econuy

Release 0.22.2

Rafael Xavier

Mar 12, 2022

CONTENTS

Wrangling Uruguayan economic data so you don't have to.

CHAPTER

ONE

USER DOCUMENTATION

Check the readme for a brief explainer on how to use the package, what extra stuff you might need and which problems you might face.

1.1 Overview

This project simplifies gathering and processing of Uruguayan economic statistics. Data is retrieved from (mostly) government sources, processed into a familiar tabular format, tagged with useful metadata and can be transformed in several ways (converting to dollars, calculating rolling averages, resampling to other frequencies, etc.).

If this screenshot gives you anxiety, this package should be of interest.

A webapp with a limited but interactive version of econuy is available at econ.uy. Check out the repo as well.

The most basic econuy workflow goes like this:

```
from econuy.core import Pipeline

p = Pipeline()
p.get("labor_rates")
```

1.2 Installation

- PyPI:

```
pip install econuy
```

- Git:

```
git clone https://github.com/rxavier/econuy.git
cd econuy
python setup.py install
```

1.3 Usage

Full API documentation available at RTD

1.3.1 The `Pipeline()` class

This is the recommended entry point for the package. It allows setting up the common behavior for downloads, and holds the current working dataset.

```
from econuy.core import Pipeline

p = Pipeline(location="your_directory")
```

The `Pipeline.get()` method

Retrieves datasets (generally downloads them, unless the `download` attribute is `False` and the requested dataset exists at the `location`) and loads them into the `dataset` attribute as a Pandas DataFrame.

The `Pipeline.available_datasets()` method returns a `dict` with the available options.

```
from econuy.core import Pipeline
from sqlalchemy import create_engine

eng = create_engine("dialect+driver://user:pwd@host:port/database")

p = Pipeline(location=eng)
p.get("industrial_production")
```

Which also shows that econuy supports SQLAlchemy `Engine` or `Connection` objects.

Note that every time a dataset is retrieved, `Pipeline` will

1. Check if a previous version exists at `location`. If it does, it will read it and combine it with the new data (unless `download=False`, in which case only existing data will be retrieved)
2. Save the dataset to `location`, unless the `always_save` attribute is set to `False` or no new data is available.

Data can be written and read to and from CSV or Excel files (controlled by the `read_fmt` and `save_fmt` attributes) or SQL (automatically determined from `location`).

Dataset metadata

Metadata for each dataset is held in Pandas MultiIndexes with the following:

1. Indicator name
2. Topic or area
3. Frequency
4. Currency
5. Inflation adjustment
6. Unit
7. Seasonal adjustment

8. Type (stock or flow)
9. Cumulative periods

When writing, metadata can be included as dataset headers (Pandas MultiIndex columns), placed on another sheet if writing to Excel, or dropped. This is controlled by `read_header` and `save_header`.

Pipeline transformation methods

`Pipeline` objects with a valid dataset can access 6 transformation methods that modify the held dataset.

- `resample()` - resample data to a different frequency, taking into account whether data is of stock or flow type.
- `chg_diff()` - calculate percent changes or differences for same period last year, last period or at annual rate.
- `decompose()` - seasonally decompose series into trend or seasonally adjusted components.
- `convert()` - convert to US dollars, constant prices or percent of GDP.
- `rebase()` - set a period or window as 100, scale rest accordingly
- `rolling()` - calculate rolling windows, either average or sum.

```
from econuy.core import Pipeline

p = Pipeline()
p.get("balance_nfps")
p.convert(flavor="usd")
p.resample(rule="A-DEC", operation="sum")
```

Saving the current dataset

While `Pipeline.get()` will generally save the retrieved dataset to `location`, transformation methods won't automatically write data.

However, `Pipeline.save()` can be used, which will overwrite the file on disk (or SQL table) with the contents in `dataset`.

1.3.2 The `Session()` class

Like a `Pipeline`, except it can hold several datasets.

The `datasets` attribute is a `dict` of name-DataFrame pairs. Additionally, `Session.get()` accepts a sequence of strings representing several datasets.

Transformation and saving methods support a `select` parameter that determines which held datasets are considered.

```
from econuy.session import Session

s = Session(location="your/directory")
s.get(["cpi", "nxr_monthly"])
s.get("commodity_index")
s.rolling(window=12, operation="mean", select=["nxr_monthly", "commodity_index"])
```

`Session.get_bulk()` makes it easy to get several datasets in one line.

```
from econuy.session import Session

s = Session()
s.get_bulk("all")
```

```
from econuy.session import Session

s = Session()
s.get_bulk("fiscal_accounts")
```

`Session.concat()` combines selected datasets into a single DataFrame with a common frequency, and adds it as a new key-pair in `datasets`.

1.3.3 External binaries and libraries

unrar libraries

The patool package is used in order to access data provided in `.rar` format. This package requires that you have the `unrar` binaries in your system, which in most cases you should already have. You can can get them from here if you don't.

Selenium webdrivers

Some retrieval functions need Selenium to be configured in order to scrape data. These functions include a `driver` parameter in which a Selenium Webdriver can be passed, or they will attempt to configure a Chrome webdriver, even downloading the chromedriver binary if needed. This still requires an existing Chrome installation.

1.4 Caveats and plans

1.4.1 Caveats

This project is heavily based on getting data from online sources that could change without notice, causing methods that download data to fail. While I try to stay on my toes and fix these quickly, it helps if you create an issue when you find one of these (or even submit a fix!).

1.4.2 Plans

- Implement a CLI.
- ~~Provide methods to make keeping an updated database easy~~. `Session.get_bulk()` mostly covers this.
- ~~Visualization.~~ (I have decided that visualization should be up to the end-user. However, the webapp is available for this purpose).
- Translations for dataset descriptions and metadata.

CHAPTER

TWO

API DOCUMENTATION

Or read the API documentation (automatically generated from source code) for the specifics.

2.1 API documentation

2.1.1 Pipeline class

class econuy.core.**Pipeline**(*location: Optional[Union[str, os.PathLike, sqlalchemy.engine.base.Engine, sqlalchemy.engine.base.Connection]] = None*, *download: bool = True*, *always_save: bool = True*, *read_fmt: str = 'csv'*, *read_header: Optional[str] = 'included'*, *save_fmt: str = 'csv'*, *save_header: Optional[str] = 'included'*, *errors: str = 'raise'*)

Bases: `object`

Main class to access download and transformation methods.

location
: Either Path or path-like string pointing to a directory where to find a CSV for updating and saving, SQLAlchemy connection or engine object, or `None`, don't save or update.

  Type str, os.PathLike, SQLAlchemy Connection or Engine, or None, default None

download
: If False the `get` method will only try to retrieve data on disk.

  Type bool, default True

always_Save
: If True, save every retrieved dataset to the specified `location`.

  Type bool, default True

read_fmt
: File format of previously downloaded data. Ignored if `location` points to a SQL object.

  Type {'csv', 'xls', 'xlsx'}

save_fmt
: File format for saving. Ignored if `location` points to a SQL object.

  Type {'csv', 'xls', 'xlsx'}

read_header
: Location of dataset metadata headers. 'included' means they are in the first 9 rows of the dataset. 'separate' means they are in a separate Excel sheet (if `read_fmt='csv'`, headers are discarded). None means there are no metadata headers.

