futures_research_class

index
/home/jirong/Desktop/github/trend_following/futures_research_class.py

```
Created on Tue Dec 15 23:57:07 2020

@author: jirong
```

Modules

datetime	numpy	matplotlib.pyplot	time
empyrical	os	quandl	util_futures
json	pandas	random	util
multiprocessing	pyfolio	statsmodels.api	yfinance

Classes

builtins.object

FuturesResearch

class **FuturesResearch**(builtins.object)

```
FuturesResearch(data_path, ewmac_variations, breakout_variations, optimize_weights_path, forecast_diff_before_rebal, notion_c
```

Methods defined here:

TSMOM_all_instr_returns(self)
```
Obtain TSMOM returns for all instrument based on 40% realized volatility for each single instrument
```

TSMOM_single_instr_monthly_returns(self, ret, lookback=12, cost=0.012)
```
Obtain TSMOM returns for each single instrument based on 40% realized volatility
```

__init__(self, data_path, ewmac_variations, breakout_variations, optimize_weights_path, forecast_diff_before_rebal, notion_capital_per_position, fix_
```
Constructor for FuturesResearch class

:param data_path: path to data file (e.g. "./trend_following/quantopian_data/futures_incl_2016.csv")
:param ewmac_varations: list of ewmac variations (e.g. [8,16,32,64])
:param breakout_variations: list of breakout variations (e.g. [40,80,160,320])
:param optimize_weights_path: path to storing weights in a folder ('./research/optimize_weights')
:param forecast_diff_before_rebal: Forecast difference before rebalancing an instrument position in a forecast range of
:param notion_capital_per_position (e.g 20000) (parameter used in study)
:param fix_capital: (e.g 500000) (parameter not used in study)
:param commission = 20,
:param bootsrap_sample_size: Minimum sample size in each boostrap (e.g. 300)
:param num_samples_per_period: Number of sample extracted from a period (e.g. 25)
:param prop_block_boostrap: Proportion of data extracted in each bootstrap sample (e.g. 0.25)
:param max_annual_volatility: Maximum portfolio realized volatility allowed (e.g. 0.15)
:param ind_instr_ref_volatility: Referenced volatility level for each instrument (e.g. 0.4)
:return: returns FutureResearch class
```

avg_optimized_sharpe_allinstr_single_period(self, period)
```
Parallelize optimization of sharpe across instruments in a period
:param period: Indexes referenced to a dictionary with reference to period which bootstrap indexes are extracted
```

compute_neg_sharpe(self, allocs_wts_forecasts, adj_forecast_single_instrument, price_series, ind_vol_target=0.4)
```
Compute sharpe in each bootstrap optimization
:param allocs_wts_forecasts: np.array weights applied to returns from individual forecasts.
:param adj_forecast_single_instrument: Normalized forecast time series for each instrument
:param price_series: Price series of instrument
:param ind_vol_target: Reference individual volatility target level (e.g. 0.4)
```

compute_optimal_leverage_all_instruments(self)
```
Obtain optimal leverage scaled to portfolio target and individual forecasts
```

create_dictionary_window_n_bootstrap_index(self, read_pickle=False)
```
Method for creating dictionar of window and bootstrap indexes.
```

create_window_index(self, df, window='expanding', days_block=252)
```
Method for creating window index

:param df: Data-frame
:param window: expanding or sliding
:param days_block: testing block size which is also used to create multiple of training block size
:return: returns list of training and testing indexes
```

extract_boostrap_periods(self, df, num_samples=10, start_sample_index=0, end_sample_index=None, sample_size=300, prop_block_boostrap=0.25
```
Function for selecting period

:param df: Data-frame
:param num_samples: Number of block samples
:param start_sample_index: Start of sample index
:param end_sample_index: End of sample index
:param sample_size: Minimum sample size length
:param prop_block_boostrap: Proportion of data used in each sample
:return: returns dictionary of start and end indexes
```

get_all_commod_returns(self)
```
Obtain returns for all instruments based on optimal leverage scaled to portfolio target and individual forecasts
```

get_all_opt_weights(self, path='research/optimize_weights/')
```
Obtain optimized weight for all files produced by method avg_optimized_sharpe_allinstr_single_period
```

get_combined_forecasts_all_instr(self, allocs_wts_forecasts=None)
```
Obtain combined forecasts for all instruments
:param allocs_wts_forecasts: np.array forecast weights. If none, equal weights are assigned to each forecast rule
```

get_combined_forecasts_single_instr(self, commod, allocs_wts_forecasts=None)
```
Obtain combined forecasts for single instrument
:param commod: Commodity symbol
:param allocs_wts_forecasts: np.array forecast weights. If none, equal weights are assigned to each forecast rule
```

get_commod_returns(self, commod)
```
Obtain returns for instrument based on optimal leverage scaled to portfolio target and individual forecasts
```

get_norm_breakout_info(self)
```
Obtain normalized donchian channel forecasts scaled to a range of -20 to +20
```

get_norm_ewmac_info(self)
```
Obtain normalized ewmac forecasts scaled to a range of -20 to +20
```

get_opt_weight_file(self, file_name, path='research/optimize_weights/')
```
Obtain optimized weight for single file produced by method avg_optimized_sharpe_allinstr_single_period
:param period: Indexes referenced to a dictionary with reference to period which bootstrap indexes are extracted
```

get_returns_data(self)
```
Obtain returns data from file; convert to price level that starts at 1
```

optimize_sharpe_single_instrument_period(self, commod, period, bootstrap_index)
```
Optimize sharpe in each bootstrap optimization and return dictionary of weights and performance. Optimize weight for ea
:param commod: Commodity symbol
:param period: Indexes referenced to a dictionary with reference to period which bootstrap indexes are extracted
:param boostrap_index: Indexes referenced to a dictionary with reference to bootstrap indexes referenced to self.price
```

select_period(self, df, start_date, end_date, index_date='date')
```
Select period in self.price data frame based on starting, ending date or indexes. indexes used in study
:param start_date: start date
:param end_date: end date
:param index_date: select by 'index' or 'date'
```

Data descriptors defined here:

__dict__
```
dictionary for instance variables (if defined)
```

__weakref__
```
list of weak references to the object (if defined)
```