Suggested Modifications

2.1 Module - infertrade.PandasEnum

PandasEnum module is used for providing the special string names used with the InferTrade interface.

Base:

- **Module:** enum.Enum

Class Name:

- **class** infertrade.PandasEnum.**PandasEnum**(*value*)

Purpose:

- To provide the strings for special column names that InferTrade uses in identifying pandas dataframe contents.

Abbreviation:

- **MID**: this is the mid-price used to calculate performance.
- **ALLOCATION**: the fraction of the overall portfolio the strategy wants to invest in the market. May differ from the amount invested where the strategy requires minimum deviations to trigger position adjustment.
- **VALUATION**: the value of strategy, after running a hypothetical rule implementing the strategy. 1.0 = 100% means no profit or loss. 0.9 = 90%, means a -10% cumulative loss. 1.1 = 110% means a 10% overall cumulative gain.
- **BID_OFFER_SPREAD**: the fractional bid-offer spread - 2 * (ask - bid)/(ask + bid) - for that time period.
- **SIGNAL**: an information time series that could be used for the construction of an allocation series.

Strings and their Synonyms:

- **ADJUSTED_CLOSE** = 'adjusted close'
- **ADJ_CLOSE** = 'adj close'
- **ADJ_DOT_CLOSE** = 'adj. close'
- **ALLOCATION** = 'allocation'
- **BID_OFFER_SPREAD** = 'bid_offer_spread'
- **CASH_INCREASE** = 'cash_flow'
- **CLOSE** = 'close'
- **FORECAST_PRICE_CHANGE** = 'forecast_price_change'
- **MID** = 'price'
- **PERCENTAGE_TO_BUY** = 'trade_percentage'
- **PERIOD_END_ALLOCATION** = 'end_of_period_allocation'
- **PERIOD_END_CASH** = 'period_end_cash'
- **PERIOD_END_SECURITIES** = 'period_end_securities'
- **PERIOD_START_ALLOCATION** = 'start_of_period_allocation'
- **PERIOD_START_CASH** = 'period_start_cash'
- **PERIOD_START_SECURITIES** = 'period_start_securities'
- **PRICE_SYNONYMS** = ['close', 'adjusted close', 'adj close', 'adj. close']
- **SECURITIES_BOUGHT** = 'security_purchases'
- **SIGNAL** = 'signal'
- **TRADING_SKIPPED** = 'trading_skipped'
- **VALUATION** = 'portfolio_return'

Validation:

- **To ensure that "Price" column is not missing**:
    - infertrade.PandasEnum.**create_price_column_from_synonym**(df_potentially_missing_price_column: pandas.core.frame.DataFrame)

2.2 Module - infertrade.api

API facade that allows interaction with the library with strings and vanilla Python objects.

Base: Object

Technical Requirements:

- All public methods should input/output JSON-serialisable dictionaries.

Codes & their usage:

- **Code that gets the list of algorithm types:**

```
static algorithm_categories() → List[str]
```

- **Code that gets the list of strings that are available strategies:**

```
static available_algorithms(filter_by_package: Optional[Union[str, List[str]]] = None,
                            filter_by_category: Optional[Union[str, List[str]]] = None
                            )→ list[str]
```

- **Code that gets the list of supported packages:**

```
static available_packages() → List[str]
```

- **Code that calculates the allocations using the supplied strategy:**

```
static calculate_allocations(df: pandas.core.frame.DataFrame, name_of_strategy: str,
                             name_of_price_series: str = 'price') →
                             pandas.core.frame.DataFrame
```

- **Code that calculates the returns using the supplied strategy:**

```
static calculate_allocations_and_returns(df: pandas.core.frame.DataFrame,
                                         name_of_strategy: str,
                                         name_of_price_series: str = 'price') →
                                         pandas.core.frame.DataFrame
```

- **Code that calculates the returns from supplied positions:**

```
static calculate_returns(df: pandas.core.frame.DataFrame)→ pandas.core.frame.DataFrame
```

- **Code that calculates the allocations using the supplied strategy:**

```
static calculate_signal(df: pandas.core.frame.DataFrame, name_of_signal: str) →
                        pandas.core.frame.DataFrame
```

- **Code that determines the original package of a strategy:**

```
static determine_package_of_algorithm(name_of_algorithm: str) → str
```

- **Code that provides information on algorithms (signals and positions) as flat list (not nested by category):**

```
static get_algorithm_information() → dict
```

- **Code that provides information on algorithms that calculate positions:**

```
static get_allocation_information() → dict
```

- **Code that describes which representations exist for the algorithm:**

```
static get_available_representations(name_of_algorithm: str) → List[str]
```

- **Code that provides information on algorithms that calculate signals:**

```
static get_signal_information() → dict
```

- **Code that describes the input columns needed for the strategy:**

```
static required_inputs_for_algorithm(name_of_strategy: str) → List[str]
```

- **Code that describes the input columns needed for the strategy:**

```
static required_parameters_for_algorithm(name_of_strategy: str) → List[str]
```

- **Code that returns the category of algorithm as a string:**

```
static return_algorithm_category(algorithm_name: str) → str
```

- **Code that returns the representations (e.g. URLs of relevant documentation):**

```
static return_representations(name_of_algorithm: str,
                              representation_or_list_of_representations:
                              Optional[Union[str, List[str]]] = None) → dict
```