Type {'included', 'separate', None}

save_header
Location of dataset metadata headers. 'included' means they will be set as the first 9 rows of the dataset. 'separate' means they will be saved in a separate Excel sheet (if `save_fmt='csv'`, headers are discarded). None discards any headers.

Type {'included', 'separate', None}

errors
How to handle errors that arise from transformations. `raise` will raise a ValueError, `coerce` will force the data into `np.nan` and `ignore` will leave the input data as is.

Type {'raise', 'coerce', 'ignore'}

property dataset: pandas.core.frame.DataFrame
Get dataset.

property dataset_flat: pandas.core.frame.DataFrame
Get dataset with no metadata in its column names.

property name: str
Get dataset name.

property description: str
Get dataset description.

static available_datasets() → Dict
Get a dictionary with all available datasets.

The dictionary is separated by original and custom keys, which denote whether the dataset has been modified in some way or if its as provided by the source

copy(*deep: bool = False*) → *econuy.core.Pipeline*
Copy or deepcopy a Pipeline object.

Parameters **deep** (*bool, default True*) – If True, deepcopy.

Returns

Return type *Pipeline*

get(*name: str*)
Main download method.

Parameters **name** (*str*) – Dataset to download, see available options in *available_datasets*.

Raises **ValueError** – If an invalid string is given to the `name` argument.

resample(*rule: Union[pandas._libs.tslibs.offsets.DateOffset, pandas._libs.tslibs.timedeltas.Timedelta, str]*, *operation: str = 'sum'*, *interpolation: str = 'linear'*, *warn: bool = False*)
Wrapper for the resample method in Pandas that integrates with econuy dataframes' metadata.

Trim partial bins, i.e. do not calculate the resampled period if it is not complete, unless the input dataframe has no defined frequency, in which case no trimming is done.

Parameters

- **rule** (*pd.DateOffset, pd.Timedelta or str*) – Target frequency to resample to. See Pandas offset aliases
- **operation** (*{'sum', 'mean', 'last', 'upsample'}*) – Operation to use for resampling.

- **interpolation** (*str, default 'linear'*) – Method to use when missing data are produced as a result of resampling, for example when upsampling to a higher frequency. See Pandas interpolation methods
- **warn** (*bool, default False*) – If False, don't raise warnings with incomplete time-range bins.

Returns

Return type None

Raises

- **ValueError** – If `operation` is not one of available options.
- **ValueError** – If the input dataframe's columns do not have the appropiate levels.

Warns **UserWarning** – If input frequencies cannot be assigned a numeric value, preventing incomplete bin trimming.

chg_diff(*operation: str = 'chg', period: str = 'last'*)

Wrapper for the pct_change and diff Pandas methods.

Calculate percentage change or difference for dataframes. The `period` argument takes into account the frequency of the dataframe, i.e., `inter` (for interannual) will calculate pct change/differences with `periods=4` for quarterly frequency, but `periods=12` for monthly frequency.

Parameters

- **df** (*pd.DataFrame*) – Input dataframe.
- **operation** (*{'chg', 'diff'}*) – `chg` for percent change or `diff` for differences.
- **period** (*{'last', 'inter', 'annual'}*) – Period with which to calculate change or difference. `last` for previous period (last month for monthly data), `inter` for same period last year, `annual` for same period last year but taking annual sums.

Returns

Return type None

Raises

- **ValueError** – If the dataframe is not of frequency `M` (month), `Q` or `Q-DEC` (quarter), or `A` or `A-DEC` (year).
- **ValueError** – If the `operation` parameter does not have a valid argument.
- **ValueError** – If the `period` parameter does not have a valid argument.
- **ValueError** – If the input dataframe's columns do not have the appropiate levels.

decompose(*component: str = 'seas', method: str = 'x13', force_x13: bool = False, fallback: str = 'loess', trading: bool = True, outlier: bool = True, x13_binary: Union[str, os.PathLike] = 'search', search_parents: int = 0, ignore_warnings: bool = True, **kwargs*)

Apply seasonal decomposition.

Decompose the series in a Pandas dataframe using either X13 ARIMA, Loess or moving averages. X13 can be forced in case of failure by alternating the underlying function's parameters. If not, it will fall back to one of the other methods. If the X13 method is chosen, the X13 binary has to be provided. Please refer to the README for instructions on where to get this binary.

Parameters

- **component** (*{'seas', 'trend'}*) – Return seasonally adjusted or trend component.

- **method** (*{'x13', 'loess', 'ma'}*) – Decomposition method. `X13` refers to X13 ARIMA from the US Census, `loess` refers to Loess decomposition and `ma` refers to moving average decomposition, in all cases as implemented by statsmodels.
- **force_x13** (*bool, default False*) – Whether to try different `outlier` and `trading` parameters in statsmodels' x13 arima analysis for each series that fails. If `False`, jump to the `fallback` method for the whole dataframe at the first error.
- **fallback** (*{'loess', 'ma'}*) – Decomposition method to fall back to if `method="x13"` fails and `force_x13=False`.
- **trading** (*bool, default True*) – Whether to automatically detect trading days in X13 ARIMA.
- **outlier** (*bool, default True*) – Whether to automatically detect outliers in X13 ARIMA.
- **x13_binary** (*str, os.PathLike or None, default 'search'*) – Location of the X13 binary. If `search` is used, will attempt to find the binary in the project structure. If `None`, statsmodels will handle it.
- **search_parents** (*int, default 0*) – If `x13_binary=search`, this parameter controls how many parent directories to go up before recursively searching for the binary.
- **ignore_warnings** (*bool, default True*) – Whether to suppress X13Warnings from statsmodels.
- **kwargs** – Keyword arguments passed to statsmodels' `x13_arima_analysis`, `STL` and `seasonal_decompose`.

Returns

Return type `None`

Raises

- **ValueError** – If the `method` parameter does not have a valid argument.
- **ValueError** – If the `component` parameter does not have a valid argument.
- **ValueError** – If the `fallback` parameter does not have a valid argument.
- **ValueError** – If the `errors` parameter does not have a valid argument.
- **FileNotFoundError** – If the path provided for the X13 binary does not point to a file and `method='x13'`.

convert(*flavor: str*, *start_date: Optional[Union[str, datetime.datetime]] = None*, *end_date: Optional[Union[str, datetime.datetime]] = None*)

Convert dataframe from UYU to USD, from UYU to real UYU or from UYU/USD to % GDP.

`flavor=usd`: Convert a dataframe's columns from Uruguayan pesos to US dollars. Call the *get* function to obtain nominal exchange rates, and take into account whether the input dataframe's `Type`, as defined by its multiindex, is flow or stock, in order to ` choose end of period or monthly average NXR. Also take into account the input dataframe's frequency and whether columns represent rolling averages or sums.

`flavor=real`: Convert a dataframe's columns to real prices. Call the *get* method to obtain the consumer price index. take into account the input dataframe's frequency and whether columns represent rolling averages or sums. Allow choosing a single period, a range of dates or no period as a base (i.e., period for which the average/sum of input dataframe and output dataframe is the same).

`flavor=gdp`: Convert a dataframe's columns to percentage of GDP. Call the the *get* method to obtain UYU and USD quarterly GDP series. Take into account the input dataframe's currency for chossing UYU or USD GDP. If frequency of input dataframe is higher than quarterly, GDP will be upsampled and linear

interpolation will be performed to complete missing data. If input dataframe's "Acum." level is not 12 for monthly frequency or 4 for quarterly frequency, calculate rolling input dataframe.

