a list of dict.

**Returns**

The resulting list of dict contains the retrieved data from every bond as indexed in Investing.com from the information previously retrieved by investpy and stored on a csv file.

In case the information was successfully retrieved, the list of dict will look like:

```
bonds_dict = {
    'country': country,
    'name': name,
    'full_name': full_name,
}
```

**Return type** list of dict OR json - bonds_dict

**Raises**

- **ValueError** – raised whenever any of the introduced arguments is not valid.
- **FileNotFoundError** – raised when bonds file was not found.
- **IOError** – raised when bond countries file is missing or empty.

investpy.bonds.**get_bonds_list**(*country=None*)

This function retrieves all the bond names as stored in *bonds.csv* file, which contains all the data from the bonds as previously retrieved from Investing.com. So on, this function will just return the government bond names which will be one of the input parameters when it comes to bond data retrieval functions from investpy. Additionally, note that the country filtering can be applied, which is really useful since this function just returns the names and in bond data retrieval functions both the name and the country must be specified and they must match.

**Parameters** **country** (str, optional) – name of the country to retrieve all its available bonds from.

**Returns**

The resulting list contains the all the bond names from the introduced country if specified, or from every country if None was specified, as indexed in Investing.com from the information previously retrieved by investpy and stored on a csv file.

In case the information was successfully retrieved, the list of bond names will look like:

```
bonds_list = ['Argentina 1Y', 'Argentina 3Y', 'Argentina 5Y',
↪'Argentina 9Y', ...]
```

**Return type** list - bonds_list

**Raises**

- **ValueError** – raised whenever any of the introduced arguments is not valid.
- **FileNotFoundError** – raised when bonds file was not found.
- **IOError** – raised when bond countries file is missing or empty.

investpy.bonds.**get_bonds_overview**(*country*, *as_json=False*)

This function retrieves an overview containing all the real time data available for the government bonds from a country, such as the names, symbols, current value, etc. as indexed in Investing.com. So on, the main usage of this function is to get an overview on the government bonds from a country, so to get a general view.

**Parameters**

- **country** (str) – name of the country to retrieve the government bonds overview from.
- **as_json** (bool, optional) – optional argument to determine the format of the output data (pandas.DataFrame or json).

**Returns**

The resulting pandas.DataFrame contains all the data available in Investing.com of the government bonds from a country in order to get an overview of it.

If the retrieval process succeeded, the resulting pandas.DataFrame should look like:

```
country | name | last | last_close | high | low | change | change_
↪percentage
--------|------|------|------------|------|-----|--------|------------
↪-------
xxxxxxx | xxxx | xxxx | xxxxxxxxxx | xxxx | xxx | xxxxxx |␣
↪xxxxxxxxxxxxxxxxx
```

**Return type** pandas.DataFrame - bonds_overview

**Raises**

- **ValueError** – raised if any of the introduced arguments is not valid or errored.
- **FileNotFoundError** – raised if *bonds.csv* file is missing.
- **IOError** – raised if data could not be retrieved due to file error.
- **RuntimeError** – raised either if the introduced country does not match any of the listed ones or if no overview results could be retrieved from Investing.com.
- **ConnectionError** – raised if GET requests does not return 200 status code.

investpy.bonds.**search_bonds**(*by*, *value*)

This function searches bonds by the introduced value for the specified field. This means that this function is going to search if there is a value that matches the introduced one for the specified field which is the *bonds.csv* column name to search in. Available fields to search bonds are 'name' or 'full_name'.

**Parameters**

- **by** (str) – name of the field to search for, which is the column name which can be: 'name' or 'full_name'.
- **value** (str) – value of the field to search for, which is the value that is going to be searched.

**Returns** The resulting pandas.DataFrame contains the search results from the given query, which is any match of the specified value in the specified field. If there are no results for the given query, an error will be raised, but otherwise the resulting pandas.DataFrame will contain all the available bonds that match the introduced query.

**Return type** pandas.DataFrame - search_result

**Raises**

- **ValueError** – raised if any of the introduced parameters is not valid or errored.
- **FileNotFoundError** – raised if *bonds.csv* file is missing.
- **IOError** – raised if data could not be retrieved due to file error.
- **RuntimeError** – raised if no results were found for the introduced value in the introduced field.

## 9.7 investpy.commodities

investpy.commodities.**get_commodities**(*group=None*)

This function retrieves all the commodities data stored in *commodities.csv* file, which previously was retrieved from Investing.com. Since the resulting object is a matrix of data, the commodities data is properly structured in rows and columns, where columns are the commodity data attribute names. Additionally, group filtering can be specified, so that the return commodities are from the specified group instead from every available group. Anyways, since it is an optional parameter it does not need to be specified, which means that if it is None or not specified, all the available commodities will be returned.

**Parameters** **group** (str, optional) – name of the group to retrieve all the available commodities from.

**Returns**

The resulting pandas.DataFrame contains all the commodities data from the introduced group if specified, or from all the commodity groups if None was specified, as indexed in Investing.com from the information previously retrieved by investpy and stored on a csv file.

So on, the resulting pandas.DataFrame will look like:

```
title | country | name | full_name | currency | group
------|---------|------|-----------|----------|-------
xxxxx | xxxxxxx | xxxx | xxxxxxxxx | xxxxxxxx | xxxxx
```

**Return type** pandas.DataFrame - commodities_df

**Raises**

- **ValueError** – raised whenever any of the introduced arguments is not valid.

- **FileNotFoundError** – raised when *commodities.csv* file was not found.
- **IOError** – raised when *commodities.csv* file is missing or empty.

investpy.commodities.**get_commodities_dict**(*group=None*, *columns=None*, *as_json=False*)

This function retrieves all the commodities information stored in the *commodities.csv* file and formats it as a Python dictionary which contains the same information as the file, but every row is a `dict` and all of them are contained in a `list`. Note that the dictionary structure is the same one as the JSON structure. Some optional paramaters can be specified such as the group, columns or as_json, which are the name of the commodity group to filter between all the available commodities so not to return all the commodities but just the ones from the introduced group, the column names that want to be retrieved in case of needing just some columns to avoid unnecessary information load, and whether the information wants to be returned as a JSON object or as a dictionary; respectively.

**Parameters**

- **group** (`str`, optional) – name of the group to retrieve all the available commodities from.
- **columns** (`list`, optional) – column names of the commodities data to retrieve, can be: <title, country, name, full_name, currency, group>
- **as_json** (`bool`, optional) – if True the returned data will be a `json` object, if False, a `list` of `dict`.

**Returns**

The resulting `list` of `dict` contains the retrieved data from every bond as indexed in Investing.com from the information previously retrieved by investpy and stored on a csv file.

In case the information was successfully retrieved, the `list` of `dict` will look like:

```
commodities_dict = {
    'title': title,
    'country': country,
    'name': name,
    'full_name': full_name,
    'currency': currency,
    'group': group,
}
```

**Return type** `list` of `dict` OR `json` - bonds_dict

**Raises**

- **ValueError** – raised whenever any of the introduced arguments is not valid.
- **FileNotFoundError** – raised when *commodities.csv* file was not found.
- **IOError** – raised when *commodities.csv* file is missing or empty.

investpy.commodities.**get_commodities_list**(*group=None*)

This function retrieves all the commodity names as stored in *commodities.csv* file, which contains all the data from the commodities as previously retrieved from Investing.com. So on, this function will just return the commodity names from either all the available groups or from any group, which will later be used when it comes to both recent and historical data retrieval.

**Parameters** **group** (`str`, optional) – name of the group to retrieve all the available commodities from.

**Returns**

The resulting `list` contains the all the commodity names from the introduced group if specified, or from every group if None was specified, as indexed in Investing.com from the information previously retrieved by investpy and stored on a csv file.

In case the information was successfully retrieved, the `list` of commodity names will look like:

```
commodities_list = ['Gold', 'Copper', 'Silver', 'Palladium', 'Platinum
↪', ...]
```

**Return type** `list` - commodities_list

**Raises**

- **ValueError** – raised whenever any of the introduced arguments is not valid.
- **FileNotFoundError** – raised when *commodities.csv* file was not found.
- **IOError** – raised when *commodities.csv* file is missing or empty.

