py_wlc Documentation

Release 0.0.1

Jonathan Sharpe

December 21, 2015

Contents

Contents:

CHAPTER 1

py_wlc

1.1 py_wlc package

1.1.1 Subpackages

py_wlc.data package

Submodules

py_wlc.data.webtag_data module

Exposes the parsed WebTAG data as *py_wlc* objects.

class py_wlc.data.webtag_data.**WebTagData**(*base_year*, *released*, *version*, *source*, ***data*)
Bases: object

Holds the data extracted from WebTAG.

classmethod from_json(*file*)
Extract data from the specified JSON.

Parameters **file** (str) – The file to import from.

Returns class:~.*WebTagData*: A new class instance.

Return type :py

classmethod from_latest_json(*dir_*)
Extract data from the most recent JSON in the directory.

Parameters **dir** (str) – The directory to start searching from.

Returns class:~.*WebTagData*: A new class instance.

Return type :py

py_wlc.data.webtag_parser module

WebTAG Parser - functionality for extracting from the Data Book.

This class parses the databook to a more convenient JSON format for use by *WebTagData*.

class py_wlc.data.webtag_parser.**WebTagParser**(*filename*)

Bases: object

Class to handle access to a WebTAG Databook Excel file.

The class is designed to operate as a context manager if needed.

Where possible, the workbook is opened in on_demand mode, to avoid loading all worksheets at once. *extract_data()* will load and unload the appropriate worksheets as required.

Parameters **filename** (str) – The WebTAG Databook file to open.

book

xlrd.Workbook

The Excel workbook.

date

datetime.datetime

The release date of the databook.

filename

str

The name of the file to open.

version

str

The version of the databook.

BASE = ('User Parameters', 0, 11, 'Price year')

Where to find the base year.

CHECK = ('Cover', 2, 0, 'WebTAG Databook')

Defines the check for a valid WebTAG workbook.

DATE = ('Audit', 0, 2)

Where to locate the workbook date.

LOCATIONS = {'rail_electricity_price': ('A1.3.7', 27, 1, 7), 'discount_rate': ('A1.1.1', 24, 1, 3), 'rail_diesel_price': ('A1.3.

Locations of defined data series for extraction.

VERSION = ('Cover', 3, 0)

Where to locate the workbook version.

close()

Release the *book* resources.

extract_all(*verbose=False*)

Extract all data from LOCATIONS and useful metadata.

Parameters **verbose** (bool, optional) – Whether to report progress. Implemented primarily for *cli()* usage. Defaults to False.

Returns The data extracted from the *book*.

Return type dict

extract_data(*sheet_name*, *start_row*, *key_col*, *value_col*)

Extract data from the specified worksheet.

Assumes that cell A3 contains the worksheet title and that cell A4 contains the table name.

Parameters

- **sheet_name** (str) – The name of the worksheet.
- **start_row** (int) – The first row to extract data from.
- **key_col** (int) – The column to extract keys from.
- **value_col** (int) – The column to extract values from.

Returns The extracted data

Return type dict

extract_named_data(*name*)
Extract a named data series from LOCATIONS.

Parameters **name** (str) – The name of the data series (must be in Locations).

Returns The extracted data series.

Return type dict

Raises KeyError – If name is not in Locations.

py_wlc.data.webtag_parser.**cli**(*args*)
Provide a CLI for the *WebTagParser*.

Will either output to a specified file (with optional verbose reporting) or dump the JSON data to stdout.

Parameters **args** (argparse.Namespace) – The parsed command line arguments.

Raises ValueError – If -v is supplied without -o.

py_wlc.data.webtag_parser.**parse_args**(*args*)
Parse the arguments for *cli()*.

Parameters **args** (list of str) – Arguments from command line.

Returns The parsed arguments.

Return type argparse.Namespace

Raises ArgumentError – If -v is supplied without -o.

Module contents

py_wlc.data provides external data handling functionality.