In all cases, if input dataframe's frequency is higher than monthly (daily, business, etc.), resample to monthly frequency.

Parameters

- **pipeline** (*econuy.core.Pipeline or None, default None*) – An instance of the econuy Pipeline class.
- **start_date** (*str, datetime.date or None, default None*) – Only used if `flavor=real`. If set to a date-like string or a date, and `end_date` is None, the base period will be `start_date`.
- **end_date** (*str, datetime.date or None, default None*) – Only used if `flavor=real`. If `start_date` is set, calculate so that the data is in constant prices of `start_date-end_date`.
- **errors** (*{'raise', 'coerce', 'ignore'}*) – What to do when a column in the input dataframe is not expressed in Uruguayan pesos. `raise` will raise a ValueError, `coerce` will force the entire column into `np.nan` and `ignore` will leave the input column as is.

Returns

Return type None

Raises

- **ValueError** – If the `errors` parameter does not have a valid argument.
- **ValueError** – If the input dataframe's columns do not have the appropiate levels.

rebase(*start_date: Union[str, datetime.datetime]*, *end_date: Optional[Union[str, datetime.datetime]] = None*, *base: Union[float, int] = 100.0*)

Rebase all dataframe columns to a date or range of dates.

Parameters

- **start_date** (*string or datetime.datetime*) – Date to which series will be rebased.
- **end_date** (*string or datetime.datetime, default None*) – If specified, series will be rebased to the average between `start_date` and `end_date`.
- **base** (*float, default 100*) – Float for which `start_date == base` or average between `start_date` and `end_date == base`.

Returns

Return type None

rolling(*window: Optional[int] = None*, *operation: str = 'sum'*)

Wrapper for the rolling method in Pandas that integrates with econuy dataframes' metadata.

If `periods` is `None`, try to infer the frequency and set `periods` according to the following logic: `{'A': 1, 'Q-DEC': 4, 'M': 12}`, that is, each period will be calculated as the sum or mean of the last year.

Parameters

- **window** (*int, default None*) – How many periods the window should cover.
- **operation** (*{'sum', 'mean'}*) – Operation used to calculate rolling windows.

Returns

Return type None

Raises

- **ValueError** – If `operation` is not one of available options.
- **ValueError** – If the input dataframe's columns do not have the appropiate levels.

Warns **UserWarning** – If the input dataframe is a stock time series, for which rolling operations are not recommended.

save()
Write held dataset.

Raises **ValueError** – If *dataset* is an empty DataFrame or *self.location* is None.

2.1.2 Session class

class econuy.session.**Session**(*location: Optional[Union[str, os.PathLike, sqlalchemy.engine.base.Engine, sqlalchemy.engine.base.Connection]] = None*, *download: bool = True*, *always_save: bool = True*, *read_fmt: str = 'csv'*, *read_header: Optional[str] = 'included'*, *save_fmt: str = 'csv'*, *save_header: Optional[str] = 'included'*, *errors: str = 'raise'*, *log: Union[int, str] = 1*, *logger: Optional[logging.Logger] = None*, *max_retries: int = 3*)

Bases: `object`

A download and transformation session that creates a Pipeline object and simplifies working with multiple datasets.

Alternatively, can be created directly from a Pipeline by using the *from_pipeline* class method.

location
Either Path or path-like string pointing to a directory where to find a CSV for updating and saving, SQLAlchemy connection or engine object, or `None`, don't save or update.

Type str, os.PathLike, SQLAlchemy Connection or Engine, or None, default None

download
If False the `get` method will only try to retrieve data on disk.

Type bool, default True

always_Save
If True, save every retrieved dataset to the specified `location`.

Type bool, default True

read_fmt
File format of previously downloaded data. Ignored if `location` points to a SQL object.

Type {'csv', 'xls', 'xlsx'}

save_fmt
File format for saving. Ignored if `location` points to a SQL object.

Type {'csv', 'xls', 'xlsx'}

read_header
Location of dataset metadata headers. 'included' means they are in the first 9 rows of the dataset. 'separate' means they are in a separate Excel sheet (if `read_fmt='csv'`, headers are discarded). None means there are no metadata headers.

Type {'included', 'separate', None}

save_header
Location of dataset metadata headers. 'included' means they will be set as the first 9 rows of the dataset. 'separate' means they will be saved in a separate Excel sheet (if `save_fmt='csv'`, headers are discarded). None discards any headers.

Type {'included', 'separate', None}

errors
How to handle errors that arise from transformations. `raise` will raise a ValueError, `coerce` will force the data into `np.nan` and `ignore` will leave the input data as is.

Type {'raise', 'coerce', 'ignore'}

log
Controls how logging works. `0`: don't log; `1`: log to console; `2`: log to console and file with default file; `str`: log to console and file with filename=str

Type {str, 0, 1, 2}

logger
Logger object. For most cases this attribute should be `None`, allowing *log* to control how logging works.

Type logging.Logger, default None

max_retries
Number of retries for `get` in case any of the selected datasets cannot be retrieved.

Type int, default 3

classmethod from_pipeline(*pipeline:* econuy.core.Pipeline) → *econuy.session.Session*

property pipeline: *econuy.core.Pipeline*

property datasets: `Dict[str, pandas.core.frame.DataFrame]`
Holds retrieved datasets.

Returns **Datasets**

Return type Dict[str, pd.DataFrame]

property datasets_flat: `Dict[str, pandas.core.frame.DataFrame]`
Holds retrieved datasets.

Returns **Datasets**

Return type Dict[str, pd.DataFrame]

copy(*deep: bool = False*) → *econuy.session.Session*
Copy or deepcopy a Session object.

Parameters **deep** (*bool, default True*) – If True, deepcopy.

Returns

Return type *Session*

static available_datasets(*functions: bool = False*) → Dict[str, Dict]
Return available `dataset` arguments for use in *get*.

Returns **Dataset**

Return type Dict[str, Dict]

get(*names: Union[str, Sequence[str]]*)
Main download method.

Parameters **names** (*Union[str, Sequence[str]]*) – Dataset to download, see available options in *available_datasets*. Either a string representing a dataset name or a sequence of strings in order to download several datasets.

Raises **ValueError** – If an invalid string is found in the names argument.

get_bulk(*names: str*)

Get datasets in bulk.

Parameters **names** (*{'all', 'original', 'custom', 'economic_activity', 'prices', 'fiscal_accounts', 'labor', 'external_sector', 'financial_sector', 'income', 'international', 'regional'}*) – Type of data to download. *all* gets all available datasets, *original* gets all original datatsets and *custom* gets all custom datasets. The remaining options get all datasets for that area.

Raises **ValueError** – If an invalid string is given to the names argument.

resample(*rule: Union[pandas._libs.tslibs.offsets.DateOffset, pandas._libs.tslibs.timedeltas.Timedelta, str, List]*, *operation: Union[str, List] = 'sum'*, *interpolation: Union[str, List] = 'linear'*, *warn: Union[bool, List] = False*, *select: Union[str, int, Sequence[str], Sequence[int]] = 'all'*)

Resample to target frequencies.