`investpy.commodities.`**get_commodities_overview**(*group*, *as_json=False*, *n_results=100*)

This function retrieves an overview containing all the real time data available for the main commodities from every commodity group (metals, softs, meats, energy and grains), such as the names, symbols, current value, etc. as indexed in Investing.com. So on, the main usage of this function is to get an overview on the main commodities from a group, so to get a general view. Note that since this function is retrieving a lot of information at once, by default just the overview of the Top 100 commodities is being retrieved, but an additional parameter called n_results can be specified so to retrieve N results. Anyways, note that in commodities case, there are just a few ones available.

**Parameters**

- **group** (`str`) – name of the commodity group to retrieve an overview from.
- **as_json** (`bool`, optional) – optional argument to determine the format of the output data (`pandas.DataFrame` or `json`).
- **n_results** (`int`, optional) – number of results to be displayed on the overview table (0-1000).

**Returns**

The resulting `pandas.DataFrame` contains all the data available in Investing.com of the main commodities from a commodity group in order to get an overview of it.

If the retrieval process succeeded, the resulting `pandas.DataFrame` should look like:

```
country | name | last | last_close | high | low | change | change_
↪percentage | currency
--------|------|------|------------|------|-----|--------|------------
↪-------|----------
xxxxxxx | xxxx | xxxx | xxxxxxxxxx | xxxx | xxx | xxxxxx |␣
↪xxxxxxxxxxxxxxxxx | xxxxxxxx
```

**Return type** `pandas.DataFrame` - commodities_overview

**Raises**

- **ValueError** – raised if any of the introduced arguments errored.
- **FileNotFoundError** – raised if *commodities.csv* file is missing.
- **IOError** – raised if data could not be retrieved due to file error.

- **RuntimeError** – raised either if the introduced group does not match any of the listed ones or if no overview results could be retrieved from Investing.com.
- **ConnectionError** – raised if GET requests does not return 200 status code.

investpy.commodities.**get_commodity_groups**()

This function returns a listing with all the available commodity groupsson that a filtering can be applied when retrieving data from commodities. The current available commodity groups are metals, agriculture and energy, which include all the raw materials or commodities included in them.

**Returns** The resulting `list` contains all the available commodity groups as indexed in Investing.com

**Return type** `list` - commodity_groups

**Raises**

- **FileNotFoundError** – raised when *commodities.csv* file was not found.
- **IOError** – raised when *commodities.csv* file is missing or empty.

investpy.commodities.**get_commodity_historical_data**(*commodity*, *from_date*, *to_date*, *country=None*, *as_json=False*, *order='ascending'*, *interval='Daily'*)

This function retrieves historical data from the introduced commodity from Investing.com. So on, the historical data of the introduced commodity in the specified date range will be retrieved and returned as a `pandas.DataFrame` if the parameters are valid and the request to Investing.com succeeds. Note that additionally some optional parameters can be specified: as_json and order, which let the user decide if the data is going to be returned as a `json` or not, and if the historical data is going to be ordered ascending or descending (where the index is the date), respectively.

**Parameters**

- **commodity** (`str`) – name of the commodity to retrieve recent data from.
- **from_date** (`str`) – date formatted as *dd/mm/yyyy*, since when data is going to be retrieved.
- **to_date** (`str`) – date formatted as *dd/mm/yyyy*, until when data is going to be retrieved.
- **country** (`str`, optional) – name of the country to retrieve the commodity data from (if there is more than one country that provides data from the same commodity).
- **as_json** (`bool`, optional) – to determine the format of the output data, either a `pandas.DataFrame` if False and a `json` if True.
- **order** (`str`, optional) – to define the order of the retrieved data which can either be ascending or descending.
- **interval** (`str`, optional) – value to define the historical data interval to retrieve, by default *Daily*, but it can also be *Weekly* or *Monthly*.

**Returns**

The function returns a either a `pandas.DataFrame` or a `json` file containing the retrieved historical data of the specified commodity. So on, the resulting dataframe contains the open, high, low and close values for the selected commodity on market days and the currency in which those values are presented.

The returned data is case we use default arguments will look like:

```
Date || Open | High | Low | Close | Volume | Currency
-----||------|------|-----|-------|--------|----------
xxxx || xxxx | xxxx | xxx | xxxxx | xxxxxx | xxxxxxxx
```

but in case that as_json parameter was defined as True, then the output will be:

```
{
    name: name,
    historical: [
        {
            date: 'dd/mm/yyyy',
            open: x,
            high: x,
            low: x,
            close: x,
            volume: x,
            currency: x
        },
        ...
    ]
}
```

**Return type** `pandas.DataFrame` or `json`

**Raises**

- **`ValueError`** – raised whenever any of the introduced arguments is not valid or errored.
- **`IOError`** – raised if commodities object/file was not found or unable to retrieve.
- **`RuntimeError`** – raised if the introduced commodity was not found or did not match any of the existing ones.
- **`ConnectionError`** – raised if connection to Investing.com could not be established.
- **`IndexError`** – raised if commodity historical data was unavailable or not found in Investing.com.

### Examples

```
>>> data = investpy.get_commodity_historical_data(commodity='gold', from_date='01/
↪01/2018', to_date='01/01/2019')
>>> data.head()
              Open    High     Low   Close  Volume Currency
Date
2018-01-01  1305.8  1309.7  1304.6  1308.7       0      USD
2018-01-02  1370.5  1370.5  1370.5  1370.5      97      USD
2018-01-03  1372.0  1372.0  1369.0  1374.2      22      USD
2018-01-04  1363.4  1375.6  1362.7  1377.4      13      USD
2018-01-05  1377.8  1377.8  1377.8  1378.4      10      USD
```

`investpy.commodities.`**`get_commodity_information`**(*commodity*, *country=None*, *as_json=False*)

This function retrieves fundamental financial information from the specified commodity. The retrieved information from the commodity can be valuable as it is additional information that can be used combined with OHLC values, so to determine financial insights from the company which holds the specified commodity.

**Parameters**

- **commodity** (str) – name of the commodity to retrieve information from.
- **country** (str, optional) – name of the country to retrieve the commodity information from (if there is more than one country that provides data from the same commodity).
- **as_json** (bool, optional) – optional argument to determine the format of the output data (dict or json).

**Returns**

The resulting pandas.DataFrame contains the information fields retrieved from Investing.com from the specified commodity; it can also be returned as a dict, if argument *as_json=True*.

If any of the information fields could not be retrieved, that field/s will be filled with None values. If the retrieval process succeeded, the resulting dict will look like:

```
commodity_information = {
    "1-Year Change": "16.15%",
    "52 wk Range": "1,270.2 - 1,566.2",
    "Base Symbol": "GC",
    "Commodity Name": "Gold",
    "Contract Size": "100 Troy Ounces",
    "Last Rollover Day": "24/11/2019",
    "Month": "Feb 20",
    "Months": "GJMQVZ",
    "Open": 1479.8,
    "Point Value": "$100",
    "Prev. Close": 1481.2,
    "Settlement Day": "25/01/2020",
    "Settlement Type": "Physical",
    "Tick Size": 0.1,
    "Tick Value": 10.0,
    "Day's Range": "1,477.55 - 1,484.25"
}
```

**Return type** pandas.DataFrame or dict- commodity_information

**Raises**

- **ValueError** – raised if any of the introduced arguments is not valid or errored.
- **FileNotFoundError** – raised if *commodities.csv* file was not found or errored.
- **IOError** – raised if commodities.csv file is empty or errored.
- **RuntimeError** – raised if scraping process failed while running.
- **ConnectionError** – raised if the connection to Investing.com errored (did not return HTTP 200)

investpy.commodities.**get_commodity_recent_data**(*commodity*, *country=None*, *as_json=False*, *order='ascending'*, *interval='Daily'*)

This function retrieves recent historical data from the introduced commodity from Investing.com, which will be returned as a pandas.DataFrame if the parameters are valid and the request to Investing.com succeeds. Note that additionally some optional parameters can be specified: as_json and order, which let the user decide if the data is going to be returned as a json or not, and if the historical data is going to be ordered ascending or descending (where the index is the date), respectively.

**Parameters**

- **commodity** (str) – name of the commodity to retrieve recent data from.

- **country** (str, optional) – name of the country to retrieve the commodity data from (if there is more than one country that provides data from the same commodity).
- **as_json** (bool, optional) – to determine the format of the output data, either a pandas.DataFrame if False and a json if True.
- **order** (str, optional) – to define the order of the retrieved data which can either be ascending or descending.
- **interval** (str, optional) – value to define the historical data interval to retrieve, by default *Daily*, but it can also be *Weekly* or *Monthly*.