Various sources of external data may be required for carrying out economic appraisals; this module provides convenience functions and classes for handling that data.

py_wlc.economics package

Submodules

py_wlc.economics.cost module

Representation of cost objects in various bases and forms.

class py_wlc.economics.cost.**Cost**(*value*, *type_*, *year*, *discount*, *deflator*, *adjustment_factor*)
Bases: object

Represents costs and holds data for type conversions.

The `type_` of a cost is necessary for conversion between the different types; costs can be provide in real or nominal terms and as a factor (or "resource") cost or a market price. Additionally, a cost can be a Present Value (discounted real factor cost or market price).

It is essential for an accurate calculation that the appropriate cost type is used for conversion to consistent output values.

Note: The default assumptions for `type_` are *NOMINAL* and *FACTOR_COST*. For example:

```
Cost(100, Cost.REAL, ...)
```

is equivalent to:

```
Cost(100, Cost.REAL | Cost.FACTOR_COST, ...)
```

The exception is *PRESENT_VALUE*, which is always a real cost.

Parameters

- **value** (`float`) – The value of the cost.
- **type** (`int`) – The type of the cost.
- **year** (`int`) – The year in which the cost is incurred.
- **discount** (*Discount*) – The discount factors to use for conversion to Present Value.
- **deflator** (*GdpDeflator*) – The GDP deflator factors to use for conversion to real prices.
- **adjustment_factor** (`float`) – The factor to use for conversion to market prices.

cost
`float`

The nominal factor cost.

year
`int`

The year in which the cost is incurred.

discount_factor
`float`

The factor for conversion to Present Value (from real factor costs or market prices).

deflation_factor
`float`

The factor for conversion to real prices (from nominal prices).

adjustment_factor
`float`

The factor for conversion to market prices (from factor costs).

Raises `ValueError` – If the `type_` argument is invalid.

FACTOR_COST = 1
Factor cost, excluding taxation.

MARKET_PRICE = 2
Market price, including taxation.

NOMINAL = 4
Nominal price, as paid in the year incurred.

PRESENT_VALUE = 16
Discounted real costs.

REAL = 8
Real price, in a constant price base year.

RESOURCE_COST = 1
Synonym of *FACTOR_COST*.

as_type(*type_*)
Convert the nominal factor cost to the specified `type_`.

Parameters **type** (`int`) – The type to convert to.

Returns The converted value.

Return type float

classmethod validate_type(*type_*)
Validate a cost type argument.

Costs cannot be both market price and factor/resource cost, or both nominal and real. Present values are necessarily discounted real factor costs or market prices.

Parameters **type** (`int`) – The type of the cost.

Raises `ValueError` – If the `type_` is invalid.

py_wlc.economics.discount module

discount enables the calculation of Present Value.

class py_wlc.economics.discount.**Discount**(*base_year*, *rates=None*, *year_zero=None*)
Bases: *py_wlc.generic.growth.IndexSeries*

Discount rates and factors for calculating Present Value.

The discount rate is assumed to be zero prior to `year_zero`, and remain at the rate corresponding to the last year in `self.rates` indefinitely. Similarly, the factor is assumed to be `1.0` prior to the base year.

The initial rate, i.e. the rate corresponding to the smallest year in the rates dictionary, is assumed to be the rate from the `base_year` onwards.

Parameters

- **base_year** (`int`) – The base year for discounting, i.e. the year in which the factor is `1.0`.
- **rates** (`dict` of `int`: `float`, optional) – The discount rates, where the key is the start year and the value is the rate to apply. Defaults to HM Treasury Green Book rates.
- **year_zero** (`int`, optional) – Year zero, the year from which the applicable rate is incremented. Defaults to `base_year`.

RATES = {0: 0.035, 201: 0.015, 76: 0.025, 301: 0.01, 126: 0.02, 31: 0.03}
Default HM Treasury "Green Book" discount rates.

rate(*year*)
The rate used in the specified year.

Parameters **year** (int) – The year to retrieve the rate for.