See also:

resample

chg_diff(*operation: Union[str, List] = 'chg'*, *period: Union[str, List] = 'last'*, *select: Union[str, int, Sequence[str], Sequence[int]] = 'all'*)

Calculate pct change or difference.

See also:

chg_diff

decompose(*component: Union[str, List] = 'seas'*, *method: Union[str, List] = 'x13'*, *force_x13: Union[bool, List] = False*, *fallback: Union[str, List] = 'loess'*, *trading: Union[bool, List] = True*, *outlier: Union[bool, List] = True*, *x13_binary: Union[str, os.PathLike, List] = 'search'*, *search_parents: Union[int, List] = 0*, *ignore_warnings: Union[bool, List] = True*, *select: Union[str, int, Sequence[str], Sequence[int]] = 'all'*, ***kwargs*)

Apply seasonal decomposition.

See also:

decompose

convert(*flavor: Union[str, List]*, *start_date: Union[str, datetime.datetime, None, List] = None*, *end_date: Union[str, datetime.datetime, None, List] = None*, *select: Union[str, int, Sequence[str], Sequence[int]] = 'all'*)

Convert to other units.

See also:

convert

rebase(*start_date: Union[str, datetime.datetime, List]*, *end_date: Union[str, datetime.datetime, None, List] = None*, *base: Union[float, List] = 100.0*, *select: Union[str, int, Sequence[str], Sequence[int]] = 'all'*)

Scale to a period or range of periods.

See also:

rebase

rolling(*window: Optional[Union[int, List]] = None*, *operation: Union[str, List] = 'sum'*, *select: Union[str, int, Sequence[str], Sequence[int]] = 'all'*)
Calculate rolling averages or sums.

See also:

rolling

concat(*select: Union[str, int, Sequence[str], Sequence[int]] = 'all'*, *concat_name: Optional[str] = None*, *force_suffix: bool = False*)
Concatenate datasets in *datasets* and add as a new dataset.

Resample to lowest frequency of selected datasets.

Parameters

- **select** (*str, int, Sequence[str] or Sequence[int], default "all"*) – Datasets to concatenate.
- **concat_name** (*Optional[str], default None*) – Name used as a key for the output dataset. The default None sets the name to "concat_{dataset_1_name}_..._{dataset_n_name}".
- **force_suffix** (*bool, default False*) – Whether to include each dataset's full name as a prefix in all indicator columns.

save(*select: Union[str, int, Sequence[str], Sequence[int]] = 'all'*)
Write datasets.

Parameters **select** (*str, int, Sequence[str] or Sequence[int], default "all"*) – Datasets to save.

Raises **ValueError** – If *self.location* is None.

2.1.3 Data retrieval functions

econuy.retrieval.economic_activity.**natacc_ind_con_nsa**() → pandas.core.frame.DataFrame
Get supply-side national accounts data in NSA constant prices, 2005-.

Returns **National accounts, supply side, constant prices, NSA**

Return type pd.DataFrame

econuy.retrieval.economic_activity.**natacc_gas_con_nsa**() → pandas.core.frame.DataFrame
Get demand-side national accounts data in NSA constant prices, 2005-.

Returns **National accounts, demand side, constant prices, NSA**

Return type pd.DataFrame

econuy.retrieval.economic_activity.**natacc_gas_cur_nsa**() → pandas.core.frame.DataFrame
Get demand-side national accounts data in NSA current prices.

Returns **National accounts, demand side, current prices, NSA**

Return type pd.DataFrame

econuy.retrieval.economic_activity.**natacc_ind_cur_nsa**() → pandas.core.frame.DataFrame
Get supply-side national accounts data in NSA current prices, 2005-.

Returns **National accounts, supply side, current prices, NSA**

Return type pd.DataFrame

econuy.retrieval.economic_activity.**gdp_con_idx_sa**() → pandas.core.frame.DataFrame
Get supply-side national accounts data in SA real index, 1997-.

Returns **National accounts, supply side, real index, SA**

Return type pd.DataFrame

econuy.retrieval.economic_activity.**natacc_ind_con_nsa_long**(*pipeline:* econuy.core.Pipeline = *None*) → pandas.core.frame.DataFrame
Get supply-side national accounts data in NSA constant prices, 1988-.

Three datasets with different base years, 1983, 2005 and 2016, are spliced in order to get to the result DataFrame.

Returns **National accounts, supply side, constant prices, NSA**

Return type pd.DataFrame

econuy.retrieval.economic_activity.**natacc_gas_con_nsa_long**(*pipeline:* econuy.core.Pipeline = *None*) → pandas.core.frame.DataFrame
Get demand-side national accounts data in NSA constant prices, 1988-.

Three datasets with different base years, 1983, 2005 and 2016, are spliced in order to get to the result DataFrame.

Returns **National accounts, demand side, constant prices, NSA**

Return type pd.DataFrame

econuy.retrieval.economic_activity.**gdp_con_idx_sa_long**(*pipeline:* econuy.core.Pipeline = *None*) → pandas.core.frame.DataFrame
Get demand-side national accounts data in NSA constant prices, 1988-.

Three datasets with different base years, 1983, 2005 and 2016, are spliced in order to get to the result DataFrame.

Returns **National accounts, demand side, constant prices, NSA**

Return type pd.DataFrame

econuy.retrieval.economic_activity.**gdp_con_nsa_long**(*pipeline:* econuy.core.Pipeline = *None*) → pandas.core.frame.DataFrame
Get GDP data in NSA constant prices, 1988-.

Three datasets with two different base years, 1983 and 2016, are spliced in order to get to the result DataFrame. It uses the BCU's working paper for retropolated GDP in current and constant prices for 1997-2015.

Returns **GDP, constant prices, NSA**

Return type pd.DataFrame

econuy.retrieval.economic_activity.**gdp_cur_nsa_long**(*pipeline:* econuy.core.Pipeline = *None*) → pandas.core.frame.DataFrame
Get GDP data in NSA current prices, 1997-.

It uses the BCU's working paper for retropolated GDP in current and constant prices for 1997-2015.

Returns **GDP, current prices, NSA**

Return type pd.DataFrame

econuy.retrieval.economic_activity.**industrial_production**() → pandas.core.frame.DataFrame
Get industrial production data.

Returns **Monthly industrial production index**

Return type pd.DataFrame

econuy.retrieval.economic_activity.**core_industrial**(*pipeline: Optional[econuy.core.Pipeline] = None*) → pandas.core.frame.DataFrame

Get total industrial production, industrial production excluding oil refinery and core industrial production.

Parameters **pipeline** (*econuy.core.Pipeline or None, default None*) – An instance of the econuy Pipeline class.

Returns **Measures of industrial production**

Return type pd.DataFrame

econuy.retrieval.economic_activity.**cattle**() → pandas.core.frame.DataFrame

Get weekly cattle slaughter data.

Returns **Weekly cattle slaughter**

Return type pd.DataFrame

econuy.retrieval.economic_activity.**milk**() → pandas.core.frame.DataFrame

Get monthly milk production in farms data.

Returns **Monhtly milk production in farms**

Return type pd.DataFrame

econuy.retrieval.economic_activity.**cement**() → pandas.core.frame.DataFrame

Get monthly cement sales data.

Returns **Monthly cement sales**

Return type pd.DataFrame

econuy.retrieval.economic_activity.**diesel**() → pandas.core.frame.DataFrame

Get diesel sales by department data.

