# ipeadatapy Documentation

***Release***

**Luan Borelli**

**May 18, 2019**

# CONTENTS:

ipeadatapy is a data and metadata manipulation, visualization and extraction package made in Python using Ipeadata database official API. In it's essence it is an API wrapper.

CHAPTER

ONE

# ABOUT IPEADATAPY

## 1.1 Purpose

The main purpose of Ipeadatapy package is to provide a way of extracting data from Ipeadata through Python using Ipeadata's API. So, in this sense, Ipeadatapy is what is called an API wrapper. Nevertheless, the goal of the package is far from being only extract data. Ipeadatapy also is concerned with treating, cleaning and making more understandable the data provided by the API as well as providing data filtering and research mechanisms. Briefly, Ipeadatapy's objective can be described as being to facilitate users to search and analyze time series data and metadata from Ipeadata database using Python.

## 1.2 License

CHAPTER

TWO

# INSTALLATION

## 2.1 PyPI

ipeadatapy can be installed via pip from PyPI.

```
>>> pip install ipeadatapy
```

If you already have it installed, make sure it is up to date by running:

```
>>> pip install --upgrade ipeadatapy
```

## 2.2 Git

The source code is currently hosted on Ipeadatapy's GitHub.

## 2.3 Dependencies

- pandas
- requests

CHAPTER

THREE

# GETTING STARTED

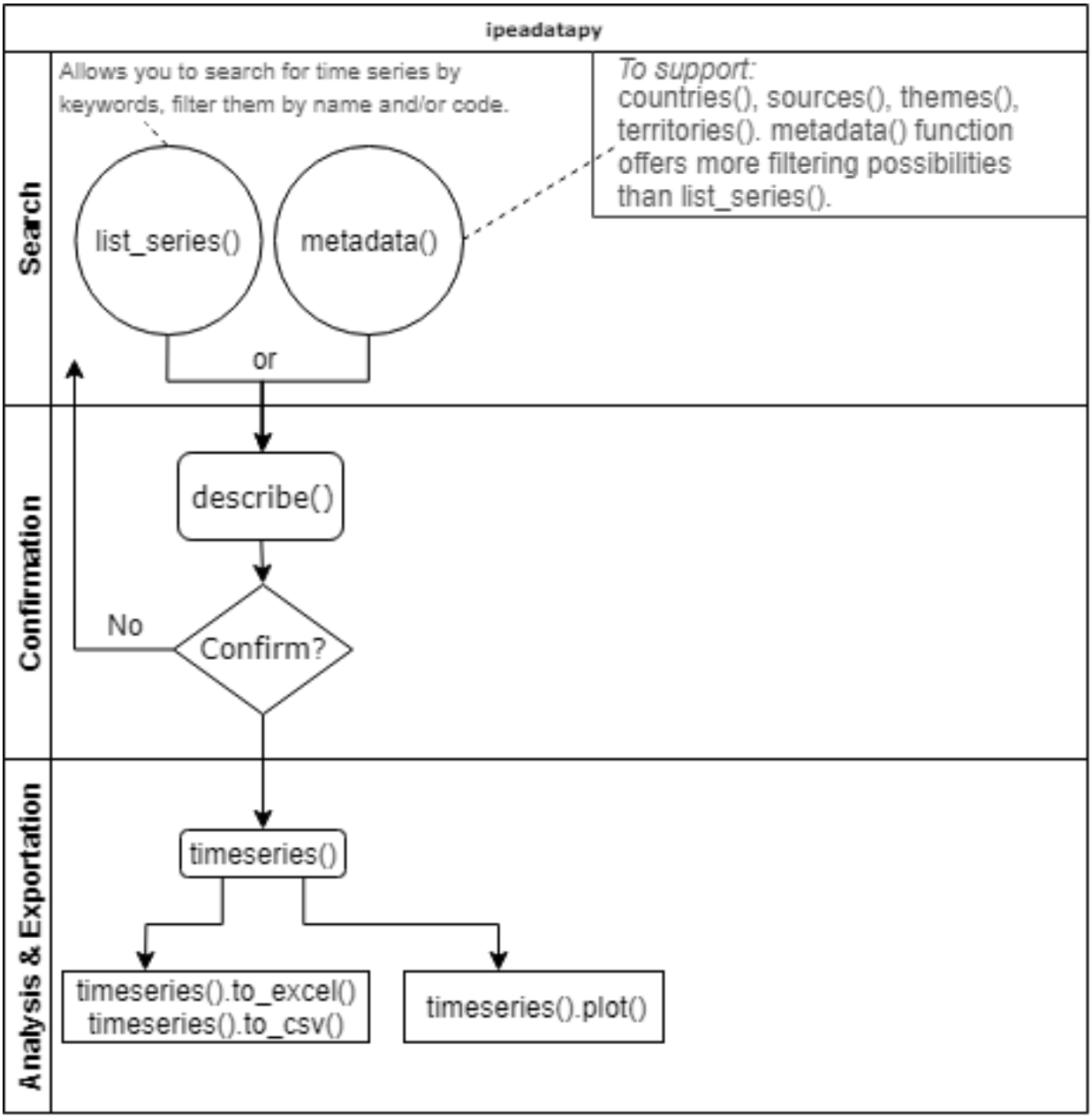
## 3.1 Prerequisites

The only technical prerequisites are the dependencies and, of course, Python itself. The unique knowledge prerequisite is a basic Python language understanding. If you never worked with Python before and for some reason ended up here, it is highly recommended to read Python's official Beginner's Guide to Python before starting with this package.

Although ipeadatapy can be run in any kind of Python environment, since the package is all about data, it is recommended, for a better experience, to work with a notebook style interactive interpreter. The more convenient recommendation is to use Jupyter Notebook.