**Returns**

The function can return either a pandas.DataFrame or a json object, containing the retrieved recent data of the specified commodity. So on, the resulting dataframe contains the open, high, low and close values for the selected commodity on market days and the currency in which those values are presented.

The returned data is case we use default arguments will look like:

```
Date || Open | High | Low | Close | Volume | Currency
-----||------|------|-----|-------|--------|----------
xxxx || xxxx | xxxx | xxx | xxxxx | xxxxxx | xxxxxxxx
```

but in case that as_json parameter was defined as True, then the output will be:

```
{
    name: name,
    recent: [
        {
            date: 'dd/mm/yyyy',
            open: x,
            high: x,
            low: x,
            close: x,
            volume: x,
            currency: x
        },
        ...
    ]
}
```

**Return type** pandas.DataFrame or json

**Raises**

- **ValueError** – raised whenever any of the introduced arguments is not valid or errored.
- **IOError** – raised if commodities object/file was not found or unable to retrieve.
- **RuntimeError** – raised if the introduced commodity was not found or did not match any of the existing ones.
- **ConnectionError** – raised if connection to Investing.com could not be established.
- **IndexError** – raised if commodity recent data was unavailable or not found in Investing.com.

### Examples

```
>>> data = investpy.get_commodity_recent_data(commodity='gold')
>>> data.head()
              Open    High     Low   Close  Volume Currency
Date
2019-10-25  1506.4  1520.9  1503.1  1505.3  368743      USD
2019-10-28  1507.4  1510.8  1492.3  1495.8  318126      USD
2019-10-29  1494.3  1497.1  1485.6  1490.7  291980      USD
2019-10-30  1490.5  1499.3  1483.1  1496.7  353638      USD
2019-10-31  1498.8  1516.7  1496.0  1514.8  390013      USD
```

investpy.commodities.**search_commodities**(*by*, *value*)

This function searches commodities by the introduced value for the specified field. This means that this function is going to search if there is a value that matches the introduced one for the specified field which is the *commodities.csv* column name to search in. Available fields to search commodities are 'name', 'full_name' and 'title'.

**Parameters**

- **by** (str) – name of the field to search for, which is the column name which can be: ''name', 'full_name' or 'title'.
- **value** (str) – value of the field to search for, which is the value that is going to be searched.

**Returns** The resulting pandas.DataFrame contains the search results from the given query, which is any match of the specified value in the specified field. If there are no results for the given query, an error will be raised, but otherwise the resulting pandas.DataFrame will contain all the available commodities that match the introduced query.

**Return type** pandas.DataFrame - search_result

**Raises**

- **ValueError** – raised if any of the introduced parameters is not valid or errored.
- **FileNotFoundError** – raised if *commodities.csv* file is missing.
- **IOError** – raised if data could not be retrieved due to file error.
- **RuntimeError** – raised if no results were found for the introduced value in the introduced field.

## 9.8 investpy.certificates

investpy.certificates.**get_certificate_countries**()

This function retrieves all the available countries to retrieve certificates from, as the listed countries are the ones indexed on Investing.com. The purpose of this function is to list the countries which have available certificates according to Investing.com data, since the country parameter is needed when retrieving data from any certificate available.

**Returns**

The resulting list contains all the countries listed on Investing.com with available certificates to retrieve data from.

In the case that the file reading of *certificate_countries.csv* which contains the names of the available countries with certificates was successfully completed, the resulting `list` will look like:

```
countries = ['france', 'germany', 'italy', 'netherlands', 'sweden']
```

**Return type** `list` - countries

**Raises** **`FileNotFoundError`** – raised when *certificate_countries.csv* file was not found.

`investpy.certificates.`**`get_certificate_historical_data`**(*certificate*, *country*, *from_date*, *to_date*, *as_json=False*, *order='ascending'*, *interval='Daily'*)

This function retrieves historical data from the introduced certificate from Investing.com. So on, the historical data of the introduced certificate from the specified country in the specified date range will be retrieved and returned as a `pandas.DataFrame` if the parameters are valid and the request to Investing.com succeeds. Note that additionally some optional parameters can be specified: as_json and order, which let the user decide if the data is going to be returned as a `json` or not, and if the historical data is going to be ordered ascending or descending (where the index is the date), respectively.

**Parameters**

- **`certificate`** (`str`) – name of the certificate to retrieve historical data from.
- **`country`** (`str`) – name of the country from where the certificate is.
- **`from_date`** (`str`) – date formatted as *dd/mm/yyyy*, since when data is going to be retrieved.
- **`to_date`** (`str`) – date formatted as *dd/mm/yyyy*, until when data is going to be retrieved.
- **`as_json`** (`bool`, optional) – to determine the format of the output data, either a `pandas.DataFrame` if False and a `json` if True.
- **`order`** (`str`, optional) – to define the order of the retrieved data which can either be ascending or descending.
- **`interval`** (`str`, optional) – value to define the historical data interval to retrieve, by default *Daily*, but it can also be *Weekly* or *Monthly*.

**Returns**

The function can return either a `pandas.DataFrame` or a `json` object, containing the retrieved historical data of the specified certificate from the specified country. So on, the resulting dataframe contains the OHLC values for the selected certificate on market days.

The returned data is case we use default arguments will look like:

```
Date || Open | High | Low | Close
-----||------|------|-----|-------
xxxx || xxxx | xxxx | xxx | xxxxx
```

but if we define *as_json=True*, then the output will be:

```
{
    name: name,
    historical: [
        {
            date: 'dd/mm/yyyy',
```

(continues on next page)

(continued from previous page)

```
            open: x,
            high: x,
            low: x,
            close: x
        },
        ...
    ]
}
```

**Return type** `pandas.DataFrame` or `json`

**Raises**

- **`ValueError`** – raised whenever any of the introduced arguments is not valid or errored.
- **`IOError`** – raised if certificates object/file was not found or unable to retrieve.
- **`RuntimeError`** – raised if the introduced certificate/country was not found or did not match any of the existing ones.
- **`ConnectionError`** – raised if connection to Investing.com could not be established.
- **`IndexError`** – raised if certificate historical data was unavailable or not found in Investing.com.

### Examples

```
>>> data = investpy.get_certificate_historical_data(certificate='BNP Gold 31Dec99
↪', country='france', from_date='01/01/2010', to_date='01/01/2019')
>>> data.head()
             Open   High    Low  Close
Date
2010-01-04  77.15  77.15  77.15  77.15
2010-01-05  77.40  77.45  77.15  77.45
2010-01-06  78.40  78.40  78.40  78.40
2010-01-07  78.40  78.45  78.35  78.35
2010-01-08  77.95  78.10  77.95  78.10
```

`investpy.certificates.`**`get_certificate_information`**(*certificate*, *country*, *as_json=False*)

This function retrieves fundamental financial information from the specified certificate. The retrieved information from the certificate can be valuable as it is additional information that can be used combined with OHLC values, so to determine financial insights from the company which holds the specified certificate.

**Parameters**

- **`certificate`** (`str`) – name of the certificate to retrieve information from
- **`country`** (`country`) – name of the country from where the certificate is from.
- **`as_json`** (`bool`, optional) – optional argument to determine the format of the output data (`dict` or `json`).

**Returns**

The resulting `pandas.DataFrame` contains the information fields retrieved from Investing.com from the specified certificate; it can also be returned as a `dict`, if argument *as_json=True*.

If any of the information fields could not be retrieved, that field/s will be filled with None values. If the retrieval process succeeded, the resulting `dict` will look like:

```
certificate_information = {
    "Certificate Name": "XXXX",
    "Certificate Country": "XXXX",
    "Prev. Close": X.Y,
    "Todays Range": "X.Y - X.Y",
    "Leverage": "X:Y",
    "Open": X.Y,
    "52 wk Range": "X.Y - X.Y",
    "Strike Price": "XXXX",
    "Volume": X.Y,
    "Issue Date": "XXXX",
    "Issue Amount": "XXXX",
    "Average Vol. (3m)": X.Y,
    "Maturity Date": "dd/mm/yyyy",
    "1-Year Change": "X.Y%",
    "Asset Class": "XXXX"
}
```

**Return type** pandas.DataFrame or dict- certificate_information

investpy.certificates.**get_certificate_recent_data**(*certificate*, *country*, *as_json=False*, *order='ascending'*, *interval='Daily'*)

This function retrieves recent historical data from the introduced certificate from Investing.com. So on, the recent data of the introduced certificate from the specified country will be retrieved and returned as a pandas.DataFrame if the parameters are valid and the request to Investing.com succeeds. Note that additionally some optional parameters can be specified: as_json and order, which let the user decide if the data is going to be returned as a json or not, and if the historical data is going to be ordered ascending or descending (where the index is the date), respectively.