This retrieval function requires the unrar binaries to be found in your system.

Returns **Monthly diesel dales**

Return type pd.DataFrame

econuy.retrieval.economic_activity.**gasoline**() → pandas.core.frame.DataFrame

Get gasoline sales by department data.

This retrieval function requires the unrar binaries to be found in your system.

Returns **Monthly gasoline dales**

Return type pd.DataFrame

econuy.retrieval.economic_activity.**electricity**() → pandas.core.frame.DataFrame

Get electricity sales by sector data.

This retrieval function requires the unrar binaries to be found in your system.

Returns **Monthly electricity dales**

Return type pd.DataFrame

econuy.retrieval.prices.**cpi**() → pandas.core.frame.DataFrame

Get CPI data.

Returns **Monthly CPI**

Return type pd.DataFrame

econuy.retrieval.prices.**cpi_divisions**() → pandas.core.frame.DataFrame

Get CPI data by division.

Returns **Monthly CPI by division**

Return type pd.DataFrame

econuy.retrieval.prices.**cpi_classes**() → pandas.core.frame.DataFrame
Get CPI data by division, group and class.

Returns **Monthly CPI by division, group and class**

Return type pd.DataFrame

econuy.retrieval.prices.**inflation_expectations**() → pandas.core.frame.DataFrame
Get data for the BCU inflation expectations survey.

Returns **Monthly inflation expectations**

Return type pd.DataFrame

econuy.retrieval.prices.**utilities**() → pandas.core.frame.DataFrame
Get prices for government-owned utilities.

Returns **Monthly utilities prices**

Return type pd.DataFrame

econuy.retrieval.prices.**ppi**() → pandas.core.frame.DataFrame
Get PPI data.

Returns **Monthly PPI**

Return type pd.DataFrame

econuy.retrieval.prices.**nxr_monthly**() → pandas.core.frame.DataFrame
Get monthly nominal exchange rate data.

Returns **Monthly nominal exchange rates** – Sell rate, monthly average and end of period.

Return type pd.DataFrame

econuy.retrieval.prices.**nxr_daily**(*pipeline: Optional[econuy.core.Pipeline] = None*, *previous_data: pandas.core.frame.DataFrame = Empty DataFrame Columns: [] Index: []*) → pandas.core.frame.DataFrame
Get daily nominal exchange rate data.

Parameters

- **pipeline** (*econuy.core.Pipeline or None, default None*) – An instance of the econuy Pipeline class.
- **previous_data** (*pd.DataFrame*) – A DataFrame representing this dataset used to extract last available dates.

Returns **Monthly nominal exchange rates** – Sell rate, monthly average and end of period.

Return type pd.DataFrame

econuy.retrieval.prices.**cpi_measures**(*pipeline: Optional[econuy.core.Pipeline] = None*) → pandas.core.frame.DataFrame
Get core CPI, Winsorized CPI, tradabe CPI, non-tradable CPI and residual CPI.

Parameters **pipeline** (*econuy.core.Pipeline or None, default None*) – An instance of the econuy Pipeline class.

Returns **Monthly CPI measures**

Return type pd.DataFrame

econuy.retrieval.fiscal_accounts.**balance_gps**() → pandas.core.frame.DataFrame
Get fiscal balance data for the consolidated public sector.

Returns **Monthly fiscal balance for the consolidated public sector**

Return type pd.DataFrame

econuy.retrieval.fiscal_accounts.**balance_nfps**() → pandas.core.frame.DataFrame
Get fiscal balance data for the non-financial public sector.

Returns **Monthly fiscal balance for the non-financial public sector**

Return type pd.DataFrame

econuy.retrieval.fiscal_accounts.**balance_cg_bps**() → pandas.core.frame.DataFrame
Get fiscal balance data for the central government + BPS.

Returns **Monthly fiscal balance for the central government + BPS**

Return type pd.DataFrame

econuy.retrieval.fiscal_accounts.**balance_pe**() → pandas.core.frame.DataFrame
Get fiscal balance data for public enterprises.

Returns **Monthly fiscal balance for public enterprises**

Return type pd.DataFrame

econuy.retrieval.fiscal_accounts.**balance_ancap**() → pandas.core.frame.DataFrame
Get fiscal balance data for ANCAP.

Returns **Monthly fiscal balance for ANCAP**

Return type pd.DataFrame

econuy.retrieval.fiscal_accounts.**balance_ute**() → pandas.core.frame.DataFrame
Get fiscal balance data for UTE.

Returns **Monthly fiscal balance for UTE**

Return type pd.DataFrame

econuy.retrieval.fiscal_accounts.**balance_antel**() → pandas.core.frame.DataFrame
Get fiscal balance data for ANTEL.

Returns **Monthly fiscal balance for ANTEL**

Return type pd.DataFrame

econuy.retrieval.fiscal_accounts.**balance_ose**() → pandas.core.frame.DataFrame
Get fiscal balance data for OSE.

Returns **Monthly fiscal balance for OSE**

Return type pd.DataFrame

econuy.retrieval.fiscal_accounts.**tax_revenue**() → pandas.core.frame.DataFrame
Get tax revenues data.

This retrieval function requires that Ghostscript and Tkinter be found in your system.

Returns **Monthly tax revenues**

Return type pd.DataFrame

econuy.retrieval.fiscal_accounts.**public_debt_gps**() → pandas.core.frame.DataFrame
Get public debt data for the consolidated public sector.

Returns **Quarterly public debt data for the consolidated public sector**

Return type pd.DataFrame

econuy.retrieval.fiscal_accounts.**public_debt_nfps**() → pandas.core.frame.DataFrame
Get public debt data for the non-financial public sector.

Returns **Quarterly public debt data for the non-financial public sector**

Return type pd.DataFrame

econuy.retrieval.fiscal_accounts.**public_debt_cb**() → pandas.core.frame.DataFrame
Get public debt data for the central bank

Returns **Quarterly public debt data for the central bank**

Return type pd.DataFrame

econuy.retrieval.fiscal_accounts.**public_assets**() → pandas.core.frame.DataFrame
Get public sector assets data.

Returns **Quarterly public sector assets**

Return type pd.DataFrame

econuy.retrieval.fiscal_accounts.**net_public_debt**(*pipeline: Optional[econuy.core.Pipeline] = None*) → pandas.core.frame.DataFrame
Get net public debt excluding deposits at the central bank.

Parameters **pipeline** (*econuy.core.Pipeline or None, default None*) – An instance of the econuy Pipeline class.

Returns **Net public debt excl. deposits at the central bank**

Return type pd.DataFrame

econuy.retrieval.fiscal_accounts.**balance_summary**(*pipeline: Optional[econuy.core.Pipeline] = None*) → pandas.core.frame.DataFrame
Get the summary fiscal balance table found in the Budget Law. Includes adjustments for the Social Security Fund.

Parameters **pipeline** (*econuy.core.Pipeline or None, default None*) – An instance of the econuy Pipeline class.

Returns **Summary fiscal balance table**

Return type pd.DataFrame

econuy.retrieval.external_sector.**trade_x_prod_val**() → pandas.core.frame.DataFrame
Get export values by product.

