API — ffn 0.3.0 documentation

core Module¶

class `ffn.core.GroupStats`(**prices*)[source]¶

Bases: `dict`

GroupStats enables one to compare multiple series side by side. It is a wrapper around a dict of {price.name: PerformanceStats} and provides many convenience methods.

The order of the series passed in will be preserved. Individual PerformanceStats objects can be accessed via index position or name via the [] accessor.

Args:

- prices (Series): Multiple price series to be compared.

Attributes:

- stats (DataFrame): Dataframe containing stats for each
  series provided. Stats in rows, series in columns.
- lookback_returns (DataFrame): Returns for diffrent
  lookback periods (1m, 3m, 6m, ytd...) Period in rows, series in columns.
- prices (DataFrame): The merged and rebased prices.

`display`()[source]¶

Display summary stats table.

`display_lookback_returns`()[source]¶

Displays the current lookback returns for each series.

`plot`(*freq=None*, *figsize=(15, 5)*, *title=None*, *logy=False*, ***kwargs*)[source]¶

Helper function for plotting the series.

Args:

- freq (str): Data frequency used for display purposes.
  Refer to pandas docs for valid freq strings.
- figsize ((x,y)): figure size
- title (str): Title if default not appropriate
- logy (bool): log-scale for y axis
- kwargs: passed to pandas’ plot method

`plot_correlation`(*freq=None*, *title=None*, *figsize=(12, 6)*, ***kwargs*)[source]¶

Utility function to plot correlations.

Args:

- freq (str): Pandas data frequency alias string
- title (str): Plot title
- figsize (tuple (x,y)): figure size
- kwargs: passed to Pandas’ plot_corr_heatmap function

`plot_histograms`(*freq=None*, *title=None*, *figsize=(10, 10)*, ***kwargs*)[source]¶

Wrapper around pandas’ hist.

Args:

- freq (str): Data frequency used for display purposes.
  Refer to pandas docs for valid freq strings.
- figsize ((x,y)): figure size
- title (str): Title if default not appropriate
- kwargs: passed to pandas' hist method

`plot_scatter_matrix`(*freq=None*, *title=None*, *figsize=(10, 10)*, ***kwargs*)[source]¶

Wrapper around pandas' scatter_matrix.

Args:

- freq (str): Data frequency used for display purposes.
  Refer to pandas docs for valid freq strings.
- figsize ((x,y)): figure size
- title (str): Title if default not appropriate
- kwargs: passed to pandas' scatter_matrix method

`set_date_range`(*start=None*, *end=None*)[source]¶

Update date range of stats, charts, etc. If None then the original date range is used. So to reset to the original range, just call with no args.

Args:

- start (date): start date
- end (end): end date

`set_riskfree_rate`(*rf*)[source]¶

Set annual risk-free rate property and calculate properly annualized monthly and daily rates. Then performance stats are recalculated. Affects only those instances of PerformanceStats that are children of this GroupStats object.

Args:

- rf (float): Annual risk-free rate

`to_csv`(*sep=', '*, *path=None*)[source]¶

Returns a CSV string with appropriate formatting. If path is not None, the string will be saved to file at path.

Args:

- sep (char): Separator
- path (str): If None, CSV string returned. Else file
  written to specified path.

class `ffn.core.PerformanceStats`(*prices*, *rf=0.0*)[source]¶

Bases: `object`

PerformanceStats is a convenience class used for the performance evaluation of a price series. It contains various helper functions to help with plotting and contains a large amount of descriptive statistics.

Args:

- prices (Series): A price series.
- rf (float): Risk-free rate used in various calculation. Should be
  expressed as a yearly (annualized) return

Attributes:

- name (str): Name, derived from price series name
- return_table (DataFrame): A table of monthly returns with
  YTD figures as well.
- lookback_returns (Series): Returns for different
  lookback periods (1m, 3m, 6m, ytd...)
- stats (Series): A series that contains all the stats

`display`()[source]¶

Displays an overview containing descriptive stats for the Series provided.

`display_lookback_returns`()[source]¶

Displays the current lookback returns.

`display_monthly_returns`()[source]¶

Display a table containing monthly returns and ytd returns for every year in range.

`plot`(*freq=None*, *figsize=(15, 5)*, *title=None*, *logy=False*, ***kwargs*)[source]¶

Helper function for plotting the series.