**Parameters**

- **certificate** (str) – name of the certificate to retrieve recent data from.
- **country** (str) – name of the country from where the certificate is.
- **as_json** (bool, optional) – to determine the format of the output data, either a pandas.DataFrame if False and a json if True.
- **order** (str, optional) – to define the order of the retrieved data which can either be ascending or descending.
- **interval** (str, optional) – value to define the historical data interval to retrieve, by default *Daily*, but it can also be *Weekly* or *Monthly*.

**Returns**

The function returns either a pandas.DataFrame or a json file containing the retrieved recent data from the specified certificate via argument. The dataset contains the OHLC values of the certificate.

The returned data is case we use default arguments will look like:

```
Date || Open | High | Low | Close
-----||------|------|-----|-------
xxxx || xxxx | xxxx | xxx | xxxxx
```

but if we define *as_json=True*, then the output will be:

```
{
    name: name,
    recent: [
        {
            date: dd/mm/yyyy,
            open: x,
            high: x,
            low: x,
            close: x
        },
        ...
    ]
}
```

**Return type** `pandas.DataFrame` or `json`

**Raises**

- **`ValueError`** – raised if there was an argument error.
- **`IOError`** – raised if certificates object/file was not found or unable to retrieve.
- **`RuntimeError`** – raised if the introduced certificate does not match any of the indexed ones.
- **`ConnectionError`** – raised if GET requests does not return 200 status code.
- **`IndexError`** – raised if certificate information was unavailable or not found.

### Examples

```
>>> data = investpy.get_certificate_recent_data(certificate='BNP Gold 31Dec99',␣
↪country='france')
>>> data.head()
              Open   High     Low   Close
Date
2020-07-09  146.4  146.8  145.95  145.95
2020-07-10  146.2  146.2  145.55  145.55
2020-07-13  145.6  145.6  145.45  145.45
2020-07-14  145.4  145.4  145.25  145.25
2020-07-15  144.9  145.1  144.70  144.95
```

`investpy.certificates.`**`get_certificates`**(*country=None*)

This function retrieves all the data stored in *certificates.csv* file, which previously was retrieved from Investing.com. Since the resulting object is a matrix of data, the certificate's data is properly structured in rows and columns, where columns are the certificate data attribute names. Additionally, country filtering can be specified, which will make this function return not all the stored certificates, but just the data of the certificates from the introduced country.

**Parameters** **`country`** (`str`, optional) – name of the country to retrieve all its available certificates from.

**Returns**

The resulting `pandas.DataFrame` contains all the certificate's data from the introduced country if specified, or from every country if None was specified, as indexed in Investing.com from the information previously retrieved by investpy and stored on a csv file.

So on, the resulting `pandas.DataFrame` will look like:

```
country | name | full_name | symbol | issuer | isin | asset_class |␣
↪underlying
--------|------|-----------|--------|--------|------|-------------|---
↪---------
xxxxxxx | xxxx | xxxxxxxxx | xxxxxx | xxxxxx | xxxx | xxxxxxxxxxx |␣
↪xxxxxxxxxx
```

**Return type** `pandas.DataFrame` - certificates_df

**Raises**

- **`ValueError`** – raised whenever any of the introduced arguments is not valid.
- **`FileNotFoundError`** – raised if *certificates.csv* file was not found.
- **`IOError`** – raised when *certificates.csv* file is missing or empty.

`investpy.certificates.`**`get_certificates_dict`**(*country=None*, *columns=None*, *as_json=False*)

This function retrieves all the available certificates indexed on Investing.com, stored on *certificates.csv*. This function also allows the user to specify which country do they want to retrieve data from, or from every listed country; the columns which the user wants to be included on the resulting `dict`; and the output of the function will either be a `dict` or a `json`.

**Parameters**

- **`country`** (`str`, optional) – name of the country to retrieve all its available certificates from.
- **`columns`** (`list`, optional) – names of the columns of the certificate data to retrieve <country, name, full_name, symbol, issuer, isin, asset_class, underlying>
- **`as_json`** (`bool`, optional) – value to determine the format of the output data which can either be a `dict` or a `json`.

**Returns**

The resulting `dict` contains the retrieved data if found, if not, the corresponding fields are filled with *None* values.

In case the information was successfully retrieved, the `dict` will look like:

```
certificates_dict = {
    "country": "france",
    "name": "SOCIETE GENERALE CAC 40 X10 31DEC99",
    "full_name": "SOCIETE GENERALE EFFEKTEN GMBH ZT CAC 40 X10␣
↪LEVERAGE 31DEC99",
    "symbol": "FR0011214527",
    "issuer": "Societe Generale Effekten GMBH",
    "isin": "FR0011214527",
    "asset_class": "index",
    "underlying": "CAC 40 Leverage x10 NR"
}
```

**Return type** `dict` or `json` - certificates_dict

**Raises**

- **`ValueError`** – raised whenever any of the introduced arguments is not valid.
- **`FileNotFoundError`** – raised if *certificates.csv* file was not found.
- **`IOError`** – raised when *certificates.csv* file is missing or empty.

investpy.certificates.**get_certificates_list**(*country=None*)

This function retrieves all the available certificates indexed on Investing.com, already stored on *certificates.csv*. This function also allows the users to specify which country do they want to retrieve data from or if they want to retrieve it from every listed country; so on, a listing of certificates will be returned. This function helps the user to get to know which certificates are available on Investing.com.

**Parameters** **country** (str, optional) – name of the country to retrieve all its available certificates from.

**Returns**

The resulting `list` contains the retrieved data from the *certificates.csv* file, which is a listing of the names of the certificates listed on Investing.com, which is the input for data retrieval functions as the name of the certificate to retrieve data from needs to be specified.

In case the listing was successfully retrieved, the `list` will look like:

```
certificates_list = ['SOCIETE GENERALE CAC 40 X10 31DEC99', 'SG ZT␣
↪CAC 40 x7 Short 31Dec99', ...]
```

**Return type** `list` - certificates_list

**Raises**

- **ValueError** – raised whenever any of the introduced arguments is not valid.
- **FileNotFoundError** – raised if *certificates.csv* file was not found.
- **IOError** – raised when *certificates.csv* file is missing or empty.

investpy.certificates.**get_certificates_overview**(*country*, *as_json=False*, *n_results=100*)

This function retrieves an overview containing all the real time data available for the main certificates from a country, such as the names, symbols, current value, etc. as indexed in Investing.com. So on, the main usage of this function is to get an overview on the main certificates from a country, so to get a general view. Note that since this function is retrieving a lot of information at once, by default just the overview of the Top 100 certificates is being retrieved, but an additional parameter called n_results can be specified so to retrieve N results.

**Parameters**

- **country** (str) – name of the country to retrieve the certificates overview from.
- **as_json** (bool, optional) – optional argument to determine the format of the output data (`pandas.DataFrame` or `json`).
- **n_results** (int, optional) – number of results to be displayed on the overview table (0-1000).

**Returns**

The resulting `pandas.DataFrame` contains all the data available in Investing.com of the main certificates from a country in order to get an overview of it.

If the retrieval process succeeded, the resulting `pandas.DataFrame` should look like:

```
country | name | symbol | last | change_percentage | turnover
--------|------|--------|------|-------------------|----------
xxxxxxx | xxxx | xxxxxx | xxxx | xxxxxxxxxxxxxxxxx | xxxxxxxx
```

**Return type** `pandas.DataFrame` - certificates_overview

**Raises**

- **ValueError** – raised if any of the introduced arguments is not valid or errored.
- **FileNotFoundError** – raised when *certificates.csv* file is missing.
- **IOError** – raised if data could not be retrieved due to file error.
- **RuntimeError** – raised either if the introduced country does not match any of the listed ones or if no overview results could be retrieved from Investing.com.
- **ConnectionError** – raised if GET requests does not return 200 status code.

investpy.certificates.**search_certificates**(*by*, *value*)

This function searches certificates by the introduced value for the specified field. This means that this function is going to search if there is a value that matches the introduced one for the specified field which is the *certificates.csv* column name to search in. Available fields to search certificates are *country*, *name*, *full_name*, *symbol*, *issuer*, *isin*, *asset_class*, *underlying*.