Jupyter Notebook is a web application for creating Jupyter notebooks. A Jupyter notebook is a JSON document containing an ordered list of input/output cells which can contain code, text, mathematics, plots and rich media. Jupyter notebooks can be converted to a number of open standard output formats (HTML, HTML presentation slides, LaTeX, PDF, ReStructuredText, Markdown, Python) through 'Download As' in the web interface and jupyter convert in a shell.

## 3.2 Usage overview




## 3.3 The basics

Firt of all, you need to import the package:

```
>>> import ipeadatapy
```

### 3.3.1 Finding and analyzing your desired time series

Running this function will show all Ipeadata's time series:

```
>>> ipeadatapy.list_series()
```

If you are looking for series of a specific subject, you can filter the function output for only series containing some keyword. For example, let's filter the function return for only time series containing the word 'BPM6' in their names. This functionality can be used as a searching mechanism.

```
>>> ipeadatapy.list_series('BPM6')
                    CODE                                                NAME
6721              BPAG_AR             Ativos de reserva (Nova metod. - BPM6)
6722              BPAG_BC     Balanca comercial - Saldo (Nova metod. - BPM6)
6723             BPAG_BCM  Balanca comercial - Importacoes (Nova metod. -...
6724             BPAG_BCX  Balanca comercial - Exportacoes (Nova metod. -...
6725              BPAG_CF  Conta Financeira - Saldo (Captacoes - Concesso...
6726              BPAG_CK         Conta capital - Saldo (Nova metod. - BPM6)
6727             BPAG_CKD         Conta capital - Desp. (Nova metod. - BPM6)
6728             BPAG_CKR          Conta capital - Rec. (Nova metod. - BPM6)
6729            BPAG_CRCO  Outros invest. - Cr?d. comerciais e adiantamen...
6730           BPAG_CRCOA  Outros invest. - Cr?d. comerciais e adiantamen...
...                   ...                                                ...
```

You can also use any other keyword you want. Be aware of the case sensitiveness of the keyword.

### 3.3.2 Getting more details

Let's suppose that we found our desired time series. Its code is BPAG_AR. We can use a handy command called `describe()` to confirm the details about this time series.

```
>>> ipeadatapy.describe('BPAG_AR')
                           Ativos de reserva (Nova metod. - BPM6)
Name                       Ativos de reserva (Nova metod. - BPM6)
Code                                                      BPAG_AR
Big Theme                                          Macroeconomico
Theme                                     Balanco de pagamentos
Source          Banco Central do Brasil, Balanco de Pagamentos...
Source acronym                                      Bacen/BP (BPM6)
Comment         Metodologia do Manual de Balanco de Pagamentos...
Last update                          2019-03-14T13:48:00.803-03:00
Frequency                                                  Anual
Measure                                                      US$
Unit                                                    milhoes
Status                                                        A
```

As you can see, this function returns some details about the specified time series: the name of the series, his code, the big theme and theme which this series correspond, his source, the source acronym, the comment, his last update date and time, his frequency, measure, unit and status. Thus, this function is a good way to have an overview of a specific time series.

If you are not satisfied with these information, you can check a more complete metadata data frame about the series by running:

```
>>> ipeadatapy.metadata('BPAG_AR')
        BIG THEME FNTEXTURL       FNTID
↪SOURCE   SOURCE ACRONYM   ...   SERTEMMUN THEME CODE TEMCODIGOPAI
↪THEME MEASURE
```

```
0  Macroecon?mico      None  1333080354  Banco Central do Brasil, Balan?o de␣
↪Pagamentos...  Bacen/BP (BPM6)   ...        None         10         None  Balan?o␣
↪de pagamentos     US$
```

We will be digging deeper into the `metadata()` function in future sections.

### 3.3.3 Observing the data

Now that you are sure about your selected time series, you might be wondering how to observe what really matters: the data. For this purpose, use the function `timeseries()`:

```
>>> ipeadatapy.timeseries('BPAG_AR')
    YEAR  DAY  MONTH     CODE                          DATE   VALUE (US$)
0   1995    1      1  BPAG_AR  1995-01-01T00:00:00-02:00  12918.900000
1   1996    1      1  BPAG_AR  1996-01-01T00:00:00-02:00   8666.100000
2   1997    1      1  BPAG_AR  1997-01-01T00:00:00-02:00  -7907.159127
3   1998    1      1  BPAG_AR  1998-01-01T00:00:00-02:00  -7970.207388
4   1999    1      1  BPAG_AR  1999-01-01T00:00:00-02:00  -7822.039996
5   2000    1      1  BPAG_AR  2000-01-01T00:00:00-02:00  -2261.654351
6   2001    1      1  BPAG_AR  2001-01-01T00:00:00-02:00   3306.600484
7   2002    1      1  BPAG_AR  2002-01-01T00:00:00-02:00    302.087225
8   2003    1      1  BPAG_AR  2003-01-01T00:00:00-02:00   8495.650494
9   2004    1      1  BPAG_AR  2004-01-01T00:00:00-02:00   2244.029835
10  2005    1      1  BPAG_AR  2005-01-01T00:00:00-02:00   4319.463872
11  2006    1      1  BPAG_AR  2006-01-01T00:00:00-02:00  30569.117416
12  2007    1      1  BPAG_AR  2007-01-01T00:00:00-02:00  87484.245682
13  2008    1      1  BPAG_AR  2008-01-01T00:00:00-02:00   2969.072068
14  2009    1      1  BPAG_AR  2009-01-01T00:00:00-02:00  46650.987800
15  2010    1      1  BPAG_AR  2010-01-01T00:00:00-02:00  49100.503587
16  2011    1      1  BPAG_AR  2011-01-01T00:00:00-02:00  58636.807211
17  2012    1      1  BPAG_AR  2012-01-01T00:00:00-02:00  18899.552358
18  2013    1      1  BPAG_AR  2013-01-01T00:00:00-02:00  -5926.487151
19  2014    1      1  BPAG_AR  2014-01-01T00:00:00-02:00  10832.657276
20  2015    1      1  BPAG_AR  2015-01-01T00:00:00-02:00   1568.772099
21  2016    1      1  BPAG_AR  2016-01-01T00:00:00-02:00   9237.436064
22  2017    1      1  BPAG_AR  2017-01-01T00:00:00-02:00   5092.868662
23  2018    1      1  BPAG_AR  2018-01-01T00:00:00-02:00   2927.674626
24  2019    1      1  BPAG_AR  2019-01-01T00:00:00-02:00    813.549854
```