Args:

- freq (str): Data frequency used for display purposes.
  Refer to pandas docs for valid freq strings.
- figsize ((x,y)): figure size
- title (str): Title if default not appropriate
- logy (bool): log-scale for y axis
- kwargs: passed to pandas' plot method

`plot_histogram`(*freq=None*, *figsize=(15, 5)*, *title=None*, *bins=20*, ***kwargs*)[source]¶

Plots a histogram of returns given a return frequency.

Args:

- freq (str): Data frequency used for display purposes.
  This will dictate the type of returns (daily returns, monthly, ...) Refer to pandas docs for valid period strings.
- figsize ((x,y)): figure size
- title (str): Title if default not appropriate
- bins (int): number of bins for the histogram
- kwargs: passed to pandas' hist method

`set_date_range`(*start=None*, *end=None*)[source]¶

Update date range of stats, charts, etc. If None then the original date is used. So to reset to the original range, just call with no args.

Args:

- start (date): start date
- end (end): end date

`set_riskfree_rate`(*rf*)[source]¶

Set annual risk-free rate property and calculate properly annualized monthly and daily rates. Then performance stats are recalculated. Affects only this instance of the PerformanceStats.

Args:

- rf (float): Annual risk-free rate

`to_csv`(*sep=', '*, *path=None*)[source]¶

Returns a CSV string with appropriate formatting. If path is not None, the string will be saved to file at path.

Args:

- sep (char): Separator
- path (str): If None, CSV string returned. Else file written

  to specified path.

`ffn.core.annualize`(*returns*, *durations*, *one_year=365.0*)[source]¶

Annualize returns using their respective durations.

Formula used is:

(1 + returns) ** (1 / (durations / one_year)) - 1

`ffn.core.asfreq_actual`(*series*, *freq*, *method='ffill'*, *how='end'*, *normalize=False*)[source]¶

Similar to pandas' asfreq but keeps the actual dates. For example, if last data point in Jan is on the 29th, that date will be used instead of the 31st.

`ffn.core.calc_cagr`(*prices*)[source]¶

Calculates the CAGR (compound annual growth rate) for a given price series.

Args:

- prices (pandas.Series): A Series of prices.

Returns:

- float – cagr.

`ffn.core.calc_calmar_ratio`(*prices*)[source]¶

Calculates the Calmar Ratio given a series of prices

Args:

- prices (Series, DataFrame): Price series

`ffn.core.calc_clusters`(*returns*, *n=None*, *plot=False*)[source]¶

Calculates the clusters based on k-means clustering.

Args:

- returns (pd.DataFrame): DataFrame of returns
- n (int): Specify # of clusters. If None, this

  will be automatically determined
- plot (bool): Show plot?

Returns:

- dict with structure: {cluster# : [col names]}

`ffn.core.calc_erc_weights`(*returns*, *initial_weights=None*, *risk_weights=None*, *covar_method='ledoit-wolf'*, *risk_parity_method='ccd'*, *maximum_iterations=100*, *tolerance=1e-08*)[source]¶

Calculates the equal risk contribution / risk parity weights given a DataFrame of returns.

Args:

- returns (DataFrame): Returns for multiple securities.
- initial_weights (list): Starting asset weights [default inverse vol].

- risk_weights (list): Risk target weights [default equal weight].
- covar_method (str): Covariance matrix estimation method.
  Currently supported:
  - ledoit-wolf [default]
  - standard
- risk_parity_method (str): Risk parity estimation method.
  Currently supported:
  - ccd (cyclical coordinate descent)[default]
- maximum_iterations (int): Maximum iterations in iterative solutions.
- tolerance (float): Tolerance level in iterative solutions.

Returns:
Series {col_name: weight}

`ffn.core.calc_ftca`(*returns*, *threshold=0.5*)[source]¶

Implementation of David Varadi's Fast Threshold Clustering Algorithm (FTCA).

http://cssanalytics.wordpress.com/2013/11/26/fast-threshold-clustering-algorithm-ftca/ # NOQA

More stable than k-means for clustering purposes. If you want more clusters, use a higher threshold.

Args:
- returns - expects a pandas dataframe of returns where
  each column is the name of a given security.
- threshold (float): Threshold parameter - use higher value
  for more clusters. Basically controls how similar (correlated) series have to be.

Returns:
dict of cluster name (a number) and list of securities in cluster

`ffn.core.calc_information_ratio`(*returns*, *benchmark_returns*)[source]¶

http://en.wikipedia.org/wiki/Information_ratio

`ffn.core.calc_inv_vol_weights`(*returns*)[source]¶

Calculates weights proportional to inverse volatility of each column.

Returns weights that are inversely proportional to the column's volatility resulting in a set of portfolio weights where each position has the same level of volatility.