Returns The rate used in that year.

Return type float

rebase(*year_zero*)
Create a *Discount* with new year_zero.

Parameters **year_zero** (int) – The new year_zero to use.

Returns class:~.*Discount*: The new *Discount* object.

Return type :py

py_wlc.economics.gdp_deflator module

Provides conversion between nominal and real prices.

class py_wlc.economics.gdp_deflator.**GdpDeflator**(*base_year*, *rates*, *extend=False*)
Bases: *py_wlc.generic.growth.IndexSeries*

If extension beyond the specified *_rates* is required, the rates argument is replaced with an *ExtendedDict*, such that the first and last rates are continued indefinitely. Otherwise, there is no growth outside the predefined data-set (i.e. assumed rate of 0.0).

Parameters

- **base_year** (int) – The price base year to deflate to.
- **rates** (dict of int: str) – The annual rates.
- **extend** (bool, optional) – Whether or not to extend the rates beyond the predefined data. Defaults to False.

conversion_factor(*year_from*, *year_to=None*)
Calculate the factor to convert costs between two years.

Parameters

- **year_from** (int) – The year to convert from.
- **year_to** (int, optional) – The year to convert to. Defaults to *base_year*.

Returns The conversion factor between the two years.

Return type float

py_wlc.economics.residual_value module

Calculation methods for residual value of assets.

class py_wlc.economics.residual_value.**ResidualValueCalculator**(*method*)
Bases: object

A calculator to generate residual values of assets.

Parameters **method** (str) – The method to use for the calculation (must be in *METHODS*, see Wikipedia for method details).

Raises ValueError – If method is not in *METHODS*.

METHODS = {'linear': 'linear', "sum of years' digits": 'sum_of_years', 'double-declining': 'double_declining'}
Available methods for calculation of residual value.

classmethod available_methods()
Show the methods available in *METHODS*.

Returns The available methods.

Return type list of str

calculate(*value*, *life*, *build_year*, *target_year*, *scrap_value=0.0*)
Calculate residual value, using selected *method*.

Residual value is assumed to be the scrap_value after the life expires (build_year + life). The residual value is never allowed to fall below the scrap_value.

Parameters

- **value** (float) – The initial asset value.
- **life** (int) – The life of the asset, in years.
- **build_year** (int) – The year in which the asset is built.
- **target_year** (int) – The year in which to calculate the asset's residual value.
- **scrap_value** (float, optional) – The asset's value after life expiry. Defaults to *0.0*.

Returns The calculated residual value.

Return type float

Raises ValueError – If target_year precedes the build_year.

static double_declining(*value*, *life*, *build_year*, *target_year*, *scrap_value*)
Calculate residual value with double-declining method.

The double-declining method, a specific declining balance method, assumes that the same proportion of the remaining value is lost in each year of the asset's life. In this case, the proportion is double the proportion lost in each year under the *linear()* method.

Parameters

- **value** (float) – The initial asset value.
- **life** (int) – The life of the asset, in years.
- **build_year** (int) – The year in which the asset is built.
- **target_year** (int) – The year in which to calculate the asset's residual value.
- **scrap_value** (float) – The asset's value after life expiry.

Returns The calculated residual value.

Return type float

static linear(*value*, *life*, *build_year*, *target_year*, *scrap_value*)
Calculate residual value with linear (straight-line) method.

The linear (or straight-line depreciation) method assumes that the same amount of the asset's value is lost in every year of its life.

Parameters

- **value** (float) – The initial asset value.
- **life** (int) – The life of the asset, in years.

- **build_year** (int) – The year in which the asset is built.
- **target_year** (int) – The year in which to calculate the asset's residual value.
- **scrap_value** (float) – The asset's value after life expiry.

Returns The calculated residual value.

Return type float

method
The current method for residual value calculations.

Returns The name of the current method (read-only).

Return type str

static sum_of_years(*value*, *life*, *build_year*, *target_year*, *scrap_value*)
Calculate residual value with sum of years' digits method.