### 3.3.4 Observing intervals of the data

If you just want the data for a specific year you can use the parameter `year`:

```
>>> ipeadatapy.timeseries("GM366_ERC366", year=2019)
       YEAR  DAY  MONTH          CODE                          DATE  VALUE (R$)
12078  2019    2      1  GM366_ERC366  2019-01-02T00:00:00-02:00      3.8589
12079  2019    3      1  GM366_ERC366  2019-01-03T00:00:00-02:00      3.7677
12080  2019    4      1  GM366_ERC366  2019-01-04T00:00:00-02:00      3.7621
12081  2019    7      1  GM366_ERC366  2019-01-07T00:00:00-02:00      3.7056
12082  2019    8      1  GM366_ERC366  2019-01-08T00:00:00-02:00      3.7202
12083  2019    9      1  GM366_ERC366  2019-01-09T00:00:00-02:00      3.6925
12084  2019   10      1  GM366_ERC366  2019-01-10T00:00:00-02:00      3.6863
12085  2019   11      1  GM366_ERC366  2019-01-11T00:00:00-02:00      3.7135
12086  2019   14      1  GM366_ERC366  2019-01-14T00:00:00-02:00      3.7255
12087  2019   15      1  GM366_ERC366  2019-01-15T00:00:00-02:00      3.7043
```

```
...     ...  ...    ...           ...                          ...         ...
[89 rows x 6 columns]
```

If you just want the data for a specific month of a specific year, use both the parameters `year` and `month`:

```
>>> ipeadatapy.timeseries("GM366_ERC366", year=2019, month=4)
       YEAR  DAY  MONTH          CODE                       DATE  VALUE (R$)
12139  2019    1      4  GM366_ERC366  2019-04-01T00:00:00-03:00      3.8676
12140  2019    2      4  GM366_ERC366  2019-04-02T00:00:00-03:00      3.8655
12141  2019    3      4  GM366_ERC366  2019-04-03T00:00:00-03:00      3.8430
12142  2019    4      4  GM366_ERC366  2019-04-04T00:00:00-03:00      3.8707
12143  2019    5      4  GM366_ERC366  2019-04-05T00:00:00-03:00      3.8616
12144  2019    8      4  GM366_ERC366  2019-04-08T00:00:00-03:00      3.8652
12145  2019    9      4  GM366_ERC366  2019-04-09T00:00:00-03:00      3.8557
12146  2019   10      4  GM366_ERC366  2019-04-10T00:00:00-03:00      3.8339
...     ...  ...    ...           ...                          ...         ...
```

Similarly, if you just want the data for a specific day of a specific month of a specific year use together the parameters `year`, `month` and `day`:

```
>>> ipeadatapy.timeseries("GM366_ERC366", year=2019, month=4, day=1)
       YEAR  DAY  MONTH          CODE                       DATE  VALUE (R$)
12139  2019    1      4  GM366_ERC366  2019-04-01T00:00:00-03:00      3.8676
```

Another option is to return only data relative to years greater than some year, say, 2017. For this, use the parameter `yearGreaterThan`:

```
>>> ipeadatapy.timeseries("GM366_ERC366", yearGreaterThan=2017)
       YEAR  DAY  MONTH          CODE                       DATE  VALUE (R$)
11828  2018    2      1  GM366_ERC366  2018-01-02T00:00:00-02:00      3.2691
11829  2018    3      1  GM366_ERC366  2018-01-03T00:00:00-02:00      3.2529
11830  2018    4      1  GM366_ERC366  2018-01-04T00:00:00-02:00      3.2312
11831  2018    5      1  GM366_ERC366  2018-01-05T00:00:00-02:00      3.2403
11832  2018    8      1  GM366_ERC366  2018-01-08T00:00:00-02:00      3.2351
11833  2018    9      1  GM366_ERC366  2018-01-09T00:00:00-02:00      3.2391
11834  2018   10      1  GM366_ERC366  2018-01-10T00:00:00-02:00      3.2461
11835  2018   11      1  GM366_ERC366  2018-01-11T00:00:00-02:00      3.2295
11836  2018   12      1  GM366_ERC366  2018-01-12T00:00:00-02:00      3.2192
11837  2018   15      1  GM366_ERC366  2018-01-15T00:00:00-02:00      3.1957
...     ...  ...    ...           ...                          ...         ...
[340 rows x 6 columns]
```