Note, that assets with returns all equal to NaN or 0 are excluded from the portfolio (their weight is set to NaN).

Returns:
Series {col_name: weight}

`ffn.core.calc_max_drawdown`(*prices*)[source]¶

Calculates the max drawdown of a price series. If you want the actual drawdown series, please use to_drawdown_series.

`ffn.core.calc_mean_var_weights`(*returns*, *weight_bounds=(0.0, 1.0)*, *rf=0.0*, *covar_method='ledoit-wolf'*)[source]¶

Calculates the mean-variance weights given a DataFrame of returns.

Args:
- returns (DataFrame): Returns for multiple securities.
- weight_bounds ((low, high)): Weigh limits for optimization.

- rf (float): Risk-free rate used in utility calculation
- covar_method (str): Covariance matrix estimation method.

  Currently supported:
  - ledoit-wolf
  - standard

Returns:

Series {col_name: weight}

`ffn.core.calc_perf_stats`(*prices*)[source]¶

Calculates the performance statistics given an object. The object should be a Series of prices.

A PerformanceStats object will be returned containing all the stats.

Args:

- prices (Series): Series of prices

`ffn.core.calc_prob_mom`(*returns*, *other_returns*)[source]¶

Probabilistic momentum

Basically the "probability or confidence that one asset is going to outperform the other".

Source:

http://cssanalytics.wordpress.com/2014/01/28/are-simple-momentum-strategies-too-dumb-introducing-probabilistic-momentum/ # NOQA

`ffn.core.calc_risk_return_ratio`(*returns*)[source]¶

Calculates the return / risk ratio. Basically the Sharpe ratio without factoring in the risk-free rate.

`ffn.core.calc_sharpe`(*returns*, *rf=0.0*, *nperiods=None*, *annualize=True*)[source]¶

Calculates the Sharpe ratio.

If rf is non-zero, you must specify nperiods. In this case, rf is assumed to be expressed in yearly (annualized) terms.

Args:

- returns (Series, DataFrame): Input return series
- rf (float): Risk-free rate expressed as a yearly (annualized) return
- nperiods (int): Frequency of returns (252 for daily, 12 for monthly,

  etc.)

`ffn.core.calc_sortino_ratio`(*returns*, *rf=0*, *nperiods=None*, *annualize=True*)[source]¶

Calculates the sortino ratio given a series of returns

Args:

- returns (Series or DataFrame): Returns
- rf (float): Risk-free rate expressed in yearly (annualized) terms.
- nperiods (int): Number of periods used for annualization. Must be

  provided if rf is non-zero

`ffn.core.calc_stats`(*prices*)[source]¶

Calculates performance stats of a given object.

If object is Series, a PerformanceStats object is returned. If object is DataFrame, a GroupStats object is returned.

Args:

- prices (Series, DataFrame): Set of prices

`ffn.core.calc_total_return`(*prices*)[source]¶

Calculates the total return of a series.

last / first - 1

`ffn.core.deannualize`(*returns*, *nperiods*)[source]¶

Convert return expressed in annual terms on a different basis.

Args:

- returns (float, Series, DataFrame): Return(s)
- nperiods (int): Target basis, typically 252 for daily, 12 for
  monthly, etc.

`ffn.core.drawdown_details`(*drawdown*)[source]¶

Returns a data frame with start, end, days (duration) and drawdown for each drawdown in a drawdown series.

Note

days are actual calendar days, not trading days

Args:

- drawdown (pandas.Series): A drawdown Series
  (can be obtained w/ drawdown(prices).

Returns:

- pandas.DataFrame – A data frame with the following
  columns: start, end, days, drawdown.

`ffn.core.drop_duplicate_cols`(*df*)[source]¶

Removes duplicate columns from a dataframe and keeps column w/ longest history

`ffn.core.extend_pandas`()[source]¶

Extends pandas' PandasObject (Series, Series, DataFrame) with some functions defined in this file.

This facilitates common functional composition used in quant finance.

Ex:

prices.to_returns().dropna().calc_clusters() (where prices would be a DataFrame)

`ffn.core.get_num_days_required`(*offset*, *period='d'*, *perc_required=0.9*)[source]¶

Estimates the number of days required to assume that data is OK.

Helper function used to determine if there are enough "good" data days over a given period.

Args:

- offset (DateOffset): Offset (lookback) period.
- period (str): Period string.
- perc_required (float): percentage of number of days
  expected required.

