Documentation

Documentation for ACE API Library

Table of Contents

ACE Package and instructions

Download our Alpha Creation Engine from here

The Alpha Creation Engine is designed to automate the process of simulating and analyzing Alphas via Python. Here's what's included:

- ace_lib.py - contains main functions that interact with the API.
- helpful_functions.py - provides additional functions. These functions transform outputs into dataframes and load data from the Data Explorer.
- How_to_use.ipynb - Jupyter notebook that demonstrates how to use the `ace_lib.py` and `helpful_functions.py` libraries for alpha simulation and results analysis.
- requirements.txt - lists all the Python modules and their respective versions needed to run our Alpha Creation Engine. This ensures you have the correct environment setup for smooth operation.
- readme.md - provides instructions on how to install the required Anaconda package.

ACE API functions (ace_lib.py)

get_credentials()

This function will read login name and password from file ~/secrets/platform-brain.json

~/secrets/platform-brain.json to store these information.

returns:

credentials: tuple[str, str]

Contains email and password. Example: (users@gmail.com, P@sSW0rd)

start_session()

This function helps you login and start and new session. This function will automatically call get_credentials to get login name and password.

returns:

s: requests.Session:

Object will be used to send requests

check_session_timeout(s)

This function will check number of remaining seconds until the session expires.

Parameters:

s: requests.Session:

Object will be used to send requests.

returns:

t: int

Number of remaining seconds until the session expires. t=0 if the session has expired.

generate_alpha(regular: str, selection: str, combo: str, region: str = "USA", universe: str = "TOP3000", neutralization: str = "INDUSTRY", delay: int = 1,

selection_handling: str, selection_limit: int, visualization: bool = False)

This function will take alpha expression and alpha setting parameters and return an object that can be used to send to simulation API.

Parameters:

regular: str

Alpha expression (for alpha_type = "REGULAR")

selection: str

Combo expression (for alpha_type = "SUPER")

combo: str

Selection expression (for alpha_type = "SUPER")

alpha_type: 'REGULAR' | 'SUPER'. Default: 'REGULAR'

Selected type of alpha

region: 'USA' | 'CHN' | 'EUR' | 'ASI' | 'KOR' | 'HKG' | 'TWN' | 'GLB'. Default: 'USA'

Selected region

universe: str. Default: 'TOP3000'

Selected universe for trading. Universe must be available in region:

'GLB' : 'TOP3000' | 'MINVOL1M'
'USA': 'TOP3000' | 'TOP1000' | 'TOP500' | 'TOP200' | 'ILLIQUID_MINVOL1M'
'EUR': 'TOP1200' | 'TOP800' | 'TOP400' | 'ILLIQUID_MINVOL1M'
'ASI': 'MINVOL1M' | 'ILLIDQUID_MINVOL1M'
'CHN': 'TOP3000' | 'TOP2000 ' | 'TOP2000U'
'KOR': 'TOP600'
'TWN': 'TOP500' | 'TOP100'
'HKG': 'TOP800' | 'TOP500'
'JPN': 'TOP1600' | 'TOP1200'
'AMR': 'TOP600'
Standard group is used for neutralization.

Second parameter in ts_decay_linear which is applied for the last alpha expression.

truncation: float. Default: 0.08

Maximum weight for each stock. 0 means no truncation.

nan_handling: 'ON' | 'OFF'. Default: 'P0Y0M0D'

Specifies the test period.

test_period :str: Default: 'VERIFY'

Raises a warning when incompatible units are used in an operator.

unit_handling: 'VERIFY'. Default: 'VERIFY'

Raises a warning when incompatible units are used in an operator.

pasteurization: 'ON' | 'OFF'. Default: 'ON'

Turn pasteurization on or off.

truncation: float. Default: 0.08

Maximum weight for each stock. 0 means no truncation.

selection_handling: 'POSITIVE' | 'NON_ZERO' | 'NON_NAN'. Default: 'POSITIVE'

Specify selection handling setting for alpha_type = "SUPER"

selection_limit: int. Default: 100

Specify limit of number of selected alphas for alpha_type = "SUPER"

visualization: bool. Default: False

Turn visualization on or off.

returns:

alpha: dict.