You can also select an interval of years, say, from 2017 to 2018 using together with `yearGreaterThan` the parameter `yearSmallerThan`:

```
>>> ipeadatapy.timeseries("GM366_ERC366", yearGreaterThan=2016, yearSmallerThan=2019)
       YEAR  DAY  MONTH          CODE                       DATE  VALUE (R$)
11579  2017    2      1  GM366_ERC366  2017-01-02T00:00:00-02:00      3.2723
11580  2017    3      1  GM366_ERC366  2017-01-03T00:00:00-02:00      3.2626
11581  2017    4      1  GM366_ERC366  2017-01-04T00:00:00-02:00      3.2327
11582  2017    5      1  GM366_ERC366  2017-01-05T00:00:00-02:00      3.2123
11583  2017    6      1  GM366_ERC366  2017-01-06T00:00:00-02:00      3.2051
11584  2017    9      1  GM366_ERC366  2017-01-09T00:00:00-02:00      3.2091
11585  2017   10      1  GM366_ERC366  2017-01-10T00:00:00-02:00      3.1912
11586  2017   11      1  GM366_ERC366  2017-01-11T00:00:00-02:00      3.2148
11587  2017   12      1  GM366_ERC366  2017-01-12T00:00:00-02:00      3.1655
11588  2017   13      1  GM366_ERC366  2017-01-13T00:00:00-02:00      3.2028
11589  2017   16      1  GM366_ERC366  2017-01-16T00:00:00-02:00      3.2228
```

```
11590  2017   17      1  GM366_ERC366  2017-01-17T00:00:00-02:00      3.2094
11591  2017   18      1  GM366_ERC366  2017-01-18T00:00:00-02:00      3.2205
11592  2017   19      1  GM366_ERC366  2017-01-19T00:00:00-02:00      3.2107
11593  2017   20      1  GM366_ERC366  2017-01-20T00:00:00-02:00      3.1912
...     ...  ...    ...           ...                        ...         ...
[499 rows x 6 columns]
```

The same logic applies to the parameters `monthGreaterThan` and `monthSmallerThan`. For example, let's restrict the function output to an interval of months (e.g.: from june to december) for a specifc year, say, 2018:

```
>>> ipeadatapy.timeseries("GM366_ERC366", year=2018, monthGreaterThan=5,␣
↪monthSmallerThan=13)
       YEAR  DAY  MONTH          CODE                        DATE  VALUE (R$)
11931  2018    1      6  GM366_ERC366  2018-06-01T00:00:00-03:00      3.7407
11932  2018    4      6  GM366_ERC366  2018-06-04T00:00:00-03:00      3.7418
11933  2018    5      6  GM366_ERC366  2018-06-05T00:00:00-03:00      3.7746
11934  2018    6      6  GM366_ERC366  2018-06-06T00:00:00-03:00      3.8187
11935  2018    7      6  GM366_ERC366  2018-06-07T00:00:00-03:00      3.8994
11936  2018    8      6  GM366_ERC366  2018-06-08T00:00:00-03:00      3.7853
11937  2018   11      6  GM366_ERC366  2018-06-11T00:00:00-03:00      3.6907
11938  2018   12      6  GM366_ERC366  2018-06-12T00:00:00-03:00      3.7038
11939  2018   13      6  GM366_ERC366  2018-06-13T00:00:00-03:00      3.7048
11940  2018   14      6  GM366_ERC366  2018-06-14T00:00:00-03:00      3.7051
11941  2018   15      6  GM366_ERC366  2018-06-15T00:00:00-03:00      3.7732
11942  2018   18      6  GM366_ERC366  2018-06-18T00:00:00-03:00      3.7537
11943  2018   19      6  GM366_ERC366  2018-06-19T00:00:00-03:00      3.7560
11944  2018   20      6  GM366_ERC366  2018-06-20T00:00:00-03:00      3.7329
...     ...  ...    ...           ...                        ...         ...
[147 rows x 6 columns]
```

From now on, use your creativity. There are a lot of possibilities with these parameter combinations. The available parameters for the function timeseries() can be found using the function `help()`: `help(ipeadatapy.timeseries)`.

## 3.4 Metadata

Every Ipeadata's time series is accompanied by a set of metadata. Metadata are data about data. Some examples of the elements of this set of metadata are country, big theme, theme, source and unit of measure. Some specific kinds of metadata have their own function on Ipeadata API. Let's see some of them:

### 3.4.1 Countries

You can have a look at the available Ipeadata's countries by running the `countries()` function:

```
>>> ipeadatapy.countries()
      ID                        COUNTRY
0    ZAF                  ?frica do Sul
1    DEU                       Alemanha
2   LATI                 Am?rica Latina
3    AGO                         Angola
4    SAU                 Ar?bia Saudita
5    DZA                        Arg?lia
6    ARG                      Argentina
7    AUS                      Austr?lia
```

```
8    AUT                          ?ustria
9    BEL                          B?lgica
10   BOL                          Bol?via
..   ...                              ...
```

### 3.4.2 Themes

You can also have a look on the available themes for Ipeadata using the function `themes()`:

```
>>> ipeadatapy.themes()
    ID                      NAME  MACRO  REGIONAL  SOCIAL
0   28             Agropecu?ria    NaN       1.0     NaN
1   23       Assist?ncia social    NaN       NaN     1.0
2   25     Avalia??o do governo    NaN       NaN     NaN
3   10    Balan?o de pagamentos    1.0       NaN     NaN
4    7                  C?mbio    1.0       NaN     NaN
5    5        Com?rcio exterior    1.0       1.0     NaN
6    2         Consumo e vendas    1.0       1.0     NaN
7    8         Contas nacionais    1.0       NaN     NaN
8   81         Contas Regionais    NaN       1.0     NaN
9   24       Corre??o monet?ria    1.0       NaN     NaN
10  37               Demografia    NaN       NaN     1.0
..  ..                      ...    ...       ...     ...
```

Let's suppose you have the interest to know which of the themes of Ipeadata are related to the Macroeconomics big theme. The parameter `macro` will solve this problem:

```
>>> ipeadatapy.themes(macro=1)
    ID                      NAME  MACRO  REGIONAL  SOCIAL
3   10    Balan?o de pagamentos    1.0       NaN     NaN
4    7                  C?mbio    1.0       NaN     NaN
5    5        Com?rcio exterior    1.0       1.0     NaN
6    2         Consumo e vendas    1.0       1.0     NaN
7    8         Contas nacionais    1.0       NaN     NaN
9   24       Corre??o monet?ria    1.0       NaN     NaN
..  ..                      ...    ...       ...     ...
```

