Pyfolio Performance

Fabian Bendun

Apr 23, 2023

CONTENTS:

Installation can be done through `pip install pyfolio-performance` or via Github.

CHAPTER

ONE

API

In the following, we have the documentation of the library's classes.

1.1 Portfolio

class pyfolio_performance.**Portfolio**(*filename*)

The main class to parse and access different aspects of a portfolio stored in a XML file.

Uses the XML file created by portfolio performance.

Parameters
filename (*str*) – The path of the XML file to parse.

evaluateCluster(*clusters*, *fn_filter*, *fn_getClusterId*, *fn_aggregation*)

Evaluates all transactions of the portfolio as follows. Every transaction that is successfully filtered by fn_filter, gets put in a cluster through fn_getClusterId. The objects in the cluster are aggregated through the fn_aggregation function.

Parameters

- **clusters** (*dict(object) / {k->v}*) – The overall clusters.
- **fn_filter** (*function(transaction) -> bool*) – Filter function. An entry needs to pass the filter with True to be considered.
- **fn_getClusterId** (*function({k->v}, Transaction) -> k*) – Given the cluster and the transaction, this method gives the key to the cluster the transaction belongs to.
- **fn_aggregation** (*function(v, Transaction) -> v*) – The aggregation function that combines cluster values. This updates the cluster itself at the position cluster-id for every considered transaction.

Returns
Nothing is returned.

Type
None

getAccounts()

Returns the list of Account objects in the portfolio.

Returns
The extracted Account list.

Type
list(*Account*)

`getDepots()`

Returns the list of Depot objects in the portfolio.

Returns
The extracted Depot list.

Type
list(*Depot*)

`getInvestmentInto`(*security*, *before=None*)

Computes how much is invested into a specific security before a given date. If no date is given, the total investment is calculated.

Returns
value in cents of investement

Type
int

`getSecurities()`

Returns the list of all unique securities in any depot. :return: The list. :type: list(Security)

`getShares`(*theSecurity*)

Returns the number of shares that the given security objects has in the portfolio overall.

Parameters
theSecurity (Security) – The security queried.

Returns
The number of shares in all depots summed up.

Type
float

`getTotalTransactions()`

Returns the list of all transactions in the portfolio across all depots and accounts.

Returns
The extracted transaction list.

Type
list(Transaction)

1.2 Account

`class pyfolio_performance.Account`

The class that manages a money account and its transactions.

`getBalance()`

Returns
Balance of the account in cents.

Type
int

`getName()`

Returns
Name of the account.

Type
str

`getTransactions()`

Returns
list of transactions in the account.

Type
list(Transaction)

1.3 Depot

`class pyfolio_performance.Depot(name, xml)`

The class that manages a depot and its transactions.

`static getDepotByName(name)`

If no such Depot exists, it returns an empty depot with the name. If it exists, it returns the corresponding Depot.

Param
Name of the depot that should be returned

Type
str

Returns
Existing or new Depot

Type
Depot

`getName()`

Returns
Name of the depot.

Type
str

`getSecurities()`

Returns
Mapping of currently Securities to the number of contained shares

Type
dict(Security -> float)

`getTransactions()`

Returns
list of transactions in the depot.

Type
list(Transaction)

1.4 Security

class pyfolio_performance.**Security**(*xml*)

A class that manages securities.

getMostRecentValue()

Returns
Current security price from the file in Euro.

Type
float

getName() → str

Returns
Name of the security

Type
str

static getSecurityByIsin(*isin*)

Param
Isin of security that should be returned.

Type
str

Returns
existing security object or None

Type
Security

static getSecurityByName(*name*)

Param
Name of security that should be returned.

Type
str

Returns
existing security object or None

Type
Security

static getSecurityByWkn(*wkn*)

Param
Wkn of security that should be returned.

Type
str

Returns
existing security object or None

Type
Security

1.5 Portfolio Performance Object

class pyfolio_performance.**PortfolioPerformanceObject**

Base class for most objects in the library. Offers basic functionality needed, such as: - cache of already parsed objects, - resolution of objects that are defined by references, - general parsing methods.

classmethod getObjectByAttribute(*attr*, *value*)

Note it only works if there is a single object for the attribute and the value. For example, we can ask for the attribute *isin* of a security with the value *DE0005190003* leading to BMW.

Parameters

- **attr** (*str*) – the attribute we are looknig for
- **value** (*str*) – the value the attribute should have

Returns
the store object for the value

Type
object

classmethod parse(*root*, *xml*)

This methods parses portfolio performance objects. It returns the parsed result of the referenced xml.

Parameters

- **root** (*xml*) – Root of the parsing, in case it is needed to resolve references.
- **xml** (*xml*) – Object to be parsed.

Returns
Parsed object.

Type
Subclass of *PortfolioPerformanceObject*

classmethod parseByReference(*root*, *reference*)

This methods resolves the attribute referenced. It returns the parsed result of the referenced xml.

Parameters

- **root** (*xml*) – Root from where the reference is searched in the XML.
- **reference** (*str*) – Encoding of the reference

Returns
Parsed object.

Type
Subclass of *PortfolioPerformanceObject*

1.6 Date Object

class pyfolio_performance.**DateObject**(*dateStr*)

Represents a data of a transaction.