Alpha simulation result in JSON format

```
      "instrumentType": "EQUITY",
      "delay": "int",
      "universe": "str",
      "truncation": "float",
      "testPeriod": "P0Y0M0D",
      "unitHandling": "str",
      "pasteurization": "str",
      "region": "str",
      "language": "FASTEXPR",
      "decay": "int",
      "neutralization": "str",
      "visualization": "str"
   },
   "regular": "str"
}
```

construct_selection_expression(selection: str, instrument_type: str = "EQUITY", region: str = "USA", delay: int = 1, selection_limit: int = 1000, selection_handling: str = "POSITIVE")

This function constructs a selection expression dictionary based on the provided parameters.

Parameters:

selection: str

The selection criteria for the simulation.

instrument_type: str. Default: 'EQUITY'

The type of instrument for the simulation.

region: 'USA' | 'CHN' | 'EUR' | 'ASI' | 'KOR' | 'HKG' | 'TWN' | 'GLB'. Default: 'USA'

The region for the simulation.

selection_limit: int. Default: 1000

The maximum number of selections for the simulation. Default is 1000.

selection_handling: 'POSITIVE' | 'NON_ZERO' | 'NON_NAN'. Default: 'POSITIVE'

The handling strategy for the selections.

returns:

dict:

A dictionary containing the selection data.

run_selection(selection: str, instrument_type: str = "EQUITY", region: str = "USA", delay: int = 1, selection_limit: int = 1000, selection_handling: str = "POSITIVE")

This function constructs a selection expression dictionary based on the provided parameters.

Parameters:

s: requests.Session:

A session object to maintain certain parameters across requests.

selection_data: dict

A dictionary containing the selection data.

returns:

dict:

A dictionary containing the count of selected alphas and any messages returned by the selection operation.

start_simulation(s, simulate_data)

s: requests.Session:

Object will be used to send requests.

simulate_data: dict

Alpha that is created by generate_alpha function.

returns:

s: Simulate_Response: requests.Response:

Contains simulation progress url in headers["Location"]

simulation_progress(s, simulate_response)

This function wait until a single simulation finish and return simulation result.

Parameters:

s: requests.Session:

Object will be used to send requests.

s: Simulate_Response: requests.Response:

Contains simulation progress url in headers["Location"]

returns:

simulation_result: dict

```
{
   "completed":"bool",
   "results":"dict"
}
```

s: Simulate_Response: requests.Response:

multisimulation_progress(s, simulate_response)

This function wait until a multi-simulation finish and return simulation result.

Parameters:

s: requests.Session:

Object will be used to send requests.

s: Simulate_Response: requests.Response:

Contains simulation progress url in headers["Location"]

returns:

simulation_result: dict

```
{
   "completed":"bool",
   "results":"dict"
}
```

get_prod_corr(s, alpha_id)

This function get alpha production correlation by id of alpha.

Parameters:

s: requests.Session:

Object will be used to send requests.

alpha_id: str

Alpha's identifier

returns:

check_prod_corr_test(s, alpha_id, threshold: float = 0.7)

This function check if alpha can pass the production correlation test.

Parameters:

s: requests.Session:

Object will be used to send requests.

alpha_id: str

Alpha's identifier

threshold: float. Default: 0.7

Alpha pass the test cannot have production correlation larger than threshold.

returns:

s: result_df: pandas.DataFrame:

Contains 'test' (test name), 'result' (True or False), 'limit' (threshold), 'value' (max production correlation), 'alpha_id'.

get_self_corr(s, alpha_id)

This function get alpha self-correlation by id of alpha.

Parameters:

s: requests.Session:

Object will be used to send requests.

alpha_id: str

Alpha's identifier

Alpha self-correlation.

check_self_corr_test(s, alpha_id, threshold: float = 0.7)

simulation_progress(s, simulate_response)

This function get alpha self-correlation by id of alpha.

Parameters:

s: requests.Session:

Object will be used to send requests.

alpha_id: str

Alpha's identifier

threshold: float. Default: 0.7

Alpha pass the test cannot have production correlation larger than threshold.

returns:

s: result_df: pandas.DataFrame:

Contains 'test' (test name), 'result' (True or False), 'limit' (threshold), 'value' (max self-correlation), 'alpha_id'.

get_check_submission(s, alpha_id)

Check if alpha can be submitted.

Parameters:

s: requests.Session:

Alpha's identifier

returns:

s: result_df: pandas.DataFrame:

Contains IS test results.

performance_comparison(s, alpha_id, team_id: str = None, competition: str = None)

Compare merged alpha performance before and after submit alpha.

