Sample Code - Summer 2022

Neil Singh

May 24, 2022

```
# -*- coding: utf-8 -*-
"""
Author & Date Created
---------------------
Neil Singh
Monday, May 16 21:11:46 2022

Account
-------
A comprehensive one dimensional container of previously transacted investments
that can be easily extended to include future opportunities for the purposes
of simulation, back-testing, and data-driven analysis.

Dependencies
------------
data_import_export.py :
    Module for reading data from CSVs. Used to retrieve both
    a Dataframe containing account transaction history and a
    dict of str: Dataframe where str is a ticker symbol and Dataframe
    is historical ticker price and volume data.
market_constructor.py :
    Module for extending market data to encompass various fundamental
    financial metrics pertaining to ticker and containing extended data.
    Used to retrieve a dict of str: Business where str is a ticker and
    Business contains a complete financial record of that ticker.
"""
from typing import Union as U  # For allowing XOR in type contracts.
import datetime as dt  # For manipulating names of time series.

from pandas import DataFrame, Series, Timestamp  # Core data structures.
from numpy import float64  # For documenting DataFrame dtypes.

from market_constructor import Business  # For retrieving real-time data.


class Investment:
    """Information about an investment including date investment was
    transacted, action (buy or sell), symbol, description given by brokerage,
    quantity, price, gross amount paid, commission fees, net amount paid,
    currency paid.

    Parameters
    ----------
    transaction : Series or None, default None
        One dimensional ndarray with name Timestamp and dtype Object. When not
        None, contains security-specific transaction data.

    Examples
    --------
    >>> blank_investment = Investment()
    >>> isinstance(blank_investment, Investment)
    True
    >>> blank_investment.__dict__
    {'date': None,
     'act': None,
     'sym': None,
     'des': None,
     'qty': None,
     'prc': None,
     'grs': None,
     'com': None,
     'net': None,
     'cur': None}

    >>> account_data = get_account_data()
    >>> account_data
                     Action               Symbol  ... Net Amount  Currency
    Transaction Date                              ...
    2022-05-12         Sell                  TWO  ...    3067.18       USD
    2022-05-11         Sell               AQN.TO  ...     868.38       CAD
    2022-05-09          Buy                 FLNC  ...    -457.45       USD
    2022-04-22          Buy            INO-UN.TO  ...    -892.95       CAD
    2022-04-06         Sell  AQN16Sep22C17.00.MX  ...     962.05       CAD
    2022-03-31         Sell     XLU19Jan24C60.00  ...    3248.03       USD
    ...                 ...                  ...  ...        ...       ...
    2021-02-22          Buy               LAC.TO  ...     -56.62       CAD
```

```
2021-02-10          Buy                 LABU  ...    -361.12       USD

[59 rows x 9 columns]

The get_account_data() function of module data_import_export
will return a DataFrame containing an account transaction history
provided by Questrade.

>>> transaction_share = account_data.iloc[2]
>>> transaction_share
Action                                                        Buy
Symbol                                                       FLNC
Description     FLUENCE ENERGY INC CLASS A COMMON STOCK WE ACT...
Quantity                                                     50.0
Price                                                        9.05
Gross Amount                                               -452.5
Commission                                                  -4.95
Net Amount                                                -457.45
Currency                                                      USD
Name: 2022-05-09 00:00:00, dtype: object

>>> flnc_share = Investment(transaction_share)
>>> flnc_share.__dict__
{'date': Timestamp('2022-05-09 00:00:00'),
 'act': 'Buy',
 'sym': 'FLNC',
 'des': 'FLUENCE ENERGY INC CLASS A COMMON STOCK WE ACTED AS AGENT',
 'qty': 50.0,
 'prc': 9.05,
 'grs': -452.5,
 'com': -4.95,
 'net': -457.45,
 'cur': 'USD'}

>>> transaction_call = account_data.iloc[5]
>>> transaction_call
Action                                                       Sell
Symbol                                           XLU19Jan24C60.00
```

```
    Description     CALL XLU 01/19/24 60 SELECT SCTR SPDR AMEX UTL...