**Parameters**

- **by** (str) – name of the field to search for, which is the column name which can be: country, name, full_name, symbol, issuer, isin, asset_class or underlying.
- **value** (str) – value of the field to search for, which is the value that is going to be searched.

**Returns** The resulting pandas.DataFrame contains the search results from the given query, which is any match of the specified value in the specified field. If there are no results for the given query, an error will be raised, but otherwise the resulting pandas.DataFrame will contain all the available certificates that match the introduced query.

**Return type** pandas.DataFrame - search_result

**Raises**

- **ValueError** – raised if any of the introduced parameters is not valid or errored.
- **FileNotFoundError** – raised if *certificates.csv* file is missing.
- **IOError** – raised if data could not be retrieved due to file error.
- **RuntimeError** – raised if no results were found for the introduced value in the introduced field.

## 9.9 investpy.crypto

investpy.crypto.**get_crypto_historical_data**(*crypto*, *from_date*, *to_date*, *as_json=False*, *order='ascending'*, *interval='Daily'*)

This function retrieves historical data from the introduced crypto from Investing.com. So on, the historical data of the introduced crypto will be retrieved and returned as a pandas.DataFrame if the parameters are valid and the request to Investing.com succeeds. Note that additionally some optional parameters can be specified: as_json and order, which let the user decide if the data is going to be returned as a json or not, and if the historical data is going to be ordered ascending or descending (where the index is the date), respectively.

**Parameters**

- **crypto** (str) – name of the crypto currency to retrieve data from.
- **from_date** (str) – date formatted as *dd/mm/yyyy*, since when data is going to be retrieved.
- **to_date** (str) – date formatted as *dd/mm/yyyy*, until when data is going to be retrieved.

- **as_json** (bool, optional) – to determine the format of the output data, either a pandas.DataFrame if False and a json if True.
- **order** (str, optional) – to define the order of the retrieved data which can either be ascending or descending.
- **interval** (str, optional) – value to define the historical data interval to retrieve, by default *Daily*, but it can also be *Weekly* or *Monthly*.

**Returns**

The function can return either a pandas.DataFrame or a json object, containing the retrieved historical data of the specified crypto currency. So on, the resulting dataframe contains the open, high, low, close and volume values for the selected crypto on market days and the currency in which those values are presented.

The returned data is case we use default arguments will look like:

```
Date || Open | High | Low | Close | Volume | Currency
-----||------|------|-----|-------|--------|----------
xxxx || xxxx | xxxx | xxx | xxxxx | xxxxxx | xxxxxxxx
```

but if we define *as_json=True*, then the output will be:

```
{
    name: name,
    historical: [
        {
            date: 'dd/mm/yyyy',
            open: x,
            high: x,
            low: x,
            close: x,
            volume: x,
            currency: x
        },
        ...
    ]
}
```

**Return type** pandas.DataFrame or json

**Raises**

- **ValueError** – raised whenever any of the introduced arguments is not valid or errored.
- **IOError** – raised if cryptos object/file was not found or unable to retrieve.
- **RuntimeError** – raised if the introduced crypto currency name was not found or did not match any of the existing ones.
- **ConnectionError** – raised if connection to Investing.com could not be established.
- **IndexError** – raised if crypto historical data was unavailable or not found in Investing.com.

### Examples

```
>>> data = investpy.get_crypto_historical_data(crypto='bitcoin', from_date='01/01/
↪2018', to_date='01/01/2019')
>>> data.head()
               Open     High      Low    Close  Volume Currency
Date
2018-01-01  13850.5  13921.5  12877.7  13444.9   78425      USD
2018-01-02  13444.9  15306.1  12934.2  14754.1  137732      USD
2018-01-03  14754.1  15435.0  14579.7  15156.6  106543      USD
2018-01-04  15156.5  15408.7  14244.7  15180.1  110969      USD
2018-01-05  15180.1  17126.9  14832.4  16954.8  141960      USD
```

investpy.crypto.**get_crypto_information**(*crypto*, *as_json=False*)

This function retrieves fundamental financial information from the specified crypto currency. The retrieved information from the crypto currency can be valuable as it is additional information that can be used combined with OHLC values, so to determine financial insights from the company which holds the specified crypto currency.

**Parameters**

- **currency_cross** (str) – name of the currency_cross to retrieve recent historical data from.
- **as_json** (bool, optional) – optional argument to determine the format of the output data (dict or json).

**Returns**

The resulting pandas.DataFrame contains the information fields retrieved from Investing.com from the specified crypto currency; it can also be returned as a dict, if argument *as_json=True*.

If any of the information fields could not be retrieved, that field/s will be filled with None values. If the retrieval process succeeded, the resulting dict will look like:

```
crypto_information = {
    'Chg (7D)': '-4.63%',
    'Circulating Supply': ' BTC18.10M',
    'Crypto Currency': 'Bitcoin',
    'Currency': 'USD',
    'Market Cap': '$129.01B',
    'Max Supply': 'BTC21.00M',
    'Todays Range': '7,057.8 - 7,153.1',
    'Vol (24H)': '$17.57B'
}
```

**Return type** pandas.DataFrame or dict- crypto_information

**Raises**

- **ValueError** – raised if any of the introduced arguments is not valid or errored.
- **FileNotFoundError** – raised if *cryptos.csv* file was not found or errored.
- **IOError** – raised if *cryptos.csv* file is empty or errored.
- **RuntimeError** – raised if scraping process failed while running.
- **ConnectionError** – raised if the connection to Investing.com errored (did not return HTTP 200)

investpy.crypto.**get_crypto_recent_data**(*crypto*, *as_json=False*, *order='ascending'*, *interval='Daily'*)

This function retrieves recent historical data from the introduced crypto from Investing.com. So on, the recent data of the introduced crypto will be retrieved and returned as a `pandas.DataFrame` if the parameters are valid and the request to Investing.com succeeds. Note that additionally some optional parameters can be specified: as_json and order, which let the user decide if the data is going to be returned as a `json` or not, and if the historical data is going to be ordered ascending or descending (where the index is the date), respectively.

**Parameters**

- **crypto** (`str`) – name of the crypto currency to retrieve data from.
- **as_json** (`bool`, optional) – to determine the format of the output data, either a `pandas.DataFrame` if False and a `json` if True.
- **order** (`str`, optional) – to define the order of the retrieved data which can either be ascending or descending.
- **interval** (`str`, optional) – value to define the historical data interval to retrieve, by default *Daily*, but it can also be *Weekly* or *Monthly*.

**Returns**

The function can return either a `pandas.DataFrame` or a `json` object, containing the retrieved recent data of the specified crypto currency. So on, the resulting dataframe contains the open, high, low, close and volume values for the selected crypto on market days and the currency in which those values are presented.

The resulting recent data, in case that the default parameters were applied, will look like:

```
Date || Open | High | Low | Close | Volume | Currency
-----||------|------|-----|-------|--------|----------
xxxx || xxxx | xxxx | xxx | xxxxx | xxxxxx | xxxxxxxx
```

but in case that as_json parameter was defined as True, then the output will be:

```
{
    name: name,
    recent: [
        {
            date: 'dd/mm/yyyy',
            open: x,
            high: x,
            low: x,
            close: x,
            volume: x,
            currency: x
        },
        ...
    ]
}
```

**Return type** `pandas.DataFrame` or `json`

**Raises**

- **ValueError** – raised whenever any of the introduced arguments is not valid or errored.
- **IOError** – raised if cryptos object/file was not found or unable to retrieve.
- **RuntimeError** – raised if the introduced crypto name was not found or did not match any of the existing ones.

- **ConnectionError** – raised if connection to Investing.com could not be established.
- **IndexError** – raised if crypto recent data was unavailable or not found in Investing.com.

### Examples

```
>>> data = investpy.get_crypto_recent_data(crypto='bitcoin')
>>> data.head()
              Open     High     Low   Close   Volume Currency
Date
2019-10-25  7422.8   8697.7  7404.9  8658.3  1177632      USD
2019-10-26  8658.4  10540.0  8061.8  9230.6  1784005      USD
2019-10-27  9230.6   9773.2  9081.0  9529.6  1155038      USD
2019-10-28  9530.1   9866.9  9202.5  9207.2  1039295      USD
2019-10-29  9206.5   9531.3  9125.3  9411.3   918477      USD
```

investpy.crypto.**get_cryptos**()

This function retrieves all the crypto data stored in *cryptos.csv* file, which previously was retrieved from Investing.com. Since the resulting object is a matrix of data, the crypto data is properly structured in rows and columns, where columns are the crypto data attribute names.