`ffn.core.limit_weights`(*weights*, *limit=0.1*)[source]¶

Limits weights and redistributes excedent amount proportionally.

ex:

- weights are {a: 0.7, b: 0.2, c: 0.1}
- call with limit=0.5
- excess 0.2 in a is ditributed to b and c
  proportionally. - result is {a: 0.5, b: 0.33, c: 0.167}

Args:
- weights (Series): A series describing the weights
- limit (float): Maximum weight allowed

`ffn.core.merge`(**series*)[source]¶

Merge Series and/or DataFrames together.

Returns a DataFrame.

`ffn.core.plot_corr_heatmap`(*data*, ***kwargs*)[source]¶

Plots the correlation heatmap for a given DataFrame.

`ffn.core.plot_heatmap`(*data*, *title='Heatmap'*, *show_legend=True*, *show_labels=True*, *label_fmt='.2f'*, *vmin=None*, *vmax=None*, *figsize=None*, *label_color='w'*, *cmap='RdBu'*, ***kwargs*)[source]¶

Plot a heatmap using matplotlib's pcolor.

Args:
- data (DataFrame): DataFrame to plot. Usually small matrix (ex.
  correlation matrix).
- title (string): Plot title
- show_legend (bool): Show color legend
- show_labels (bool): Show value labels
- label_fmt (str): Label format string
- vmin (float): Min value for scale
- vmax (float): Max value for scale
- cmap (string): Color map
- kwargs: Passed to matplotlib's pcolor

`ffn.core.random_weights`(*n*, *bounds=(0.0, 1.0)*, *total=1.0*)[source]¶

Generate pseudo-random weights.

Returns a list of random weights that is of length n, where each weight is in the range bounds, and where the weights sum up to total.

Useful for creating random portfolios when benchmarking.

Args:
- n (int): number of random weights
- bounds ((low, high)): bounds for each weight
- total (float): total sum of the weights

`ffn.core.rebase`(*prices*, *value=100*)[source]¶

Rebase all series to a given intial value.

This makes comparing/plotting different series together easier.

Args:

- prices: Expects a price series
- value (number): starting value for all series.

`ffn.core.rescale`(*x*, *min=0.0*, *max=1.0*, *axis=0*)[source]¶

Rescale values to fit a certain range [min, max]

`ffn.core.rollapply`(*data*, *window*, *fn*)[source]¶

Apply a function fn over a rolling window of size window.

Args:

- data (Series or DataFrame): Series or DataFrame
- window (int): Window size
- fn (function): Function to apply over the rolling window.
  For a series, the return value is expected to be a single number. For a DataFrame, it shuold return a new row.

Returns:

- Object of same dimensions as data

`ffn.core.to_drawdown_series`(*prices*)[source]¶

Calculates the drawdown series.

This returns a series representing a drawdown. When the price is at all time highs, the drawdown is 0. However, when prices are below high water marks, the drawdown series = current / hwm - 1

The max drawdown can be obtained by simply calling .min() on the result (since the drawdown series is negative)

Method ignores all gaps of NaN's in the price series.

Args:

- prices (Series or DataFrame): Series of prices.

`ffn.core.to_excess_returns`(*returns*, *rf*, *nperiods=None*)[source]¶

Given a series of returns, it will return the excess returns over rf.

Args:

- returns (Series, DataFrame): Returns
- rf (float, Series, DataFrame): Risk-Free rate(s)
- nperiods (int): Optional. If provided, will convert rf to different
  frequency using deannualize

Returns:

- excess_returns (Series, DataFrame): Returns - rf

`ffn.core.to_log_returns`(*prices*)[source]¶

Calculates the log returns of a price series.

Formula is: ln(p1/p0)

Args:

- prices: Expects a price series

`ffn.core.to_monthly`(*series*, *method='ffill'*, *how='end'*)[source]¶

Convenience method that wraps asfreq_actual with 'M' param (method='ffill', how='end').

`ffn.core.to_price_index`(*returns*, *start=100*)[source]¶

Returns a price index given a series of returns.

Args:

- returns: Expects a return series
- start (number): Starting level

Assumes arithmetic returns.

Formula is: cumprod (1+r)

`ffn.core.to_returns`(*prices*)[source]¶

Calculates the simple arithmetic returns of a price series.

Formula is: (t1 / t0) - 1

Args:

- prices: Expects a price series

`ffn.core.to_ulcer_index`(*prices*)[source]¶

Converts from prices -> Ulcer index

See https://en.wikipedia.org/wiki/Ulcer_index

Args:

- prices (Series, DataFrame): Prices

`ffn.core.to_ulcer_performance_index`(*prices*, *rf=0.0*, *nperiods=None*)[source]¶

Converts from prices -> ulcer performance index.