The sum of years' digits method uses a "schedule of fractions" to depreciate the value, based on summing the digits of all years in the life for the denominator.

Parameters

- **value** (float) – The initial asset value.
- **life** (int) – The life of the asset, in years.
- **build_year** (int) – The year in which the asset is built.
- **target_year** (int) – The year in which to calculate the asset's residual value.
- **scrap_value** (float) – The asset's value after life expiry.

Returns The calculated residual value.

Return type float

py_wlc.economics.residual_value.**sum_of_years_digits**(*year*)
Calculate the sum of years' digits.

Parameters **year** (int) – The year to calculate up to.

Returns The sum of years' digits.

Return type int

Module contents

The *economics* module provides core economic models.

py_wlc.generic package

Submodules

py_wlc.generic.growth module

Generic functionality for modelling growth series.

class py_wlc.generic.growth.**IndexSeries**(*base_year*, *rates*, *initial_value*, *year_zero=None*)
Bases: object

Growth rates and factors for indexation series.

The _rates dictionary is fully-populated at initialisation, but the _values dictionary is filled lazily - values are only calculated as needed.

The class supports a Mapping-like interface; factors can be accessed with value = growth_rate[year] or value = growth_rate.get(year, default).

Note: The term 'relative year' refers to the year relative to year_zero e.g. 3. The term 'absolute year' refers to a calendar year, e.g. 2013. Relative years are used internally, absolute years for the external interface.

Parameters

- **base_year** (int) – the year in which the value is equal to the initial_value
- **rates** (dict of int: float) – the growth rates to use, keyed by relative year
- **initial_value** (float) – the first value for the output series.
- **year_zero** (int, optional) – the zeroth year for accessing growth rates. Defaults to *base_year*.

base_year
int

The base year for growth, i.e. the year in which the value is the initial_value.

year_zero
int

The zeroth year for growth, i.e. the year from which the rates are selected from _rates.

_rates
dict of int: float

The growth rates, where the key is the relative start year and the value is the rate to apply.

_initial_rate
float

The growth rate corresponding to the first year in the rates dictionary.

_final_rate
float

The growth rate corresponding to the last year in the rates dictionary.

_values
dict of int: float

The values, keyed by year.

get(*year*, *default=None*)
Retrieve value or supplied default for given year.

Parameters

- **year** (*int*) – The year to retrieve the value for.
- **default** (*float or None, optional*) – The value to return if retrieval fails. Defaults to None.

Returns The retrieved or `default` value.

Return type float or None

rate(*year*)
The rate used in the specified year.

Parameters **year** (*int*) – The year to retrieve the rate for.

Returns The rate used in that year.

Return type float

Module contents

Generic functionality supporting the core modelling.

class py_wlc.generic.**ExtendedDict**
Bases: `dict`

Dictionary subclass that provides values beyond defined keys.

Provides values outside the predefined range, according to the following rules:

- If the key is larger than the largest key in the dictionary, the value from the largest key is returned.
- If the key is smaller than the smallest key in the dictionary, the value from the smallest key is returned.
- If the key is between the smallest and largest keys in the dictionary but no value is found, a `KeyError` occurs.

copy()
Create a copy of the extended dictionary.

Returns A new class instance.

Return type *ExtendedDict*

get(*key*, *default=None*)
Return either the value of the `key`, or the `default`.

Parameters

- **key** – The key to return the value of.
- **default** – The object to return if the key is not found.

Returns Either the value for the specified `key`, or `default`.

py_wlc.utils package

Module contents

General useful functionality.

py_wlc.utils.**memo**(*func*)
Memoizing decorator for caching function results.

Note: Currently only implemented for hashable positional arguments.

Parameters **func** (`callable`) – The function to decorate.

Returns The decorated function.

Return type callable

1.1.2 Module contents

Functionality for whole-life costing in Python

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

p

Symbols

A

B

C

D

E

F

G

I

L