Note that just some cryptos are available for retrieval, since Investing.com does not provide information from all the available ones, just the main ones.

**Returns**

The resulting `pandas.DataFrame` contains all the crypto data from every available crypto coin as indexed in Investing.com from the information previously retrieved by investpy and stored on a csv file.

So on, the resulting `pandas.DataFrame` will look like:

```
name | symbol | currency
-----|--------|----------
xxxx | xxxxxx | xxxxxxxx
```

**Return type** `pandas.DataFrame` - cryptos_df

**Raises**

- **FileNotFoundError** – raised if *cryptos.csv* file was not found.
- **IOError** – raised when *cryptos.csv* file is missing or empty.

investpy.crypto.**get_cryptos_dict**(*columns=None*, *as_json=False*)

This function retrieves all the crypto information stored in the *cryptos.csv* file and formats it as a Python dictionary which contains the same information as the file, but every row is a `dict` and all of them are contained in a `list`. Note that the dictionary structure is the same one as the JSON structure. Some optional paramaters can be specified such as the columns or as_json, which are the column names that want to be retrieved in case of needing just some columns to avoid unnecessary information load, and whether the information wants to be returned as a JSON object or as a dictionary; respectively.

Note that just some cryptos are available for retrieval, since Investing.com does not provide information from all the available ones, just the main ones.

**Parameters**

- **columns** (`list`, optional) – column names of the crypto data to retrieve, can be: <name, currency, symbol>

- **as_json** (bool, optional) – if True the returned data will be a json object, if False, a list of dict.

**Returns**

The resulting list of dict contains the retrieved data from every crypto coin as indexed in Investing.com from the information previously retrieved by investpy and stored on a csv file.

In case the information was successfully retrieved, the list of dict will look like:

```
cryptos_dict = {
    'name': name,
    'currency': currency,
    'symbol': symbol,
}
```

**Return type** list of dict OR json - cryptos_dict

**Raises**

- **ValueError** – raised whenever any of the introduced arguments is not valid.
- **FileNotFoundError** – raised if *cryptos.csv* file was not found.
- **IOError** – raised when *cryptos.csv* file is missing or empty.

investpy.crypto.**get_cryptos_list**()

This function retrieves all the crypto coin names stored in *cryptos.csv* file, which contains all the data from the crypto coins as previously retrieved from Investing.com. So on, this function will just return the crypto coin names which will be the main input parameters when it comes to crypto data retrieval functions from investpy.

Note that just some cryptos are available for retrieval, since Investing.com does not provide information from all the available ones, just the main ones.

**Returns**

The resulting list contains the all the available crypto coin names as indexed in Investing.com from the information previously retrieved by investpy and stored on a csv file.

In case the information was successfully retrieved, the list of crypto coin names will look like:

```
cryptos_list = ['Bitcoin', 'Ethereum', 'XRP', 'Bitcoin Cash', 'Tether
↪', 'Litecoin', ...]
```

**Return type** list - cryptos_list

**Raises**

- **FileNotFoundError** – raised if *cryptos.csv* file was not found.
- **IOError** – raised when *cryptos.csv* file is missing or empty.

investpy.crypto.**get_cryptos_overview**(*as_json=False*, *n_results=100*)

This function retrieves an overview containing all the real time data available for the main crypto currencies, such as the names, symbols, current value, etc. as indexed in Investing.com. So on, the main usage of this function is to get an overview on the main crypto currencies, so to get a general view. Note that since this function is retrieving a lot of information at once, by default just the overview of the Top 100 crypto currencies is being retrieved, but an additional parameter called n_results can be specified so to retrieve N results.

**Parameters**

- **as_json** (bool, optional) – optional argument to determine the format of the output data (pandas.DataFrame or json).

- **n_results** (int, optional) – number of results to be displayed on the overview table (0-all_cryptos), where all crypto currencies will be retrieved if n_results=None.

---

**Note:** The amount of indexed crypto currencies may vary, so if n_results is set to *None*, all the available crypto currencies in Investing.com while retrieving the overview, will be retrieved and returned.

---

**Returns**

The resulting pandas.DataFrame contains all the data available in Investing.com of the main crypto currencies in order to get an overview of it.

If the retrieval process succeeded, the resulting pandas.DataFrame should look like:

```
name | symbol | price | market_cap | volume24h | total_volume |␣
↪change24h | change7d | currency
-----|--------|-------|------------|-----------|--------------|-------
↪----|----------|----------
xxxx | xxxxxx | xxxxx | xxxxxxxxxx | xxxxxxxxx | xxxxxxxxxxxx |␣
↪xxxxxxxxx | xxxxxxxx | xxxxxxxx
```

**Return type** pandas.DataFrame - cryptos_overview

**Raises**

- **ValueError** – raised if any of the introduced arguments is not valid or errored.
- **IOError** – raised if data could not be retrieved due to file error.
- **RuntimeError** – raised it no overview results could be retrieved from Investing.com.
- **ConnectionError** – raised if GET requests does not return 200 status code.

investpy.crypto.**search_cryptos**(*by*, *value*)

This function searches cryptos by the introduced value for the specified field. This means that this function is going to search if there is a value that matches the introduced one for the specified field which is the *cryptos.csv* column name to search in. Available fields to search cryptos are 'name' and 'symbol'.

**Parameters**

- **by** (str) – name of the field to search for, which is the column name which can be: 'name' or 'symbol'.
- **value** (str) – value of the field to search for, which is the value that is going to be searched.

**Returns** The resulting pandas.DataFrame contains the search results from the given query, which is any match of the specified value in the specified field. If there are no results for the given query, an error will be raised, but otherwise the resulting pandas.DataFrame will contain all the available cryptos that match the introduced query.

**Return type** pandas.DataFrame - search_result

**Raises**

- **ValueError** – raised if any of the introduced parameters is not valid or errored.
- **FileNotFoundError** – raised if *cryptos.csv* file is missing.
- **IOError** – raised if data could not be retrieved due to file error.
- **RuntimeError** – raised if no results were found for the introduced value in the introduced field.

## 9.10 investpy.news

investpy.news.**economic_calendar**(*time_zone=None*, *time_filter='time_only'*, *countries=None*, *importances=None*, *categories=None*, *from_date=None*, *to_date=None*)

This function retrieves the economic calendar, which covers financial events and indicators from all over the world updated in real-time. By default, the economic calendar of the currrent day from you local timezone will be retrieved, but note that some parameters can be specified so that the economic calendar to retrieve can be filtered.

**Parameters**

- **time_zone** (str, optional) – time zone in GMT +/- hours:minutes format, which will be the reference time, if None, the local GMT time zone will be used.
- **time_filter** (str, optional) – it can be *time_only* or *time_remain*, so that the calendar will display the time when the event will occurr according to the time zone or the remaining time until an event occurs.
- **countries** (list of str, optional) – list of countries from where the events of the economic calendar will be retrieved, all contries will be taken into consideration if this parameter is None.
- **importances** (list of str, optional) – list of importances of the events to be taken into consideration, can contain: high, medium and low; if None all the importance ratings will be taken into consideration including holidays.
- **categories** (list of str, optional) – list of categories to which the events will be related to, if None all the available categories will be taken into consideration.
- **from_date** (str, optional) – date from when the economic calendar will be retrieved in dd/mm/yyyy format, if None just current day's economic calendar will be retrieved.
- **to_date** (str, optional) – date until when the economic calendar will be retrieved in dd/mm/yyyy format, if None just current day's economic calendar will be retrieved.

**Returns** The resulting pandas.DataFrame will contain the retrieved information from the economic calendar with the specified parameters which will include information such as: date, time, zone or country of the event, event's title, etc. Note that some of the retrieved fields may be None since Investing.com does not provides that information.