Parameters:

s: requests.Session:

Object will be used to send requests.

alpha_id: str

Alpha's identifier

team_id: str. Default: None competition: str. Default: None

If **competition** is not None, merged alpha performance is contributed from alpha that satisfy competition requirements.

If **competition** is None, merged alpha performance is contributed from alpha of all team's members (determined by **team_id**).

If **competition** and **team_id** are None, merged alpha performance is contributed from alpha of individual user.

returns:

result: dict

Comparison result.

Parameters:

s: requests.Session:

Object will be used to send requests.

alpha_id: str

Alpha's identifier

returns:

status: bool

Return True if successfully submit else return False.

get_simulation_result_json(s, alpha_id)

This function get alpha simulation result in JSON format.

Parameters:

s: requests.Session:

Object will be used to send requests.

alpha_id: str

Alpha's identifier

returns:

result_json: str

Alpha simulation result in JSON format

simulate_single_alpha(s, simulate_data)

This function wait until a single simulation finish and return simulation result.

Object will be used to send requests.

simulate_data: dict

Alpha expression and settings generated by generate_alpha function.

returns:

result: dict

Contains alpha's identifier and simulate data: {"alpha_id": str, "simulate_data": simulate_data}

simulate_multi_alpha(s, simulate_data_list)

This function wait until a single simulation finish and return simulation result.

Parameters:

s: requests.Session:

Object will be used to send requests.

simulate_data_list: List[dict]

List of simulate_data. Simulate data is a dict contains alpha expression and settings generated by generate_alpha function.

returns:

result: List[dict]

A list of dict objects. Each of them contains alpha's identifier and simulate data: {"alpha_id": str, "simulate_data": simulate_data}

get_specified_alpha_stats(s, alpha_id, simulate_data, get_pnl: bool = False, get_stats: bool = False, save_pnl_file: bool = False, save_stats_file: bool = False, save_result_file: bool = False, check_submission: bool = False, check_self_corr: bool = False, check_prod_corr: bool = False, name=None, color=None)

This function wait until a single simulation finish and return simulation result.

Parameters:

s: requests.Session:

Object will be used to send requests.

alpha_id: str

Alpha's identifier.

simulate_data: dict

Simulate data that will be wrapped into return alpha stats.

get_pnl: bool. Default: False

Return PnL or not. **get_stats: bool. Default: False**

Return stats or not. **save_pnl_file: bool. Default: False**

Save PnL to file or not. **save_stats_file: bool. Default: False**

Save stats to file or not.**save_result_file: bool. Default: False**

Save result to file or not.**check_submission: bool. Default: False**

Check submission or not.**check_self_corr: bool. Default: False**

Check self-correlation or not. **check_prod_corr: bool. Default: False**

Check production correlation or not.

name: str. Default: None

Alpha's name **color: str (must be a color). Default: None**

Alpha's color.

returns:

result: List[dict]

Contains 'alpha_id', 'simulate_data', 'is_stats', 'pnl', 'stats', 'is_tests'.

simulation_config=DEFAULT_CONFIG)

This function take a list of alpha and concurrently create multiple single simulations to simulate that alpha.

Parameters:

s: requests.Session:

Object will be used to send requests.

alpha_list: List[dict]

A list contains dict objects. Each object is a simulate_data object created by generated_alpha function.

limit_of_concurrent_simulations: int. Default: 3

Maximum number of concurrent simulations.

simulation_config: dict. Default: DEFAUT_CONFIG

Default flags indicate which information need to be returned by get_specified_alpha_stats function.

returns:

result: List[dict]

Contains result of each simulation. Result is an object created by get_specified_alpha_stats function.

simulate_alpha_list_multi(s, alpha_list, limit_of_concurrent_simulations=3, limit_of_multi_simulations=3, simulation_config=DEFAULT_CONFIG)

This function take a list of alpha and concurrently create multiple multi-simulations to simulate that alpha.

Parameters:

s: requests.Session:

A list contains dict objects. Each object is a simulate_data object created by generated_alpha function.

limit_of_concurrent_simulations: int. Default: 3

Maximum number of concurrent simulations.

limit_of_multi_simulations: int. Default: 3

Maximum number of simulation in a multi-simulation.

simulation_config: dict. Default: DEFAUT_CONFIG

Default flags indicate which information need to be returned by get_specified_alpha_stats function.

returns:

result: List[dict]

Contains result of each simulation. Result is an object created by get_specified_alpha_stats function.