Returns **Export values by product**

Return type pd.DataFrame

econuy.retrieval.external_sector.**trade_x_prod_vol**() → pandas.core.frame.DataFrame
Get export volumes by product.

Returns **Export volumes by product**

Return type pd.DataFrame

econuy.retrieval.external_sector.**trade_x_prod_pri**() → pandas.core.frame.DataFrame
Get export prices by product.

Returns **Export prices by product**

Return type pd.DataFrame

econuy.retrieval.external_sector.**trade_x_dest_val**() → pandas.core.frame.DataFrame
Get export values by destination.

Returns **Export values by destination**

Return type pd.DataFrame

econuy.retrieval.external_sector.**trade_x_dest_vol**() → pandas.core.frame.DataFrame
Get export volumes by destination.

Returns **Export volumes by destination**

Return type pd.DataFrame

econuy.retrieval.external_sector.**trade_x_dest_pri**() → pandas.core.frame.DataFrame
Get export prices by destination.

Returns **Export prices by destination**

Return type pd.DataFrame

econuy.retrieval.external_sector.**trade_m_sect_val**() → pandas.core.frame.DataFrame
Get import values by sector.

Returns **Import values by sector**

Return type pd.DataFrame

econuy.retrieval.external_sector.**trade_m_sect_vol**() → pandas.core.frame.DataFrame
Get import volumes by sector.

Returns **Import volumes by sector**

Return type pd.DataFrame

econuy.retrieval.external_sector.**trade_m_sect_pri**() → pandas.core.frame.DataFrame
Get import prices by sector.

Returns **Import prices by sector**

Return type pd.DataFrame

econuy.retrieval.external_sector.**trade_m_orig_val**() → pandas.core.frame.DataFrame
Get import values by origin.

Returns **Import values by origin**

Return type pd.DataFrame

econuy.retrieval.external_sector.**trade_m_orig_vol**() → pandas.core.frame.DataFrame
Get import volumes by origin.

Returns **Import volumes by origin**

Return type pd.DataFrame

econuy.retrieval.external_sector.**trade_m_orig_pri**() → pandas.core.frame.DataFrame
Get import prices by origin.

Returns **Import prices by origin**

Return type pd.DataFrame

econuy.retrieval.external_sector.**trade_balance**(*pipeline: Optional[*econuy.core.Pipeline*] = None*) → pandas.core.frame.DataFrame
Get net trade balance data by country/region.

Parameters **pipeline** (econuy.core.Pipeline *or None, default None*) – An instance of the econuy Pipeline class.

Returns **Net trade balance value by region/country**

Return type pd.DataFrame

econuy.retrieval.external_sector.**terms_of_trade**(*pipeline: Optional[*econuy.core.Pipeline*] = None*) → pandas.core.frame.DataFrame

Get terms of trade.

Parameters **pipeline** (econuy.core.Pipeline *or None, default None*) – An instance of the econuy Pipeline class.

Returns **Terms of trade (exports/imports)**

Return type pd.DataFrame

econuy.retrieval.external_sector.**commodity_prices**() → pandas.core.frame.DataFrame

Get commodity prices for Uruguay.

Returns **Commodity prices** – Prices and price indexes of relevant commodities for Uruguay.

Return type pd.DataFrame

econuy.retrieval.external_sector.**commodity_index**(*pipeline: Optional[*econuy.core.Pipeline*] = None*) → pandas.core.frame.DataFrame

Get export-weighted commodity price index for Uruguay.

Parameters **pipeline** (econuy.core.Pipeline *or None, default None*) – An instance of the econuy Pipeline class.

Returns **Monthly export-weighted commodity index** – Export-weighted average of commodity prices relevant to Uruguay.

Return type pd.DataFrame

econuy.retrieval.external_sector.**rxr_official**() → pandas.core.frame.DataFrame

Get official (BCU) real exchange rates.

Returns **Monthly real exchange rates vs select countries/regions** – Available: global, regional, extraregional, Argentina, Brazil, US.

Return type pd.DataFrame

econuy.retrieval.external_sector.**rxr_custom**(*pipeline: Optional[*econuy.core.Pipeline*] = None*) → pandas.core.frame.DataFrame

Get custom real exchange rates vis-à-vis the US, Argentina and Brazil.

Parameters **pipeline** (econuy.core.Pipeline *or None, default None*) – An instance of the econuy Pipeline class.

Returns **Monthly real exchange rates vs select countries** – Available: Argentina, Brazil, US.

Return type pd.DataFrame

econuy.retrieval.external_sector.**bop**() → pandas.core.frame.DataFrame

Get balance of payments.

Returns **Quarterly balance of payments**

Return type pd.DataFrame

econuy.retrieval.external_sector.**bop_summary**(*pipeline: Optional[*econuy.core.Pipeline*] = None*) → pandas.core.frame.DataFrame

Get a balance of payments summary and capital flows calculations.

Returns **Quarterly balance of payments summary**

Return type pd.DataFrame

econuy.retrieval.external_sector.**reserves**() → pandas.core.frame.DataFrame
Get international reserves data.

Returns **Daily international reserves**

Return type pd.DataFrame

econuy.retrieval.external_sector.**reserves_changes**(*pipeline: Optional[econuy.core.Pipeline] = None*, *previous_data: pandas.core.frame.DataFrame = Empty DataFrame Columns: [] Index: []*) → pandas.core.frame.DataFrame
Get international reserves changes data.

Parameters

- **pipeline** (econuy.core.Pipeline *or None, default None*) – An instance of the econuy Pipeline class.
- **previous_data** (*pd.DataFrame*) – A DataFrame representing this dataset used to extract last available dates.

Returns **Daily international reserves changes**

Return type pd.DataFrame

econuy.retrieval.labor.**labor_rates**() → pandas.core.frame.DataFrame
Get labor market data (LFPR, employment and unemployment).

Returns **Monthly participation, employment and unemployment rates**

Return type pd.DataFrame

econuy.retrieval.labor.**nominal_wages**() → pandas.core.frame.DataFrame
Get nominal general, public and private sector wages data

Returns **Monthly wages separated by public and private sector**

Return type pd.DataFrame

econuy.retrieval.labor.**hours**() → pandas.core.frame.DataFrame
Get average hours worked data.

Returns **Monthly hours worked**

Return type pd.DataFrame

econuy.retrieval.labor.**rates_people**(*pipeline: Optional[econuy.core.Pipeline] = None*) → pandas.core.frame.DataFrame
Get labor data, both rates and persons. Extends national data between 1991 and 2005 with data for jurisdictions with more than 5,000 inhabitants.

Parameters **pipeline** (econuy.core.Pipeline *or None, default None*) – An instance of the econuy Pipeline class.

Returns **Labor market data**

Return type pd.DataFrame

econuy.retrieval.labor.**real_wages**(*pipeline: Optional[econuy.core.Pipeline] = None*) → pandas.core.frame.DataFrame
Get real wages.

Parameters **pipeline** (*econuy.core.Pipeline or None, default None*) – An instance of the econuy Pipeline class.

Returns **Real wages data**

Return type pd.DataFrame

econuy.retrieval.financial_sector.**credit**() → pandas.core.frame.DataFrame
Get bank credit data.

Returns **Monthly credit**

Return type pd.DataFrame

econuy.retrieval.financial_sector.**deposits**() → pandas.core.frame.DataFrame
Get bank deposits data.