**Return type** pandas.DataFrame - economic_calendar

**Raises** **ValueError** – raised if any of the introduced parameters is not valid or errored.

### Examples

```
>>> data = investpy.economic_calendar()
>>> data.head()
    id        date     time         zone currency importance                    ␣
↪   event actual forecast previous
0  323  27/01/2020  All Day    singapore     None       None  Singapore - Chinese␣
↪New Year   None     None     None
1    9  27/01/2020  All Day    hong kong     None       None    Hong Kong - New␣
↪Year's Day   None     None     None
2   71  27/01/2020  All Day    australia     None       None     Australia -␣
↪Australia Day   None     None     None
```

(continues on next page)

(continued from previous page)

```
3  750  27/01/2020  All Day        china     None       None       China - Spring␣
↪Festival   None     None     None
4  304  27/01/2020  All Day  south korea     None       None  South Korea -␣
↪Market Holiday   None     None     None
```

## 9.11 investpy.technical

investpy.technical.**moving_averages**(*name*, *country*, *product_type*, *interval='daily'*)

This function retrieves the moving averages values calculated by Investing.com for every financial product available (stocks, funds, etfs, indices, currency crosses, bonds, certificates and commodities) for different time intervals. So on, the user must provide the product_type name and the name of the product (unless product_type is 'stock' which name value will be the stock's symbol) and the country if required (mandatory unless product_type is either 'currency_cross' or 'commodity', where it must be None). Additionally, the interval can be specified which defines the update frequency of the calculations of the moving averages (both simple and exponential). Note that the specified interval is not the moving average's interval, since all the available time frames used on the calculation of the moving averages are retrieved.

**Parameters**

- **name** (str) – name of the product to retrieve the moving averages table from (if product_type is *stock*, its value must be the stock's symbol not the name).
- **country** (str) – country name of the introduced product if applicable (if product_type is either *currency_cross* or *commodity* this parameter should be None, unless it can be specified just for *commodity* product_type).
- **product_type** (str) – identifier of the introduced product, available ones are: *stock*, *fund*, *etf*, *index*, *currency_cross*, *bond*, *certificate* and *commodity*.
- **interval** (str) – time interval of the resulting calculations, available values are: *5mins*, *15mins*, *30mins*, *1hour*, *5hours*, *daily*, *weekly* and *monthly*.

**Returns**

The resulting pandas.DataFrame contains the table with the results of the calculation of the moving averages made by Investing.com for the introduced financial product. So on, if the retrieval process succeed its result will look like:

```
 period | sma_value | sma_signal | ema_value | ema_signal
--------|-----------|------------|-----------|------------
 xxxxxx | xxxxxxxxx | xxxxxxxxxx | xxxxxxxxx | xxxxxxxxxx
```

**Return type** pandas.DataFrame - moving_averages

**Raises**

- **ValueError** – raised if any of the introduced parameters is not valid or errored.
- **ConnectionError** – raised if the connection to Investing.com errored or could not be established.

### Examples

```
>>> data = investpy.moving_averages(name='bbva', country='spain', product_type=
↪'stock', interval='daily')
>>> data.head()
  period  sma_value sma_signal  ema_value ema_signal
0      5      4.615        buy      4.650        buy
1     10      4.675       sell      4.693       sell
2     20      4.817       sell      4.763       sell
3     50      4.859       sell      4.825       sell
4    100      4.809       sell      4.830       sell
5    200      4.822       sell      4.867       sell
```

investpy.technical.**pivot_points**(*name*, *country*, *product_type*, *interval='daily'*)

This function retrieves the pivot points values calculated by Investing.com for every financial product available (stocks, funds, etfs, indices, currency crosses, bonds, certificates and commodities) for different time intervals. Pivot points are calculated on different levels: three support levels (S) and three resistance ones (R). So on, the user must provide the product_type name and the name of the product (unless product_type is 'stock' which name value will be the stock's symbol) and the country if required (mandatory unless product_type is either 'currency_cross' or 'commodity', where it must be None). Additionally, the interval can be specified which defines the update frequency of the calculations of the technical indicators (mainly momentum indicators).

**Parameters**

- **name** (str) – name of the product to retrieve the technical indicators table from (if product_type is *stock*, its value must be the stock's symbol not the name).
- **country** (str) – country name of the introduced product if applicable (if product_type is either *currency_cross* or *commodity* this parameter should be None, unless it can be specified just for *commodity* product_type).
- **product_type** (str) – identifier of the introduced product, available ones are: *stock*, *fund*, *etf*, *index*, *currency_cross*, *bond*, *certificate* and *commodity*.
- **interval** (str) – time interval of the resulting calculations, available values are: *5mins*, *15mins*, *30mins*, *1hour*, *5hours*, *daily*, *weekly* and *monthly*.

**Returns**

The resulting pandas.DataFrame contains the table with the results of the calculation of the pivot points made by Investing.com for the introduced financial product. So on, if the retrieval process succeed its result will look like:

```
 name | s3 | s2 | s1 | pivot_points | r1 | r2 | r3
------|----|----|----|--------------|----|----|----
 xxxx | xx | xx | xx | xxxxxxxxxxx | xx | xx | xx
```

**Return type** pandas.DataFrame - pivot_points

**Raises**

- **ValueError** – raised if any of the introduced parameters is not valid or errored.
- **ConnectionError** – raised if the connection to Investing.com errored or could not be established.

### Examples

```
>>> data = investpy.pivot_points(name='bbva', country='spain', product_type='stock
↪', interval='daily')
>>> data.head()
        name     s3     s2     s1  pivot_points     r1     r2     r3
0    Classic  4.537  4.573  4.620         4.656  4.703  4.739  4.786
1  Fibonacci  4.573  4.605  4.624         4.656  4.688  4.707  4.739
2  Camarilla  4.645  4.653  4.660         4.656  4.676  4.683  4.691
3   Woodie's  4.543  4.576  4.626         4.659  4.709  4.742  4.792
4   DeMark's    NaN    NaN  4.639         4.665  4.721    NaN    NaN
```

investpy.technical.**technical_indicators**(*name*, *country*, *product_type*, *interval='daily'*)

This function retrieves the technical indicators values calculated by Investing.com for every financial product available (stocks, funds, etfs, indices, currency crosses, bonds, certificates and commodities) for different time intervals. So on, the user must provide the product_type name and the name of the product (unless product_type is 'stock' which name value will be the stock's symbol) and the country if required (mandatory unless product_type is either 'currency_cross' or 'commodity', where it must be None). Additionally, the interval can be specified which defines the update frequency of the calculations of the technical indicators (mainly momentum indicators).

**Parameters**

- **name** (str) – name of the product to retrieve the technical indicators table from (if product_type is *stock*, its value must be the stock's symbol not the name).
- **country** (str) – country name of the introduced product if applicable (if product_type is either *currency_cross* or *commodity* this parameter should be None, unless it can be specified just for *commodity* product_type).
- **product_type** (str) – identifier of the introduced product, available ones are: *stock*, *fund*, *etf*, *index*, *currency_cross*, *bond*, *certificate* and *commodity*.
- **interval** (str) – time interval of the resulting calculations, available values are: *5mins*, *15mins*, *30mins*, *1hour*, *5hours*, *daily*, *weekly* and *monthly*.

**Returns**

The resulting pandas.DataFrame contains the table with the results of the calculation of the technical indicators made by Investing.com for the introduced financial product. So on, if the retrieval process succeed its result will look like:

```
 technical_indicator | value | signal
---------------------|-------|--------
 xxxxxxxxxxxxxxxxxx | xxxxx | xxxxxx
```

**Return type** pandas.DataFrame - technical_indicators

**Raises**

- **ValueError** – raised if any of the introduced parameters is not valid or errored.
- **ConnectionError** – raised if the connection to Investing.com errored or could not be established.

### Examples

```
>>> data = investpy.technical_indicators(name='bbva', country='spain', product_
↪type='stock', interval='daily')
>>> data.head()
    technical_indicator    value           signal
0               RSI(14)  39.1500             sell
1            STOCH(9,6)  33.2340             sell
2          STOCHRSI(14)  67.7390              buy
3           MACD(12,26)  -0.0740             sell
4               ADX(14)  55.1150             sell
5           Williams %R -66.6670             sell
6               CCI(14) -77.1409             sell
7               ATR(14)   0.0939  less_volatility
8        Highs/Lows(14)  -0.0199             sell
9   Ultimate Oscillator  43.0010             sell
10                  ROC  -6.6240             sell
11  Bull/Bear Power(13)  -0.1590             sell
```

## 9.12 investpy.search

investpy.search.**search_events**(*text*, *importances=None*, *countries=None*, *n_results=None*)
TODO

investpy.search.**search_quotes**(*text*, *products=None*, *countries=None*, *n_results=None*)
This function will use the Investing.com search engine so to retrieve the search results of the introduced text. This function will create a `list` of *investpy.utils.search_obj.SearchObj* class instances, unless *n_results* is set to 1, where just a single *investpy.utils.search_obj.SearchObj* will be returned.