Miscellanious functions (helpful_functions)

make_clickable_alpha_id(alpha_id)

Make alpha_id clickable in dataframes.

Parameters:

alpha_id: str

Alpha's identifier.

returns:

clickable_alpha_id: str

A HTML string contains reference to alpha.

False)

Combine needed results in one dataframes to analyze your alphas. Sort by fitness absolute value.

Parameters:

result: List[Dict]

A JSON-like object contains simulation results.

detailed_tests_view: bool. Default: False

If this parameter is False, test columns will just indicate your alpha pass or fail that test. Otherwise, test columns will contain limit, your alpha value and result (pass or fail).

clickable_alpha_id: bool. Default: False

If this parameter is True, alpha id in returned dataframe has a hyperlink to alpha page on BRAIN platform.

returns:

s: alpha_stats: pandas.DataFrame:

A dataframe contains alpha stats.

concat_pnl(result)

Combine needed results in one dataframe to analyze pnls of your alphas.

Parameters:

result: List[Dict]

A JSON-like object contains simulation results.

returns:

s: pnls_df: pandas.DataFrame:

A dataframe contains PnL.

Combine IS test results into a dataframe.

Parameters:

result: List[Dict]

A JSON-like object contains simulation results.

returns:

s: is_tests_df: pandas.DataFrame:

A dataframe contains IS test results.

save_simulation_result(result)

This function save simulation result as a JSON file in folder simulation_results . Filename is determined by id and region of alpha.

Parameters:

result: List[Dict]

result: JSON-like Simulation result.

save_pnl(pnl_df, alpha_id, region)

This function save PnL as a CSV file in folder alphas_pnl . Filename is determined by id and region of alpha.

Parameters:

s: pnl_df: pandas.DataFrame:

Alpha PnL.

alpha_id: str

Alpha's identifier.

save_yearly_stats(yearly_stats, alpha_id, region)

This function save yearly stats as a CSV file in folder yearly_stats . Filename is determined by id and region of alpha.

Parameters:

s: yearly_stats: pandas.DataFrame:

Alpha yearly stats.

alpha_id: str

Alpha's identifier.

alpha_region: str

Alpha's trading region

get_alpha_pnl(s, alpha_id)

This function get alpha PnL by id of alpha.

Parameters:

s: requests.Session:

Object will be used to send requests.

alpha_id: str

Alpha's identifier.

returns:

s: pnl_df: pandas.DataFrame:

Alpha PnL

This function get alpha yearly stats by id of alpha.

Parameters:

s: requests.Session:

Object will be used to send requests.

alpha_id: str

Alpha's identifier.

returns:

s: yearly_stats_df: pandas.DataFrame:

Alpha yearly stats.

set_alpha_properties(s, alpha_id, name: str = None, color: str = None, selection_desc: str = "None", combo_desc: str = "None", tag : str = "None")

This function change alpha's description parameters by id of alpha.

Parameters:

s: requests.Session:

Object will be used to send requests.

alpha_id: str

Alpha's identifier.

name: str. Default: None

name: str. Default: None

color: str. Default: None

Alpha's color.

selection_desc: str. Default: "None"

Super alpha's combo description.

tag: List[str]. Default: ["None"]

Alpha's tags.

get_datasets(s, instrument_type: str = 'EQUITY', region: str = 'USA', delay: int = 1, universe: str = 'TOP3000')

This function return datasets by instrument type, region, delay and universe.

Parameters:

s: requests.Session:

Object will be used to send requests.

instrument_type: str. Default: 'EQUITY'

Instrument type.

region: str. Default: 'USA'

Region

delay: int. Default: 1

Delay

universe: str. Default: 'TOP3000'

Universe.

returns:

s: datasets_df: pandas.DataFrame:

List of datasets

identifier and keywords.

Parameters:

s: requests.Session:

Object will be used to send requests.

instrument_type: str. Default: 'EQUITY'

Instrument type.

region: str. Default: 'USA'

Region

delay: int. Default: 1

Delay

universe: str. Default: 'TOP3000'

Universe.

dataset_id: str. Default: ''

Keywords.

returns:

s: datafieldss_df: pandas.DataFrame:

List of datafields

Prev: BRAIN API