Returns **Monthly deposits**

Return type pd.DataFrame

econuy.retrieval.financial_sector.**interest_rates**() → pandas.core.frame.DataFrame
Get interest rates data.

Returns **Monthly interest rates**

Return type pd.DataFrame

econuy.retrieval.financial_sector.**sovereign_risk**() → pandas.core.frame.DataFrame
Get Uruguayan Bond Index (sovereign risk spreads) data.

Returns **Uruguayan Bond Index**

Return type pd.DataFrame

econuy.retrieval.financial_sector.**call_rate**(*driver: Optional[selenium.webdriver.remote.webdriver.WebDriver] = None*) → pandas.core.frame.DataFrame
Get 1-day call interest rate data.

This function requires a Selenium webdriver. It can be provided in the driver parameter, or it will attempt to configure a Chrome webdriver.

Parameters **driver** (*selenium.webdriver.chrome.webdriver.WebDriver, default None*) – Selenium webdriver for scraping. If None, build a Chrome webdriver.

Returns **Daily call rate**

Return type pd.DataFrame

econuy.retrieval.financial_sector.**bonds**(*driver: Optional[selenium.webdriver.remote.webdriver.WebDriver] = None*) → pandas.core.frame.DataFrame
Get interest rate yield for Uruguayan US-denominated bonds, inflation-linked bonds and peso bonds.

This function requires a Selenium webdriver. It can be provided in the driver parameter, or it will attempt to configure a Chrome webdriver.

Parameters **driver** (*selenium.webdriver.chrome.webdriver.WebDriver, default None*) – Selenium webdriver for scraping. If None, build a Chrome webdriver.

Returns **Daily bond yields in basis points**

Return type pd.DataFrame

econuy.retrieval.income.**income_household**() → pandas.core.frame.DataFrame

Get average household income.

Parameters

- **update_loc** (*str, os.PathLike, SQLAlchemy Connection or Engine, or None, default None*) – Either Path or path-like string pointing to a directory where to find a CSV for updating, SQLAlchemy connection or engine object, or None, don't update.
- **revise_rows** (*{'nodup', 'auto', int}*) – Defines how to process data updates. An integer indicates how many rows to remove from the tail of the dataframe and replace with new data. String can either be auto, which automatically determines number of rows to replace from the inferred data frequency, or nodup, which replaces existing periods with new data.
- **save_loc** (*str, os.PathLike, SQLAlchemy Connection or Engine, or None, default None*) – Either Path or path-like string pointing to a directory where to save the CSV, SQL Alchemy connection or engine object, or None, don't save.
- **only_get** (*bool, default False*) – If True, don't download data, retrieve what is available from update_loc.

Returns **Monthly average household income**

Return type pd.DataFrame

econuy.retrieval.income.**income_capita**(*update_loc: Optional[Union[str, os.PathLike, sqlalchemy.engine.base.Engine, sqlalchemy.engine.base.Connection]] = None*, *revise_rows: Union[str, int] = 'nodup'*, *save_loc: Optional[Union[str, os.PathLike, sqlalchemy.engine.base.Engine, sqlalchemy.engine.base.Connection]] = None*, *only_get: bool = False*) → pandas.core.frame.DataFrame

Get average per capita income.

Parameters

- **update_loc** (*str, os.PathLike, SQLAlchemy Connection or Engine, or None, default None*) – Either Path or path-like string pointing to a directory where to find a CSV for updating, SQLAlchemy connection or engine object, or None, don't update.
- **revise_rows** (*{'nodup', 'auto', int}*) – Defines how to process data updates. An integer indicates how many rows to remove from the tail of the dataframe and replace with new data. String can either be auto, which automatically determines number of rows to replace from the inferred data frequency, or nodup, which replaces existing periods with new data.
- **save_loc** (*str, os.PathLike, SQLAlchemy Connection or Engine, or None, default None*) – Either Path or path-like string pointing to a directory where to save the CSV, SQL Alchemy connection or engine object, or None, don't save.
- **only_get** (*bool, default False*) – If True, don't download data, retrieve what is available from update_loc.

Returns **Monthly average per capita income**

Return type pd.DataFrame

econuy.retrieval.income.**consumer_confidence**(*update_loc: Optional[Union[str, os.PathLike, sqlalchemy.engine.base.Engine, sqlalchemy.engine.base.Connection]] = None*, *revise_rows: Union[str, int] = 'nodup'*, *save_loc: Optional[Union[str, os.PathLike, sqlalchemy.engine.base.Engine, sqlalchemy.engine.base.Connection]] = None*, *only_get: bool = False*) → pandas.core.frame.DataFrame

Get monthly consumer confidence data.

Parameters

- **update_loc** (*str, os.PathLike, SQLAlchemy Connection or Engine, or None, default None*) – Either Path or path-like string pointing to a directory where to find a CSV for updating, SQLAlchemy connection or engine object, or `None`, don't update.
- **revise_rows** (*{'nodup', 'auto', int}*) – Defines how to process data updates. An integer indicates how many rows to remove from the tail of the dataframe and replace with new data. String can either be `auto`, which automatically determines number of rows to replace from the inferred data frequency, or `nodup`, which replaces existing periods with new data.
- **save_loc** (*str, os.PathLike, SQLAlchemy Connection or Engine, or None, default None*) – Either Path or path-like string pointing to a directory where to save the CSV, SQL Alchemy connection or engine object, or `None`, don't save.
- **only_get** (*bool, default False*) – If True, don't download data, retrieve what is available from `update_loc`.

Returns **Monthly consumer confidence data**

Return type pd.DataFrame

econuy.retrieval.international.**gdp**() → pandas.core.frame.DataFrame

Get seasonally adjusted real quarterly GDP for select countries.

Countries/aggregates are US, EU-27, Japan and China.

Returns **Quarterly real GDP in seasonally adjusted terms**

Return type pd.DataFrame

econuy.retrieval.international.**stocks**() → pandas.core.frame.DataFrame

Get stock market index data.

Indexes selected are S&P 500, Euronext 100, Nikkei 225 and Shanghai Composite.

Returns **Daily stock market index in USD**

Return type pd.DataFrame

econuy.retrieval.international.**policy_rates**() → pandas.core.frame.DataFrame

Get central bank policy interest rates data.

Countries/aggregates selected are US, Euro Area, Japan and China.

Returns **Daily policy interest rates**

Return type pd.DataFrame

econuy.retrieval.international.**long_rates**() → pandas.core.frame.DataFrame

Get 10-year government bonds interest rates.

Countries/aggregates selected are US, Germany, France, Italy, Spain United Kingdom, Japan and China.

Returns **Daily 10-year government bonds interest rates**

Return type pd.DataFrame

econuy.retrieval.international.**nxr**() → pandas.core.frame.DataFrame
Get currencies data.

Selected currencies are the US dollar index, USDEUR, USDJPY and USDCNY.

Returns **Daily currencies**

Return type pd.DataFrame

econuy.retrieval.regional.**gdp**(*driver: selenium.webdriver.remote.webdriver.WebDriver = None*) → pandas.core.frame.DataFrame
Get seasonally adjusted real GDP for Argentina and Brazil.

This function requires a Selenium webdriver. It can be provided in the driver parameter, or it will attempt to configure a Chrome webdriver.