    Quantity                                                      -2.0
    Price                                                         16.3
    Gross Amount                                                3260.0
    Commission                                                  -11.97
    Net Amount                                                 3248.03
    Currency                                                       USD
    Name: 2022-03-31 00:00:00, dtype: object

    >>> xlu_call = Investment(transaction_call)
    >>> xlu_call.__dict__
    {'date': Timestamp('2022-03-31 00:00:00'),
     'act': 'Sell',
     'sym': 'XLU19Jan24C60.00',
     'des': 'CALL XLU 01/19/24 60 SELECT SCTR SPDR AMEX
             UTLTS SL WE ACTED AS AGENT',
     'qty': -2.0,
     'prc': 16.3,
     'grs': 3260.0,
     'com': -11.97,
     'net': 3248.03,
     'cur': 'USD'}
    """

    def __init__(self, transaction: U[Series, None] = None):
        self.date = None
        self.act = None
        self.sym = None
        self.des = None
        self.qty = None
        self.prc = None
        self.grs = None
        self.com = None
        self.net = None
        self.cur = None

        if isinstance(transaction, Series):
            self.date = self.get_date(transaction)
            self.act = self.get_action(transaction)
            self.sym = self.get_symbol(transaction)
            self.des = self.get_description(transaction)
            self.qty = self.get_quantity(transaction)
            self.prc = self.get_price(transaction)
            self.grs = self.get_gross_amount(transaction)
            self.com = self.get_commission(transaction)
            self.net = self.get_net_amount(transaction)
            self.cur = self.get_currency(transaction)

    def get_date(self, transaction: Series) -> Timestamp:
        """Returns the date of account transaction.

        >>> investment = Investment()
        >>> transaction = pd.Series(dtype=object, name=dt.date(2022, 5, 24))
        >>> transaction
        Series([], Name: 2022-05-24, dtype: object)
        >>> investment.date = investment.get_date(transaction)
        >>> investment.date
        datetime.date(2022, 5, 24)
        """
        date = transaction.name
        return date

    def get_action(self, transaction: Series) -> str:
        """Returns the action of an investment.

        >>> investment = Investment()
        >>> transaction = pd.Series(
                {'Action': 'Buy'},
                dtype=object,
                name=dt.date(2022, 5, 24)
                )
        >>> transaction
        Action    Buy
        Name: 2022-05-24, dtype: object
        >>> investment.act = investment.get_action(transaction)
        >>> investment.act
```

```python
        'Buy'
        """
        action = transaction[0]
        return action

    def get_symbol(self, transaction: Series) -> str:
        """Returns the symbol of an investment. If the investment is a share,
        the symbol is simply the ticker. If the investment is an option, the
        symbol is a consolidated representation of the underlying, expirery,
        type and strike price.

        >>> investment = Investment()
        >>> transaction = pd.Series(
                {'Action': 'Sell',
                 'Symbol': 'XLU19Jan24C60.00',
                 'Description': ('CALL XLU 01/19/24 60 SELECT SCTR '
                                 + 'SPDR AMEX UTLTS SL WE ACTED AS AGENT'),
                 'Quantity': -2.0,
                 'Price': 16.3,
                 'Gross Amount': 3260.0,
                 'Commission': -11.97,
                 'Net Amount': 3248.03,
                 'Currency': 'USD'},
                dtype=object,
                name=dt.date(2022, 5, 24)
                )
        >>> investment.sym = investment.get_symbol(transaction)
        >>> investment.sym
        'XLU19Jan24C60.00'
        """
        symbol = transaction[1]
        return symbol

    def get_description(self, transaction: Series) -> str:
        """Returns the action of an investment.

        >>> investment = Investment()
        >>> transaction = pd.Series(
                {'Action': 'Buy',
                 'Symbol': 'FLNC',
                 'Description': ('FLUENCE ENERGY INC CLASS A '
                                 + 'COMMON STOCK WE ACTED AS AGENT'),
                 'Quantity': 50.0,
                 'Price': 9.05,
                 'Gross Amount': -452.5,
                 'Commission': -4.95,
                 'Net Amount': -457.45,
                 'Currency': 'USD'},
                dtype=object,
                name=dt.date(2022, 5, 24)
                )
        >>> investment.des = investment.get_description(transaction)
        >>> investment.des
        'FLUENCE ENERGY INC CLASS A COMMON STOCK WE ACTED AS AGENT'
        """
        description = transaction[2]
        return description

    def get_quantity(self, transaction: Series) -> float64:
        """Returns the quantity of asset transacted.

