Module `brownbear`

Sub-modules

- brownbear.analyze
- brownbear.fetch
- brownbear.metrics
- brownbear.optimizer
- brownbear.portfolio
- brownbear.symbol_cache
- brownbear.utility
- brownbear.weight

Functions

Function `DBG`

```
def DBG(
    *s
)
```

Module `brownbear.analyze`

Portfolio analysis and optimization.

Functions

Function `analyze`

```
def analyze(
    df,
    portfolio_option,
    weight_by=None,
    default_correlation=1
)
```

Analyze Portfolio.

Parameters

`df` : pd.DataFrame Dataframe of investment options with columns for asset class, description, and performace metrics.

`portfolio_option` : dict Dictionary of investment options along with their weights. The keys are the investment options and the values are the weights. The first entry in the dict must be the title of the portfolio. portfolio_option may be modified if weight_by is not None, i.e. the weights for each investment option might be adjusted.

`weight_by` : dict of dicts, optional Specify the weighting scheme. If not None, it will replace the weights specified in the portfolio. You can also fix the weights on some Investent Options, Asset Classes, and Asset Subclasses, while the others are automatically calculated (default is None, which implies use the user specified weights specified in portfolio_option).

'Equal' - use equal weights.

'Sharpe Ratio' - use proportionally weighted allocations based on the percent of an investment option's sharpe ratio to the sum of all the sharpe ratios in the portfolio.

'Std Dev' - use standard deviation adjusted weights.

'Annual Returns' - use return adjusted weights.

'Vola' - use volatility adjusted weights.

'DS Vola' - use downside volatility adjusted weights.

None: ‘Investment Option’ means use user specified weights ‘Asset Class’ means do not group by Asset Class ‘Asset Subclass’ means do not group by Asset Subclass

Example:

At the Asset Class level, explicitly specify US Stock, US Bonds, and Risk-Free Asset weights, then equally allocate among any remaining asset classes. Next, do not consider the Asset Subclass within an Asset Class. Finally, weight the Investment Options within each Asset Class by Annual Return.

weight_by = {
‘Asset Class’: {‘weight_by’: ‘Equal’,
‘US Stocks’: 0.40,
‘US Bonds’: 0.40,
‘Risk-Free Asset’: 0.10},
‘Asset Subclass’: {‘weight_by’: None},
‘Investment Option’: {‘weight_by’: ‘Annual Returns’},
}

default_correlation : int, optional Correlation to use when no correlation has been specified between two asset classes. If you use only the Asset Classes defined in universe/asset-classes.csv, then this will never happen. (default is 1, which assumes that the assets are perfectly coorelated, i.e. worst case for asset diversification).

Returns

`annual_return` : float The expected annualized return of the portfolio.
`std_dev` : float The standard deviation of the portfolio.
`sharpe_ratio` : float The overall sharpe ratio of the portfolio.

Function `print_portfolio`

```
def print_portfolio(
    portfolio_option
)
```

Print portfolio options with their weights.

Function `show_pie_charts`

```
def show_pie_charts(
    df,
    portfolio_option,
    charts=None
)
```

Show pie chart(s) of investment allocations by percent.

Parameters

`df` : pd.DataFrame Dataframe of investment options with columns for asset class, description, and performace metrics.
`portfolio_option` : dict Dictionary of investment options along with their weights. The keys are the investment options and the values are the weights. The first entry in the dict must be the title of the portfolio. portfolio_option may be modified if weight_by is not None, i.e. the weights for each investment option might be adjusted.
`charts` : list of str, optional {‘Investment Option’, ‘Asset Class’, ‘Asset Subclass’}. The charts to display (default is None, which implies all charts).

Returns

None

Function `summary`

```
def summary(
    df,
    portfolio_option,
    annual_ret,
    std_dev,
    sharpe_ratio
)
```

Generate summary results.

Note: analyze() must be called before calling summary().

Parameters

`df` : pd.DataFrame Dataframe of investment options with columns for asset class, description, and performace metrics.

`portfolio_option` : dict Dictionary of investment options along with their weights. The keys are the investment options and the values are the weights. The first entry in the dict must be the title of the portfolio. portfolio_option may be modified if weight_by is not None, i.e. the weights for each investment option might be adjusted.

`annual_return` : float The expected annualized return of the portfolio.

`std_dev` : float The standard deviation of the portfolio.

`sharpe_ratio` : float The overall sharpe ratio of the portfolio.