See https://en.wikipedia.org/wiki/Ulcer_index

Args:

- prices (Series, DataFrame): Prices
- rf (float): Risk-free rate of return. Assumed to be expressed in
  yearly (annualized) terms
- nperiods (int): Used to deannualize rf if rf is provided (non-zero)

`ffn.core.winsorize`(*x*, *axis=0*, *limits=0.01*)[source]¶

Winsorize values based on limits

`ffn.core.year_frac`(*start*, *end*)[source]¶

Similar to excel's yearfrac function. Returns a year fraction between two dates (i.e. 1.53 years).

Approximation using the average number of seconds in a year.

Args:

- start (datetime): start date
- end (datetime): end date

`data` Module¶

`ffn.data.DEFAULT_PROVIDER`(*ticker*, *field*, *start=None*, *end=None*, *mrefresh=False*)¶

`ffn.data.csv`(*ticker*, *path='data.csv'*, *field=''*, *mrefresh=False*, ***kwargs*)[source]¶

Data provider wrapper around pandas' read_csv. Provides memoization.

`ffn.data.get`(*tickers*, *provider=None*, *common_dates=True*, *forward_fill=False*, *clean_tickers=True*, *column_names=None*, *ticker_field_sep=':'*, *mrefresh=False*, *existing=None*, ***kwargs*)[source]¶

Helper function for retrieving data as a DataFrame.

Args:

- tickers (list, string, csv string): Tickers to download.
- provider (function): Provider to use for downloading data.
  By default it will be ffn.DEFAULT_PROVIDER if not provided.
- common_dates (bool): Keep common dates only? Drop na's.
- forward_fill (bool): forward fill values if missing. Only works
  if common_dates is False, since common_dates will remove all nan's, so no filling forward necessary.
- clean_tickers (bool): Should the tickers be 'cleaned' using
  ffn.utils.clean_tickers? Basically remove non-standard characters (^VIX -> vix) and standardize to lower case.
- column_names (list): List of column names if clean_tickers
  is not satisfactory.
- ticker_field_sep (char): separator used to determine the
  ticker and field. This is in case we want to specify particular, non-default fields. For example, we might want: AAPL:Low,AAPL:High,AAPL:Close. ':' is the separator.
- mrefresh (bool): Ignore memoization.
- existing (DataFrame): Existing DataFrame to append returns
  to - used when we download from multiple sources
- kwargs: passed to provider

`ffn.data.web`(*ticker*, *field=None*, *start=None*, *end=None*, *mrefresh=False*, *source='yahoo'*)[source]¶

Data provider wrapper around pandas.io.data provider. Provides memoization.

`ffn.data.yf`(*ticker*, *field*, *start=None*, *end=None*, *mrefresh=False*)[source]¶

`utils` Module¶

`ffn.utils.as_format`(*item*, *format_str='.2f'*)[source]¶

Map a format string over a pandas object.

`ffn.utils.as_percent`(*self*, *digits=2*)[source]¶

`ffn.utils.clean_ticker`(*ticker*)[source]¶

Cleans a ticker for easier use throughout MoneyTree

Splits by space and only keeps first bit. Also removes any characters that are not letters. Returns as lowercase.

```
>>> clean_ticker('^VIX')
'vix'
>>> clean_ticker('SPX Index')
'spx'
```

`ffn.utils.clean_tickers`(*tickers*)[source]¶

Maps clean_ticker over tickers.

`ffn.utils.fmtn`(*number*)[source]¶

Formatting helper - float

`ffn.utils.fmtp`(*number*)[source]¶

Formatting helper - percent

`ffn.utils.fmtpn`(*number*)[source]¶

Formatting helper - percent no % sign

`ffn.utils.get_freq_name`(*period*)[source]¶

`ffn.utils.memoize`(*f*, *refresh_keyword='mrefresh'*)[source]¶

Memoize decorator. The refresh keyword is the keyword used to bypass the cache (in the function call).

`ffn.utils.parse_arg`(*arg*)[source]¶

Parses arguments for convenience. Argument can be a csv list (‘a,b,c’), a string, a list, a tuple.

Returns a list.

`ffn.utils.scale`(*val*, *src*, *dst*)[source]¶

Scale value from src range to dst range. If value outside bounds, it is clipped and set to the low or high bound of dst.

Ex:

scale(0, (0.0, 99.0), (-1.0, 1.0)) == -1.0 scale(-5, (0.0, 99.0), (-1.0, 1.0)) == -1.0