Let's now suppose that you just want the function to return themes that are related both to the macroeconomics and regional themes. For this, use `macro` and `regional` parameters together:

```
>>> ipeadatapy.themes(macro=1, regional=1)
    ID                NAME  MACRO  REGIONAL  SOCIAL
5    5  Com?rcio exterior    1.0       1.0     NaN
6    2   Consumo e vendas    1.0       1.0     NaN
18  12            Emprego    1.0       1.0     NaN
19  19  Estoque de capital    1.0       1.0     NaN
20   6   Finan?as p?blicas    1.0       1.0     NaN
31   3     Moeda e cr?dito    1.0       1.0     NaN
33  14           Popula??o    1.0       1.0     NaN
34   9              Pre?os    1.0       1.0     NaN
37   1           Produ??o    1.0       1.0     NaN
45  33          Transporte    1.0       1.0     NaN
..  ..                 ...    ...       ...     ...
```

The parameter `social` is also available and works in the same way of macro and regional. For more parameters available for the function `themes()` run `help(idpy.themes)`.

### 3.4.3 Sources

Other important metadata is the source. This metadata have his own functions, `sources()`. Let's have a look:

```
>>> ipeadatapy.sources()
0                          Abia
1                        Abinee
2                          ABPO
3                       Abracal
4                         Abras
5                      ACSP/IEGV
6                          Anac
7                        Anatel
8                        Anbima
9                        Anbima
10                         Anda
..                          ...
```

### 3.4.4 Territories

For regional time series we also have some information about Brazilian territories through the function `territories()`:

```
>>> ipeadatapy.territories()
                                                NAME             ID    ...
↪AREA  CAPITAL
0                                   (n?o definido)                     ...
↪  NaN     None
1                                            Brasil              0    ...
↪8531507.6    False
2                                     Regi?o Norte              1    ...
↪3869637.9    False
3                                         Rond?nia             11    ...
↪238512.8    False
4                          Alta Floresta D'Oeste        1100015    ...
↪7111.8    False
5                                       Ariquemes       1100023    ...
↪4995.3    False
6                                          Cabixi       1100031    ...
↪1530.7    False
7                                          Cacoal       1100049    ...
↪3808.4    False
8                                       Cerejeiras       1100056    ...
↪2645.0    False
9                              Colorado do Oeste        1100064    ...
↪1442.4    False
10                                      Corumbiara       1100072    ...
↪3079.7    False
...                                            ...           ...    ...
↪  ...      ...
```

Two interesting parameters of `territories()` function are `areaGreaterThan` and `areaSmallerThan`. With these parameters, it is possible to filter the return of the function for just territories greater than, smaller than or between the specified parameters. For example, let's check which of the Brazilian territories have the area greater than 1000000:

```
>>> ipeadatapy.territories(areaGreaterThan=1000000)
                      NAME                    ID        LEVEL       AREA CAPITAL
1                   Brasil                     0       Brasil  8531507.6   False
2             Regi?o Norte                     1      Regi?es  3869637.9   False
138               Amazonas                    13      Estados  1577820.2   False
386                   Par?                    15      Estados  1253164.5   False
1161       Regi?o Nordeste                     2      Regi?es  1558200.4   False
17960  Regi?o Centro-oeste                     5      Regi?es  1612077.2   False
18452     AMC1872_1997 001  513AMC1872_1997001  AMC 1872-00  1947986.1    None
18454          AMC2097 001        51AMC2097001    AMC 20-00  1061175.7    None
```

Let's now check the territories which the area is between 1000000 and 1100000:

```
>>> ipeadatapy.territories(areaGreaterThan=1000000, areaSmallerThan=1500000)
              NAME            ID      LEVEL       AREA CAPITAL
386           Par?            15    Estados  1253164.5   False
18454  AMC2097 001  51AMC2097001  AMC 20-00  1061175.7    None
```