Those class instances will contain the search results so that they can be easily accessed and so to ease the data retrieval process since it can be done calling the methods *self.retrieve_recent_data()* or *self.retrieve_historical_data(from_date, to_date)* from each class instance, which will fill the historical data attribute, *self.data*. Also the information of the financial product can be retrieved using the function *self.retrieve_information()*, that will also dump the information in the attribute *self.information*; the technical indicators can be retrieved using *self.retrieve_technical_indicators()* dumped in the attribute *self.technical_indicators*; the default currency using *self.retrieve_currecy()* dumped in the attribute *self.default_currency*.

**Parameters**

- **text** (`str`) – text to search in Investing.com among all its indexed data.
- **products** (`list` of `str`, optional) – list with the product type filter/s to be applied to search result quotes so that they match the filters. Possible products are: *indices*, *stocks*, *etfs*, *funds*, *commodities*, *currencies*, *cryptos*, *bonds*, *certificates* and *fxfutures*, by default this parameter is set to *None* which means that no filter will be applied, and all product type quotes will be retrieved.
- **countries** (`list` of `str`, optional) – list with the country name filter/s to be applied to search result quotes so that they match the filters. Possible countries can be found in the docs, by default this paremeter is set to *None* which means that no filter will be applied, and quotes from every country will be retrieved.
- **n_results** (`int`, optional) – number of search results to retrieve and return.

**Returns** The resulting `list` of *investpy.utils.search_obj.SearchObj* will contained the retrieved financial products matching the introduced text if found, otherwise a RuntimeError

will be raised, so as to let the user know that no results were found for the introduced text. But note that if the n_results value is equal to 1, a single value will be returned, instead of a list of values.

**Return type** list of *investpy.utils.search_obj.SearchObj* or *investpy.utils.search_obj.SearchObj*

**Raises**

- **ValueError** – raised whenever any of the introduced parameter is not valid or errored.
- **ConnectionError** – raised whenever the connection to Investing.com failed.
- **RuntimeError** – raised when there was an error while executing the function.

**class** investpy.utils.search_obj.**SearchObj**(*id_*, *name*, *symbol*, *tag*, *country*, *pair_type*, *exchange*)

Class which contains each search result when searching data in Investing.com.

This class contains the search results of the Investing.com search made with the function call *investpy.search_quotes(text, products, countries, n_results)* which returns a list of instances of this class with the formatted retrieved information. Additionally, data can either be retrieved or not including both recent and historical data, which will be included in the *SearchObj.data* attribute when calling either *SearchObj.retrieve_recent_data()* or *SearchObj.retrieve_historical_data(from_date, to_date)*, respectively.

**id_**
ID value used by Investing.com to retrieve data.

**Type** int

**name**
name of the retrieved financial product.

**Type** str

**symbol**
symbol of the retrieved financial product.

**Type** str

**tag**
tag (which is the Investing.com URL) of the retrieved financial product.

**Type** str

**country**
name of the country from where the retrieved financial product is.

**Type** str

**pair_type**
type of retrieved financial product (stocks, funds, etfs, etc.).

**Type** str

**exchange**
name of the stock exchange of the retrieved financial product.

**Type** str

**Extra Attributes:**

**data (pandas.DataFrame):** recent or historical data to retrieve from the current financial product, generated after calling either self.retrieve_recent_data or self.retrieve_historical_data().

**info (dict):** contains the information of the current financial product, generated after calling the self.retrieve_information() function.

**__eq__**(*other*)
Return self==value.

**__hash__**()
Return hash(self).

**__init__**(*id_*, *name*, *symbol*, *tag*, *country*, *pair_type*, *exchange*)
Initialize self. See help(type(self)) for accurate signature.

**__str__**()
Return str(self).

**__weakref__**
list of weak references to the object (if defined)

**retrieve_currency**()
Class method used to retrieve the default currency from the class instance of any financial product.

This method retrieves the default currency from Investing.com of the financial product of the current class instance. This method uses the data previously retrieved from the *investpy.search_quotes(text, products, countries, n_results)* function search results to build the request that it is going to be sent to Investing.com so to retrieve and parse the information, since the product tag is required.

**Returns** This method retrieves the default currency from the current class instance of a financial product from Investing.com.

**Return type** `str` - default_currency

**Raises**

- **ConnectionError** – raised if connection to Investing.com could not be established.
- **RuntimeError** – raised if there was any problem while retrieving the data from Investing.com.

**retrieve_historical_data**(*from_date*, *to_date*)
Class method used to retrieve the historical data from the class instance of any financial product.

This method retrieves the historical data from Investing.com of the financial product of the current class instance on the specified date range, so it fills the *SearchObj.data* attribute with the retrieved `pandas.DataFrame`. This method uses the previously filled data from the *investpy.search_quotes(text, products, countries, n_results)* function search results to build the request that it is going to be sent to Investing.com so to retrieve and parse the data.

**Parameters**

- **from_date** (`str`) – date from which data will be retrieved, specified in dd/mm/yyyy format.
- **to_date** (`str`) – date until data will be retrieved, specified in dd/mm/yyyy format.

**Returns** This method retrieves the historical data from the current class instance of a financial product from Investing.com. This method both stores retrieved data in self.data attribute of the class instance and it also returns it as a normal function will do.

**Return type** `pandas.DataFrame` - data

**Raises**

- **ValueError** – raised if any of the introduced parameters was not valid or errored.

- **RuntimeError** – raised if there was any error while retrieving the data from Investing.com.

**retrieve_information**()

Class method used to retrieve the information from the class instance of any financial product.

This method retrieves the information from Investing.com of the financial product of the current class instance, so it fills the *SearchObj.info* attribute with the retrieved `dict`. This method uses the previously retrieved data from the *investpy.search_quotes(text, products, countries, n_results)* function search results to build the request that it is going to be sent to Investing.com so to retrieve and parse the information, since the product tag is required.

**Returns** This method retrieves the information from the current class instance of a financial product from Investing.com. This method both stores retrieved information in self.information attribute of the class instance and it also returns it as a normal function will do.

**Return type** `dict` - info

**Raises**

- **ConnectionError** – raised if connection to Investing.com could not be established.
- **RuntimeError** – raised if there was any problem while retrieving the data from Investing.com.

**retrieve_recent_data**()

Class method used to retrieve the recent data from the class instance of any financial product.

This method retrieves the recent data from Investing.com of the financial product of the current class instance, so it fills the *SearchObj.data* attribute with the retrieved `pandas.DataFrame`. This method uses the previously filled data from the *investpy.search_quotes(text, products, countries, n_results)* function search results to build the request that it is going to be sent to Investing.com so to retrieve and parse the data.

**Returns** This method retrieves the recent data from the current class instance of a financial product from Investing.com. This method both stores retrieved data in self.data attribute of the class instance and it also returns it as a normal function will do.

**Return type** `pandas.DataFrame` - data

**retrieve_technical_indicators**(*interval='daily'*)

Class method used to retrieve the technical indicators from the class instance of any financial product.

This method retrieves the technical indicators from Investing.com for the financial product of the current class instance, to later put in into the *SearchObj.technical_indicators* attribute. This method uses the previously retrieved data from the *investpy.search_quotes(text, products, countries, n_results)* function search results to build the request that it is going to be sent to Investing.com so to retrieve and parse the technical indicators, since the product id is required.

**Parameters** **interval** (`str`, optional) – time interval of the technical indicators' calculations, available values are: *5mins*, *15mins*, *30mins*, *1hour*, *5hours*, *daily*, *weekly* and *monthly*. Note that for funds just the intervals: *daily*, *weekly* and *monthly* are available.

**Returns** This method retrieves the technical indicators from the current class instance of a financial product from Investing.com. This method not just stores retrieved technical indicators table into self.technical_indicators but it also returns it as a normal function will do.

**Return type** `pd.DataFrame` - technical_indicators

**Raises**

- **ValueError** – raised if any of the input parameters is not valid.

- **ConnectionError** – raised if connection to Investing.com could not be established.
- **RuntimeError** – raised if there was any problem while retrieving the data from Investing.com.

CHAPTER

TEN

# INDICES AND TABLES

- genindex
- modindex
- search

# PYTHON MODULE INDEX

## i

# INDEX

## Symbols



## C



## E



## G

## I



## M

## N



## P



## R



## S



## T