Returns

`summary` : pd.DataFrame Summary results.

Module `brownbear.fetch`

Fetch performance data for investment options.

Functions

Function `add_fundamental_columns`

```
def add_fundamental_columns(
    inv_opts,
    clean=True
)
```

Add fundamental data columns to inv_opts dataframe.

Columns added: 'Previous Close', 'Trailing PE', 'Dividend Yield', 'Market Cap'

Parameters

`inv_opts` : pd.DataFrame Dataframe of investment options with columns for asset class, description, and performace metrics.

`clean` : bool, optional True to remove rows that have a 'nan' as a column value (default is True).

Returns

`inv_opts` : pd.DataFrame Dataframe of investment options with fundamental data columns.

Function `fetch`

```
def fetch(
    investment_universe,
    risk_free_rate=0,
    annual_returns='Annual Returns',
    vola='Std Dev',
    ds_vola='Std Dev',
```

```
    clean=True
)
```

Fetch Investment Universe and asset classes

investment-options.csv format: “Investment Option”, “Description”(optimal), “Asset Class”, “Annual Returns”, “Std Dev”

asset-classes.csv format: “Asset Class A”, “Asset Class B”,“Correlation” “Description” field is optional. It is not referenced in code. “Annual Returns” column(s) can named anything. Recommend “1 Yr”, “3 Yr”, “5 Yr”, or “10 Yr”. Then annual_returns parameter can select the column to use.

Parameters

`investment_universe` : list of str List of investment galaxies. These are the dirs within universe/, for example [‘dow30-galaxy’, ‘alabama-galaxy’].

`risk_free_rate` : float, optional Risk free rate (default is 0).

`annual_returns` : str, optional Specifies which column to use for annualized returns (default is ‘Annual Returns’).

`vola` : str, optional Specifies which column to use for volatility (default is ‘Std Dev’).

`ds_vola` : str, optional Specifies which column to use for downside volatility (default is ‘Std Dev’).

`clean` : bool, optional True to remove rows that have a ‘nan’ as a column value (default is True).

Returns

`inv_opts` : pd.DataFrame Dataframe of investment options with columns for asset class, description, and performace metrics.

Function `rank`

```
def rank(
    inv_opts,
    rank_by,
    group_by=None,
    num_per_group=None,
    ascending=False
)
```

Rank investment options.

Parameters

`inv_opts` : pd.DataFrame Dataframe of investment options with columns for asset class, description, and performace metrics.

`rank_by` : str The performance or fundamental metric used to sort the investment options.

`group_by` : str, optional `{None, 'Asset Class', 'Asset Subclass'}` How to group investment options (default is None, which imples no grouping)

`num_per_group` : int, optional The number of investment options for each group (default is None, which imples 5 if group_by is specified, otherwise 1000).

`ascending` : bool, optional True to sort in ascending order (default is False, which imples sorting in descending order).

Returns

`df` : pd.DataFrame Dataframe of investment options with ranking.

Module `brownbear.metrics`

Compute metrics.

Functions

Function `annualized_returns`

```
def annualized_returns(
```

```
    df,
    timeperiod='daily',
    years=5
)
```

Calculate the annualized returns of entire dataframe.

Parameters

df : pd.DataFrame Timeseries with a column for each symbol. Each row contains the daily closing prices for the symbols.

timeperiod : str, optional {'daily', 'weekly', 'monthly', 'quarterly'} Specifies the sample rate of the timeseries 'df' (default is 'daily').

years : float, optional Number of years over which to calculate annualized returns (default is 5).

Returns

s : pd.Series Series of key[value] pairs in which key is the symbol and value is the annualized return.

Function annualized_standard_deviation

```
def annualized_standard_deviation(
    returns,
    timeperiod='monthly',
    years=3,
    downside=False
)
```

Calculate the annualized standard deviation of entire dataframe.

Parameters

df : pd.DataFrame Timeseries with a column for each symbol. Each row contains the daily closing prices for the symbols.

timeperiod : str, optional {'daily', 'weekly', 'monthly', 'quarterly'} Specifies the sample rate of the timeseries 'df' (default is 'daily').

years : float, optional Number of years over which to calculate standard deviation (default is 3).

downside : bool, optional True to calculate the downside standard deviation, otherwise False (default is False).

Returns

s : pd.Series Series of key[value] pairs in which key is the symbol and value is the annualized standard deviation.

Function cagr

```
def cagr(
    B,
    A,
    n
)
```

Calculate compound annual growth rate.