### 3.4.5 Other metadata

Although only 4 metadata from Ipeadata have their own function, there are a lot more metadata available for the data base time series. The function `metadata()` returns all Ipeadata time series in a data frame with all of his metadata. Each of the collumns of the data frame represents a metadata.

```
>>> ipeadatapy.metadata()
          BIG THEME                                                SOURCE SOURCE␣
↪ACRONYM            ...            SERIES STATUS THEME CODE                  MEASURE
0     Macroecon?mico  Instituto Brasileiro de Geografia e Estat?stic...    IBGE/
↪Coagro             ...                              A          1               Tonelada
1     Macroecon?mico  Instituto Brasileiro de Geografia e Estat?stic...    IBGE/
↪Coagro             ...                              A          1               Tonelada
2     Macroecon?mico  Instituto Brasileiro de Geografia e Estat?stic...    IBGE/
↪Coagro             ...                              A          1               Tonelada
3     Macroecon?mico  Instituto Brasileiro de Geografia e Estat?stic...    IBGE/
↪Coagro             ...                              A          1                  Cabe?a
4     Macroecon?mico  Instituto Brasileiro de Geografia e Estat?stic...    IBGE/
↪Coagro             ...                              A          1                  Cabe?a
5     Macroecon?mico  Instituto Brasileiro de Geografia e Estat?stic...    IBGE/
↪Coagro             ...                              A          1                  Cabe?a
6     Macroecon?mico  Instituto Brasileiro de Geografia e Estat?stic...    IBGE/
↪Coagro             ...                              I          1               Tonelada
7     Macroecon?mico  Instituto Brasileiro de Geografia e Estat?stic...    IBGE/
↪Coagro             ...                              A          1               Tonelada
8     Macroecon?mico  Instituto Brasileiro de Geografia e Estat?stic...    IBGE/
↪Coagro             ...                              A          1               Tonelada
9     Macroecon?mico  Instituto Brasileiro de Geografia e Estat?stic...    IBGE/
↪Coagro             ...                              A          1               Tonelada
10    Macroecon?mico  Instituto Brasileiro de Geografia e Estat?stic...    IBGE/
↪Coagro             ...                              A          1               Tonelada
11    Macroecon?mico  Instituto Brasileiro de Geografia e Estat?stic...    IBGE/
↪Coagro             ...                              A          1               Tonelada
12    Macroecon?mico  Instituto Brasileiro de Geografia e Estat?stic...    IBGE/
↪Coagro             ...                              A          1               Tonelada
13    Macroecon?mico  Instituto Brasileiro de Geografia e Estat?stic...    IBGE/
↪Coagro             ...                              I          1               Tonelada
14    Macroecon?mico  Instituto Brasileiro de Geografia e Estat?stic...    IBGE/
↪Coagro             ...                              I          1                  Cabe?a
```

```
15    Macroecon?mico  Instituto Brasileiro de Geografia e Estat?stic...    IBGE/
↪Coagro            ...                          A          1                   Cabe?a
...              ...                                                  ...           ...
↪             ...                      ...        ...                      ...
[8549 rows x 15 columns]
```

As you can see, this data frame is too big to be represented here. His dimension is 8549 rows by 15 columns. Each of these columns represents one metadata. The columns are BIG THEME, SOURCE, SOURCE ACRONYM, SOURCE URL, UNIT, COUNTRY, FREQUENCY, LAST UPDATE, CODE, COMMENT, NAME, NUMERICA, SERIES STATUS, THEME CODE, and MEASURE. In the next section, we will learn how to use these metadata as filtering options to improve our research.

## 3.5 Advanced filtering using metadata

### 3.5.1 The metadata() function

Now that you have knowledge of some of the metadata of Ipeadata, let's introduce yourself to a function called `metadata()`. This function returns all Ipeadata's time series in a data frame, similarly to the `list_series()` function. However, the difference between the two functions is that `metadata()` returns not only the time series but also their metadata. You might then be asking yourself why these two functions exists, since `metadata()` is a more complete version of the `list_series()` function (`metadata()` features all of the `list_series()` information plus metadata). The answer is: `list_series()` is intended to be a more simplistic version, aiming unexperienced users and designed to be friendly to them. `metadata()`, in fact, is a more complete version as well as more confusing because of the quantity of information returned. No more words, let's run the function:

```
>>> ipeadatapy.metadata()
          BIG THEME                                                SOURCE SOURCE␣
↪ACRONYM            ...            SERIES STATUS THEME CODE                  MEASURE
0     Macroecon?mico  Instituto Brasileiro de Geografia e Estat?stic...    IBGE/
↪Coagro            ...                          A          1                 Tonelada
1     Macroecon?mico  Instituto Brasileiro de Geografia e Estat?stic...    IBGE/
↪Coagro            ...                          A          1                 Tonelada
2     Macroecon?mico  Instituto Brasileiro de Geografia e Estat?stic...    IBGE/
↪Coagro            ...                          A          1                 Tonelada
3     Macroecon?mico  Instituto Brasileiro de Geografia e Estat?stic...    IBGE/
↪Coagro            ...                          A          1                   Cabe?a
4     Macroecon?mico  Instituto Brasileiro de Geografia e Estat?stic...    IBGE/
↪Coagro            ...                          A          1                   Cabe?a
5     Macroecon?mico  Instituto Brasileiro de Geografia e Estat?stic...    IBGE/
↪Coagro            ...                          A          1                   Cabe?a
6     Macroecon?mico  Instituto Brasileiro de Geografia e Estat?stic...    IBGE/
↪Coagro            ...                          I          1                 Tonelada
7     Macroecon?mico  Instituto Brasileiro de Geografia e Estat?stic...    IBGE/
↪Coagro            ...                          A          1                 Tonelada
8     Macroecon?mico  Instituto Brasileiro de Geografia e Estat?stic...    IBGE/
↪Coagro            ...                          A          1                 Tonelada
9     Macroecon?mico  Instituto Brasileiro de Geografia e Estat?stic...    IBGE/
↪Coagro            ...                          A          1                 Tonelada
10    Macroecon?mico  Instituto Brasileiro de Geografia e Estat?stic...    IBGE/
↪Coagro            ...                          A          1                 Tonelada
11    Macroecon?mico  Instituto Brasileiro de Geografia e Estat?stic...    IBGE/
↪Coagro            ...                          A          1                 Tonelada
12    Macroecon?mico  Instituto Brasileiro de Geografia e Estat?stic...    IBGE/
↪Coagro            ...                          A          1                 Tonelada
```

```
13    Macroecon?mico  Instituto Brasileiro de Geografia e Estat?stic...    IBGE/
↪Coagro           ...                            I          1                  Tonelada
14    Macroecon?mico  Instituto Brasileiro de Geografia e Estat?stic...    IBGE/
↪Coagro           ...                            I          1                      Cabe?a
15    Macroecon?mico  Instituto Brasileiro de Geografia e Estat?stic...    IBGE/
↪Coagro           ...                            A          1                      Cabe?a
...             ...                                                   ...           ...
↪            ...                      ...        ...                      ...
[8549 rows x 15 columns]
```

### 3.5.2 Better filtering with metadata()

Why is this function so powerful and important? The first obvious answer is: it gives you more informations about time series. The not-so-obvious answer is: it allows you to better filter time series from Ipeadata. Let's state an illustrative problem for better understanding:

> *Ipeadata API has 8565 time series in total. Let's suppose you are doing research in macroeconomics about the United States, but for some specific reason, your interest in data is restricted to data published by The Economist. It also needs to be quarterly published. How to solve this problem using ipeadatapy Python package?*

```
>>> ipeadatapy.metadata(big_theme="Macroecon?mico", country="USA", source="Economist",
↪ frequency="Trimestral")
          BIG THEME         SOURCE SOURCE ACRONYM          SOURCE URL     UNIT     ...␣
↪                                                NAME NUMERICA SERIES STATUS THEME␣
↪CODE   MEASURE
5585  Macroecon?mico  The Economist      Economist  www.economist.com  bilh?es     ...␣
↪    balan?o - conta corrente - saldo (acum. 12 meses)     True             I        ␣
↪ 11       US$
5586  Macroecon?mico  The Economist      Economist  www.economist.com     None     ...␣
↪                  PIB - var. real trimestral anualiz.     True             A        ␣
↪ 11  (% a.a.)
5587  Macroecon?mico  The Economist      Economist  www.economist.com     None     ...␣
↪    PIB - var. real contra igual trimestre do ano ...     True             A        ␣
↪ 11  (% a.a.)
[3 rows x 15 columns]
```

Gotcha! Other metadata also can be used as filtering parameters. For all parameters run `help(idpy.metadata)`.

## 3.6 Data Visualization

Although data visualization is not the main purpose of Ipeadatapy package, one of ipeadatapy's dependencies (pandas) allows plots. Because ipeadatapy is a package related to time series, graphic representations are very important. Thus, we consider a good idea to include the pandas' `plot()` function here in our documentation. It is good to have knowledge of the possibility of plotting the time series directly trough Python. Let's do an example by plotting the data of GM366_ERC366 time series for April, 2019:

```
ipeadatapy.timeseries("GM366_ERC366", year=2019, month=4).plot("DAY", "VALUE (R$)")
```


Note that the `plot()` function parameters are, respectively, the x and y axis of the graph. These parameters must match the column titles of the desired data in the `timeseries()` data frame. The tip for correctly filling these parameters is to first run the `timeseries()` function alone for your desired time series, check the column names to, then, use these column titles as parameters for the `plot()` function.

## 3.7 Data Extraction

In this section we will show you how to extract data from ipeadatapy using the package together with one of his dependencies (pandas) and other Python Built-In features. One of the most useful aspects of having an API wrapper is to have the option of extracting data in a more efficient and practical way than extracting the same data from the database's website. With this package it's possible to extract not only data but also quantitative and descriptive metadatas from Ipeadata database. As we will see, you can even extract mass quantities of spreadsheets at once, being also possible extraction filterings accordingly to your needs. Let's start with the basics.

### 3.7.1 Extracting one single time series

Let's first import ipeadatapy package.

```
>>> import ipeadatapy
```

then, let's suppose that we already know which time series we want to extract and already found his code using the `list_series()` function. Let, e.g., this time series be the one which the code is GM366_ERC366. Thus, we can show the data calling the following function:

```
>>> ipeadatapy.timeseries('GM366_ERC366')
       YEAR  DAY  MONTH          CODE                       DATE    VALUE (R$)
0      1985    2      1  GM366_ERC366  1985-01-02T00:00:00-02:00  1.152000e-09
1      1985    3      1  GM366_ERC366  1985-01-03T00:00:00-02:00  1.152000e-09
2      1985    4      1  GM366_ERC366  1985-01-04T00:00:00-02:00  1.152000e-09
```

```
3      1985    5      1  GM366_ERC366  1985-01-05T00:00:00-02:00           NaN
...     ...   ...    ...           ...                        ...           ...
```

To extract this dataframe as a .xlsx file you simply do:

```
>>> ipeadatapy.timeseries('GM366_ERC366').to_excel('.../path/yourFileName.xlsx')
```

If you prefer, you can extract the data in csv format instead of xlsx. For this, you can use the function called `to_csv()`:

```
>>> ipeadatapy.timeseries('GM366_ERC366').to_csv('.../path/yourFileName.csv', sep=';')
```

where '. . . /path/' represents the desired directory path where the file will be placed and 'yourFile.csv' the name of the file. If you just set the file name without a directory path, then the file will be saved in the directory where you are running your Python. Pay attention to not omit the filename extensions, '.csv' or '.xlsx'.

### 3.7.2 Extracting multiple time series at once

Let's suppose that we want to extract more than one time series at once. We can do it by defining a list containing the codes of the time series that we want to extract then running a 'for' structure that will loop through this list, extracting a file for each of the timeseries contained there. For illustration purposes, let's suppose that we want to extract these three timeseries: GM366_ERC366, IBMEC12_TJTIT12 and PIBE. Then we need to define a list containing them:

```
>>> timeseries = ['GM366_ERC366', 'IBMEC12_TJTIT12', 'PIBE']
```

Now, let's define a 'for' loop to extract these series:

```
>>> for i in timeseries:
        ipeadatapy.timeseries(i).to_csv(i + '.csv', sep=';')
```

The output will be three '.csv' files: GM366_ERC366.csv, IBMEC12_TJTIT12.csv and PIBE.csv, saved in the directory where you runned your Python.

### 3.7.3 Extracting other kinds of data

Ipeadatapy's functions were defined to always return data in the form of data frames. Thus, every function output can be extracted using pandas' `to_csv()` or `to_excel()` functions in the same way we've shown in the past example.