Parameters **driver** (*selenium.webdriver.chrome.webdriver.WebDriver, default None*) – Selenium webdriver for scraping. If None, build a Chrome webdriver.

Returns **Quarterly real GDP**

Return type pd.DataFrame

econuy.retrieval.regional.**monthly_gdp**() → pandas.core.frame.DataFrame
Get monthly GDP data.

Countries/aggregates selected are Argentina and Brazil.

Returns **Daily policy interest rates**

Return type pd.DataFrame

econuy.retrieval.regional.**cpi**() → pandas.core.frame.DataFrame
Get consumer price index for Argentina and Brazil.

Returns **Monthly CPI**

Return type pd.DataFrame

econuy.retrieval.regional.**embi_spreads**() → pandas.core.frame.DataFrame
Get EMBI spread for Argentina, Brazil and the EMBI Global.

Returns **Daily 10-year government bond spreads**

Return type pd.DataFrame

econuy.retrieval.regional.**embi_yields**(*pipeline: Optional[econuy.core.Pipeline] = None*) → pandas.core.frame.DataFrame
Get EMBI yields for Argentina, Brazil and the EMBI Global.

Yields are calculated by adding EMBI spreads to the 10-year US Treasury bond rate.

Parameters **pipeline** (*econuy.core.Pipeline or None, default None*) – An instance of the econuy Pipeline class.

Returns **Daily 10-year government bonds interest rates**

Return type pd.DataFrame

econuy.retrieval.regional.**nxr**() → pandas.core.frame.DataFrame
Get USDARS and USDBRL.

Returns **Daily exchange rates**

Return type pd.DataFrame

econuy.retrieval.regional.**policy_rates**() → pandas.core.frame.DataFrame
Get central bank policy interest rates data.

Countries/aggregates selected are Argentina and Brazil.

Returns **Daily policy interest rates**

Return type pd.DataFrame

econuy.retrieval.regional.**stocks**(*pipeline: Optional[*econuy.core.Pipeline*] = None*) → pandas.core.frame.DataFrame
Get stock market index data in USD terms.

Indexes selected are MERVAL and BOVESPA.

Parameters **pipeline** (*econuy.core.Pipeline or None, default None*) – An instance of the econuy Pipeline class.

Returns **Daily stock market index in USD terms**

Return type pd.DataFrame

econuy.retrieval.regional.**rxr**(*pipeline: Optional[*econuy.core.Pipeline*] = None*) → pandas.core.frame.DataFrame
Get real exchange rates vis-á-vis the US dollar for Argentina and Brasil .

Returns **Monthly real exchange rate**

Return type pd.DataFrame

2.1.4 Transformation functions

econuy.transform.**convert_usd**(*df: pandas.core.frame.DataFrame*, *pipeline=None*, *errors: str = 'raise'*) → pandas.core.frame.DataFrame
Convert to other units.

See also:

convert

econuy.transform.**convert_real**(*df: pandas.core.frame.DataFrame*, *start_date: Optional[Union[str, datetime.datetime]] = None*, *end_date: Optional[Union[str, datetime.datetime]] = None*, *pipeline=None*, *errors: str = 'raise'*) → pandas.core.frame.DataFrame
Convert to other units.

See also:

convert

econuy.transform.**convert_gdp**(*df: pandas.core.frame.DataFrame*, *pipeline=None*, *errors: str = 'raise'*) → pandas.core.frame.DataFrame
Convert to other units.

See also:

convert

econuy.transform.**resample**(*df: pandas.core.frame.DataFrame*, *rule: Union[pandas._libs.tslibs.offsets.DateOffset, pandas._libs.tslibs.timedeltas.Timedelta, str]*, *operation: str = 'sum'*, *interpolation: str = 'linear'*, *warn: bool = False*) → pandas.core.frame.DataFrame
Resample to target frequencies.

See also:

resample

econuy.transform.**rolling**(*df: pandas.core.frame.DataFrame*, *window: Optional[int] = None*, *operation: str = 'sum'*) → pandas.core.frame.DataFrame

Calculate rolling averages or sums.

See also:

rolling

econuy.transform.**rebase**(*df: pandas.core.frame.DataFrame*, *start_date: Union[str, datetime.datetime]*, *end_date: Optional[Union[str, datetime.datetime]] = None*, *base: Union[int, float] = 100.0*) → pandas.core.frame.DataFrame

Scale to a period or range of periods.

See also:

rebase

econuy.transform.**decompose**(*df: pandas.core.frame.DataFrame*, *component: str = 'both'*, *method: str = 'x13'*, *force_x13: bool = False*, *fallback: str = 'loess'*, *outlier: bool = True*, *trading: bool = True*, *x13_binary: Optional[Union[str, os.PathLike]] = 'search'*, *search_parents: int = 0*, *ignore_warnings: bool = True*, *errors: str = 'raise'*, ***kwargs*) → Union[Dict[str, pandas.core.frame.DataFrame], pandas.core.frame.DataFrame]

Apply seasonal decomposition.

By default returns both trend and seasonally adjusted components, unlike the class method referred below.

See also:

decompose

econuy.transform.**chg_diff**(*df: pandas.core.frame.DataFrame*, *operation: str = 'chg'*, *period: str = 'last'*) → pandas.core.frame.DataFrame

Calculate pct change or difference.

See also:

chg_diff

econuy.transform.**error_handler**(*df: pandas.core.frame.DataFrame*, *errors: str*, *msg: Optional[str] = None*) → pandas.core.frame.DataFrame

2.1.5 Utility functions

econuy.utils.sqlutil.**read**(*con: sqlalchemy.engine.base.Connection*, *command: Optional[str] = None*, *table_name: Optional[str] = None*, *cols: Optional[Union[str, Iterable[str]]] = None*, *start_date: Optional[str] = None*, *end_date: Optional[str] = None*, ***kwargs*) → pandas.core.frame.DataFrame

Convenience wrapper around pandas.read_sql_query.

Deals with multiindex column names.

Parameters

- **con** (*sqlalchemy.engine.base.Connection*) – Connection to SQL database.
- **command** (*str, sqlalchemy.sql.Selectable or None, default None*) – Command to pass to pandas.read_sql_query. If this parameter is not None, *table*, *cols*, *start_date* and *end_date* will be ignored.

- **table_name** (*str or None, default None*) – String representing which table should be retrieved from the database.
- **cols** (*str, iterable or None, default None*) – Column(s) to retrieve. By default, gets all all columns.
- **start_date** (*str or None, default None*) – Dates to filter. Inclusive.
- **end_date** (*str or None, default None*) – Dates to filter. Inclusive.
- ****kwargs** – Keyword arguments passed to pandas.read_sql_query.

Returns **SQL queried table**

Return type pd.DataFrame

econuy.utils.sqlutil.**df_to_sql**(*df: pandas.core.frame.DataFrame*, *name: str*, *con: sqlalchemy.engine.base.Connection*, *if_exists: str = 'replace'*) → None

Flatten MultiIndex index columns before creating SQL table from dataframe.

econuy.utils.sqlutil.**insert_csvs**(*con: sqlalchemy.engine.base.Connection*, *directory: Union[str, pathlib.Path, os.PathLike]*) → None

Insert all CSV files in data directory into a SQL database.

PYTHON MODULE INDEX

e

INDEX

E

F

G

H

I

L

M

N

P

R

S

T

U