Parameters

B : float Ending balance.

A : float Beginning balance.

n : float Number of years over which to calculate cagr.

Returns

float Compound annual growth rate.

Function `correlation_map`

```
def correlation_map(
    df,
    method='log',
    days=None
)
```

Compute and Plot correlation map between symbols.

See example use in asset-class-galaxy/asset-classes.ipynb

Parameters

`df` : pd.DataFrame Timeseries with a column for each symbol. Each row contains the daily closing prices for the symbols.

`method` : str, optional `{log, price, returns}` Timeseries can be altered so that correlation is based on a price derivative (log or returns) instead of price. 'price' does not alter the timeseries. (default is 'log').

`days` : int, optional The last number of days over which to compute the correlations. (default is None, which implies all days).

Returns

`df` : pd.DataFrame Dataframe representing the correlation map between symbols.

Module `brownbear.optimizer`

Portfolio analysis and optimization.

Functions

Function `optimizer`

```
def optimizer(
    df,
    portfolio_option,
    constraints=None
)
```

Optimize the Portfolio based on Sharpe Ratio.

Optimize sharpe ratio while specifying Annual Rate, Worst Typical Down Year, and Black Swan. Setting a constraint to None optimizes absolute Sharpe Ratio without regard to that constraint.

Parameters

`df` : pd.DataFrame Dataframe of investment options with columns for asset class, description, and performace metrics.

`portfolio_option` : dict Dictionary of investment options along with their weights. The keys are the investment options and the values are the weights. The first entry in the dict must be the title of the portfolio. portfolio_option may be modified if weight_by is not None, i.e. the weights for each investment option might be adjusted.

`constraints` : dict Used to specify constraints for the optimization. Valid constraints are: 'Annual Return', 'Worst Typical Down Year', and 'Black Swan' (default is None, which implies maximize Sharpe Ratio without considering any constraints).

Returns

None

Module `brownbear.portfolio`

Portfolio analysis and optimization.

Variables

Variable `PORT`

class : Single instance of Portfolio object to be used glabally.

Functions

Function `correlation`

```
def correlation(
    correlation_table,
    a,
    b
)
```

Return the correlation between asset a and b using the correlation table dict. Assets are in the form a=asset_class:asset_subclass.

Function `expected_return`

```
def expected_return(
    annual_returns,
    weights
)
```

Returns expected return given list of investment option returns and their corresponding weights.

Function `get_metric_lists`

```
def get_metric_lists(
    df,
    portfolio_option
)
```

Creates lists for investment option, std_dev, asset_class, etc… for each investment option for the specified portfolio.

Function `sharpe_ratio`

```
def sharpe_ratio(
    annual_ret,
    std_dev,
    risk_free_rate
)
```

Return the sharpe ratio.

This is the modified sharpe ratio formulated by Craig L. Israelsen. It's the same as the sharpe ration when the excess return is positive.

Function `standard_deviation`

```
def standard_deviation(
    weights,
    std_devs,
    asset_classes
)
```

Return std_dev given lists of weights, std_devs, and asset_classes Reference: https://en.wikipedia.org/wiki/Modern_portfolio_theory

Classes

Class `Portfolio`

```
class Portfolio
```

Initialize instance variables.

Attributes

`investment_universe` : str List of investment galaxies.
`risk_free_rate` : float Risk free rate.
`correlation_table` : dict Correlation table of asset classes within universe.
`asset_class_table` : pd.DataFrame Asset class returns.
`default_correlation` : int Default correlation when none is specified.
`vola_column` : str Column name to use for vola.
`ds_vola_column` : str Column name to use for ds_vola.
`portfolio_title` : str Portfolio title.

Module `brownbear.symbol_cache`

Symbol cache management.

Functions

Function `compile_timeseries`

```
def compile_timeseries(
    symbols
)
```

Compile one or more symbols' timeseries into a single dataframe.

The timeseries are read from the symbol cache only, so the timeseries must exists for every symbol in symbols. Otherwise an exception will be thrown.

The compiled timeseries has a column for each symbol. Each row contains the daily closing prices for the symbols. This timeseries is written to 'symbols-timeseries.csv' in the current directory.

Parameters

`symbols` : list of str The list of symbols for securities.

Returns

None

Function `fetch_timeseries`

```
def fetch_timeseries(
    symbols,
    start=None,
    end=None,
    refresh=False
)
```

Fetch timeseries for one or more symbols from yahoo finance.