CHAPTER

# FOUR

# FUNCTIONS

## 4.1 list_series

| list_series(keyword=None, code=None, name=None) | | |
|---|---|---|
| Lists Ipeadata available time series. | | |
| keyword | str, optional | Filtering keyword, defaults to None |
| code | str, optional | Specifies a time series code to return. code must be a time series code respecting case sensitiveness |
| name | str, optional | Specifies a time series name to return. name must be a time series name respecting case sensitiveness |
| return | pandas.DataFrame | If no keyword is specified, returns a data frame containing all Ipeadata?s time series. Otherwise, returns a data frame respecting the introduced parameters |

[source]

## 4.2 describe

| describe(series) | | |
|---|---|---|
| Describes the specified time series. series must be the time series' code. | | |
| series | str | Time series code |

[source]

## 4.3 timeseries

| timeseries(series, groupby=None, year=None, yearGreaterThan=None, yearSmallerThan=None, day=None, dayGreaterThan=None, daySmallerThan=None, month=None, monthGreaterThan=None, monthSmallerThan=None, code=None, date=None) | | |
|---|---|---|
| Returns the specified time series' data values. *series* must be a time series code | | |
| series | str | Time series code. For the available time series run list_series() |
| groupby | str, optional | Grouping criteria |
| year | int, optional | Year which the data set will be restricted to. |
| yearGreaterThan | int, optional | Year which the data set will be restricted to years strictly greater. |
| yearSmallerThan | int, optional | Year which the data set will be restricted to years strictly smaller. |
| day | int, optional | Day which the data set will be restricted to. |
| dayGreaterThan | int, optional | Day which the data set will be restricted to days strictly greater. |
| daySmallerThan | int, optional | Day which the data set will be restricted to days strictly smaller. |
| month | int, optional | Month which the data set will be restricted to. |
| monthGreaterThan | int, optional | Month which the data set will be restricted to months strictly greater. |
| monthSmallerThan | int, optional | Month which the data set will be restricted to months strictly smaller. |
| code | str, optional | Time series code which the data set will be restricted to. |
| date | str, optional | Date which the data set will be restricted to. |
| return | pandas.DataFrame | Returns the data series for the specified time series. |

[source]

## 4.4 metadata

| metadata(series=None, big_theme=None, source=None, country=None, frequency=None, unit=None, measure=None, status=None, source_ext=None, source_url=None, last_update=None, code=None, comment=None, name=None, numerica=None, theme_code=None) | | |
|---|---|---|
| | | |
| series \| str, optional | | Time series code. For the available time series run list_series() |
| big_theme | str, optional | Big theme by which the return will be fitered. Options: “Macroecon?mico”, “Regional” or “Social” |
| source | str, optional | Source by which the return will be filtered. For available sources run sources() function. |
| country | str, optional | Country ID by which the return will be filtered. For available countries and their IDs run countries() function. |
| frequency | str, optional | Frequency by which the return will be filtered. |
| unit | str, optional | Unit by which the return will be filtered. |
| measure | str, optional | Measure by which the return will be filtered. |
| status | str, optional | Status by which the return will be filtered. Available options: “A” and “I” |
| source_ext | str, optional | Source extended name by which the return will be filtered. |
| source_url | str, optional | Source URL by which the return will be filtered. |
| last_update | str, optional | Last update date by which the return will be filtered. |
| code | str, optional | Time series code by which the return will be filtered. |
| comment | str, optional | Time series comment by which the return will be filtered. |
| name | str, optional | Time series name by which the return will be filtered. |
| numerica | bool, optional | Numeric? True or False. |
| theme_id | str, optional | Theme by which the return will be filtered. For available themes run themes() function |
| return | pandas.DataFrame | If no keyword is specified, returns a data frame containing all Ipeadata’s time series. Else, returns only the ones that respects the specified parameters |

[source]

## 4.5 latest_updates

| latest_updates() |
|---|
| Returns the latest time series’ updates from Ipeadata, from the most to the less recent updated time series. |

[source]

## 4.6 sources

| sources() |
| --- |
| Returns available Ipeadata's sources in the form of a data frame. |

[source]

## 4.7 themes

| themes(theme_id=None, name=None, macro=None, regional=None, social=None) | | |
| --- | --- | --- |
| | | |
| theme_id | int, optional | Theme ID by which the return will be filtered |
| name | str, optional | Theme name by which the return will be filtered |
| macro | int, optional | If macro=1, the function will return only themes related to the Macroeconomics big theme |
| regional | int, optional | If regional=1, the function will return only themes related to the Regional big theme |
| social | int, optional | If social=1, the function will return only themes related to the Social big theme |
| return | pandas.DataFrame | Returns available Ipeadata themes |

[source]

## 4.8 territories

| territories(name=None, level=None, territory_id=None, area=None, areaGreaterThan=None, areaSmallerThan=None, capital=None): | | |
| --- | --- | --- |
| | | |
| name | str, optional | Territory name by which the return will be fitered. |
| level | str, optional | Territory name by which the return will be fitered. |
| territory_id | str, optional | Territory ID by which the return will be fitered. |
| area | float, optional | Territorial area by which the return will be fitered. |
| areaGreaterThan | float, optional | Territorial area restriction by which the return will be fitered. The function will return only territories with area strictly greater than the submitted value. |
| areaSmallerThan | float, optional | Territorial area restriction by which the return will be fitered. The function will return only territories with area strictly smaller than the submitted value. |
| capital | bool, optional | Return only capitals? True or False. |
| return | pandasDataframe | Returns available Ipeadata territories. |

[source]

## 4.9 countries

| countries() |
| --- |
| Returns all available Ipeadata's countries and country IDs in the form of a data frame. |

[source]

## 4.10 api_call

| api_call(api) | |
| --- | --- |
| For advanced users. Returns raw Ipeadata API data in the form of a data frame. | |
| **api** | str |

[source]