Parameters
dateStr – Date string as used by portfolio performance in the XML.

getDay()

Returns
Returns the day in the month of the date.

Type
int

getMonth()

Returns
Returns the month of the date.

Type
int

getOrderValue()

Used to order dates. Gives a comparable int s.t. *getOrderValue(a)* < *getOrderValue(b)* iff the date *a* was before the date *b*. :return: Returns an int representing the position in an order of the date. :type: int

getYear()

Returns
Returns the year of the date.

Type
int

1.7 Filters

class pyfolio_performance.**Filters**

Class that provides usefull filtering functions for the cluster analysis.

static **fAnd**(*f1*, *f2*)

Parameters

- **f1** – First function.
- **f2** – Second function.

Type
function entry -> bool

Type
function entry -> bool

Returns
Returns a function that first evaluates both functions and returns the *and*.

Type
Entry -> bool

`static fBefore(date)`

Parameters
year (DateObject) – The date to filter for.

Returns
A filter function that ensures the entry was made before or on the date (<=).

Type
Entry -> bool

`static fDay(day)`

Parameters
day (*int*) – The day to filter for.

Returns
A filter function that ensures the entry was made in the specified day.

Type
Entry -> bool

`static fDepotTransaction()`

Returns
A filter function that ensures the entry is a Depot Transaction.

Type
Entry -> bool

`static fEnsureTypeList(typelist)`

Parameters
typelist (*list(str)*) – List of types that are required by the filter.

Returns
A filter function that ensures the entry has a type contained in the typelist.

Type
Entry -> bool

`static fExcludeTypeList(typelist)`

Parameters
typelist (*list(str)*) – List of types that are not allowed by the filter.

Returns
A filter function that ensures the entry has *not* a type contained in the typelist.

Type
Entry -> bool

`static fMonth(month)`

Parameters
month (*int*) – The month to filter for.

Returns
A filter function that ensures the entry was made in the specified month.

Type
Entry -> bool

`static fOr(f1, f2)`

Parameters

- **f1** – First function.
- **f2** – Second function.

Type
function entry -> bool

Type
function entry -> bool

Returns
Returns a function that first evaluates both functions and returns the *or*.

Type
Entry -> bool

`static fSecurityTransaction(sec)`

Parameters
sec (Security) – A security to filter for.

Returns
A filter function that ensures the entry is a transaction about the given security.

Type
Entry -> bool

`static fYear(year)`

Parameters
year (*int*) – The year to filter for.

Returns
A filter function that ensures the entry was made in the specified year.

Type
Entry -> bool

CHAPTER

TWO

EXAMPLES

Some examples of how the pyfolio performance library can be used to analyse to analyse your portfolio.

2.1 First Example: Loading the portfolio

In this example we load the portfolio and display some content.

```
from pyfolio-performance import Portfolio

portfolio = Portfolio("portfolio.xml")

print(portfolio.getAccounts())
print(portfolio.getDepots())
```

The result will look like:

```
[Account/Comdirect Cash: 2500, Account/Norisbank: 30100, Account/P2P Bondora: 80000]
[Depot/Comdirect, Depot/Consorsbank]
```

The string method for the accounts and the depots returns the *type* followed by a / and then the *name* of the object as it appears in portfolio performance.

The account representation also includes the value of the account in cents.

2.2 Analysis: Dividend of the month

In the following example we compute the dividends received this month in 2 different way. The first one simply aggregates the dividend. The second examlpe aggregates it by security.

```
from pyfolio_performance import Portfolio, Filters
from datetime import datetime
portfolio = Portfolio("portfolio.xml")
currentNow = datetime.now()

def filter_month(entry, month, year):
    if year != entry.getYear() or month != entry.getMonth():
        return False
    return True
```

(continues on next page)

(continued from previous page)

```
filter_dividend = Filters.fAnd(Filters.fEnsureTypeList(['DIVIDENDS']),
    lambda entry: filter_month(entry, currentNow.month, currentNow.year) )

# different clustering
def cluster_dividend(allCluster, entry):
    return "val"

def aggregate_dividend(cluster, entry):
    return cluster + entry.getValue()

divicluster = {'val': 0}
portfolio.evaluateCluster(divicluster, filter_dividend, cluster_dividend, aggregate_
↪dividend)
print(divicluster)

# Dividends are clustered by their name
def cluster_dividend2(allCluster, entry):
    k = entry.getSourceName()
    if k not in allCluster:
        allCluster[k] = 0
    return k
divicluster = {}
portfolio.evaluateCluster(divicluster, filter_dividend, cluster_dividend2, aggregate_
↪dividend)
print(divicluster)
```

The code leads to the following two outputs. The first one simply gives the sum of all dividends. The second one aggregates it by the security that distributes the dividend.

The value is returned in cents. There were 13,79 received through the securities *AT + T*, *Realty Income* and *Proctor and Gamble*. The displayed name and corresponding key used in the code reflects the name given in portfolio performance's security section.

```
{'val': 1379}
{'AT + T': 190, 'Realty Income': 781, 'Proctor and Gamble': 408}
```

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

INDEX

S