Write the timeseries to the symbol cache as symbol.csv

Parameters

`symbols` : list of str The list of symbols for securities.
`start` : (string, int, date, datetime, Timestamp), optional Starting date. Parses many different kind of date representations (e.g., 'JAN-01-2010', '1/1/10', 'Jan, 1, 1980') (Default is None, which implies 01-01-2015).

end : (string, int, date, datetime, Timestamp), optional Ending date, timestamp. Same format as starting date (Default is NONE, which implies yesterday).

refresh : bool, optional True to retrieve timeseries from internet instead of using symbol cache (default is False).

Returns

None

Function `get_symbol_fundamentals`

```
def get_symbol_fundamentals(
    symbols=None
)
```

Get fundamental data for list of symbols.

Filter out any symbols prefixed with '___'.

Parameters

symbols : list of str, optional The list of symbols for securities (default is None, which imples all symbols in symbol cache).

Returns

None

Function `get_symbol_metadata`

```
def get_symbol_metadata(
    symbols=None
)
```

Get symbol metadata for list of symbols

Filter out any symbols prefixed with '___'.

Parameters

symbols : list of str, optional The list of symbols for securities (default is None, which imples all symbols in symbol cache).

Returns

None

Function `remove_cache_symbols`

```
def remove_cache_symbols(
    symbols=None
)
```

Remove cached timeseries for list of symbols.

Filter out any symbols prefixed with '___'.

Parameters

symbols : list of str, optional The list of symbols for securities (default is None, which imples all symbols in symbol cache).

Returns

None

Function `update_cache_symbols`

```
def update_cache_symbols(
    symbols=None
)
```

Update cached timeseries for list of symbols.

Filter out any symbols prefixed with '___'.

Parameters

`symbols` : list of str, optional The list of symbols for securities (default is None, which imples all symbols in symbol cache).

Returns

None

Module `brownbear.utility`

Utility functions.

Variables

Variable `ROOT`

str : Full path to brownbear project root dir.

Variable `SYMBOL_CACHE`

str : Full path to symbol-cache dir.

Variable `TRADING_DAYS_PER_MONTH`

int : The number of trading days per month.

Variable `TRADING_DAYS_PER_WEEK`

int : The number of trading days per week.

Variable `TRADING_DAYS_PER_YEAR`

int : The number of trading days per year.

Functions

Function `csv_to_df`

```
def csv_to_df(
    filepaths
)
```

Read multiple csv files into a dataframe.

Parameters

`filepaths` : list of str List of of full path to csv files.

Returns

`df` : pd.DataFrame Dataframe representing the concatination of the csv files listed in filepaths.

Function `get_quote`

```
def get_quote(
    symbols
)
```

Returns the current quote for a list of symbols as a dict.

Function `print_full`

```
def print_full(
    x
)
```

Print every row of list-like object.

Classes

Class `dotdict`

```
class dotdict(
    *args,
    **kwargs
)
```

Provides dot.notation access to dictionary attributes.

Examples

```
>>> mydict = {'val' : 'it works'}
>>> mydict = dotdict(mydict)
>>> mydict.val
'it works'
>>> nested_dict = {'val' : 'nested works too'}
>>> mydict.nested = dotdict(nested_dict)
>>> mydict.nested.val
'nested works too'
```

Ancestors (in MRO)

- builtins.dict

Module `brownbear.weight`

Portfolio analysis and optimization.

Functions

Function `assign_weights`

```
def assign_weights(
    df,
    ml,
    portfolio_option,
    weight_by
)
```

Specify the weighting scheme. It will replace the weights specified in the portfolio. You can also fix the weights on some Investent Options, Asset Classes, and Asset Subclasses while the others are automatically calculated.

'Equal' - will use equal weights.

'Sharpe Ratio' - will use proportionally weighted allocations based on the percent of an investment option's sharpe ratio to the sum of all the sharpe ratios in the portfolio.

'Std Dev' - will use standard deviation adjusted weights.

'Annual Returns' - will use return adjusted weights.

'Vola' - will use volatility adjusted weights.

'DS Vola' - will use downside volatility adjusted weights.

None: 'Investment Option' means use use specified weights. 'Asset Class' means do not group by Asset Class. 'Asset Subclass means do not group by Asset Subclass.

Function `check_allocation`

```
def check_allocation(
    weights,
    asset_class_name
)
```

Make sure total adds to 100.