        >>> investment = Investment()
        >>> transaction = pd.Series(
                {'Action': 'Buy',
                 'Symbol': 'FLNC',
                 'Description': ('FLUENCE ENERGY INC CLASS A '
                                 + 'COMMON STOCK WE ACTED AS AGENT'),
                 'Quantity': 50.0,
                 'Price': 9.05,
                 'Gross Amount': -452.5,
                 'Commission': -4.95,
                 'Net Amount': -457.45,
                 'Currency': 'USD'},
                dtype=object,
                name=dt.date(2022, 5, 24)
                )
        >>> investment.qty = investment.get_quantity(transaction)
        >>> investment.qty
        50.0
        """
        quantity = transaction[3]
        return quantity

    def get_price(self, transaction: Series) -> float64:
        """Returns the price asset was transacted at.

        >>> investment = Investment()
        >>> transaction = pd.Series(
                {'Action': 'Buy',
                 'Symbol': 'FLNC',
                 'Description': ('FLUENCE ENERGY INC CLASS A '
                                 + 'COMMON STOCK WE ACTED AS AGENT'),
                 'Quantity': 50.0,
                 'Price': 9.05,
                 'Gross Amount': -452.5,
                 'Commission': -4.95,
                 'Net Amount': -457.45,
                 'Currency': 'USD'},
                dtype=object,
                name=dt.date(2022, 5, 24)
                )
        >>> investment.prc = investment.get_price(transaction)
        >>> investment.prc
        9.05
        """
        price = transaction[4]
        return price

    def get_gross_amount(self, transaction: Series) -> float64:
        """Returns the gross amount of cash exchanged for asset.

        >>> investment = Investment()
        >>> transaction = pd.Series(
                {'Action': 'Buy',
                 'Symbol': 'FLNC',
                 'Description': ('FLUENCE ENERGY INC CLASS A '
                                 + 'COMMON STOCK WE ACTED AS AGENT'),
                 'Quantity': 50.0,
                 'Price': 9.05,
                 'Gross Amount': -452.5,
                 'Commission': -4.95,
                 'Net Amount': -457.45,
                 'Currency': 'USD'},
                dtype=object,
                name=dt.date(2022, 5, 24)
                )
        >>> investment.grs = investment.get_gross_amount(transaction)
        >>> investment.grs
        -452.5
        """
        gross_amount = transaction[5]
        return gross_amount

    def get_commission(self, transaction: Series) -> float64:
        """Returns the commission paid to brokerage for transaction.

        >>> investment = Investment()
        >>> transaction = pd.Series(
                {'Action': 'Buy',
                 'Symbol': 'FLNC',
                 'Description': ('FLUENCE ENERGY INC CLASS A '
                                 + 'COMMON STOCK WE ACTED AS AGENT'),
                 'Quantity': 50.0,
                 'Price': 9.05,
                 'Gross Amount': -452.5,
                 'Commission': -4.95,
                 'Net Amount': -457.45,
                 'Currency': 'USD'},
                dtype=object,
                name=dt.date(2022, 5, 24)
                )
        >>> investment.com = investment.get_commission(transaction)
        >>> investment.com
        -4.95
        """
        commission = transaction[6]
        return commission

    def get_net_amount(self, transaction: Series) -> float64:
        """Returns the gross amount paid plus commission.

        >>> investment = Investment()
        >>> transaction = pd.Series(
                {'Action': 'Buy',
                 'Symbol': 'FLNC',
                 'Description': ('FLUENCE ENERGY INC CLASS A '
                                 + 'COMMON STOCK WE ACTED AS AGENT'),
                 'Quantity': 50.0,
                 'Price': 9.05,
                 'Gross Amount': -452.5,
                 'Commission': -4.95,
                 'Net Amount': -457.45,
                 'Currency': 'USD'},
                dtype=object,
                name=dt.date(2022, 5, 24)
                )
        >>> investment.net = investment.get_net_amount(transaction)
        >>> investment.net
        -457.45
        """
        net_amount = transaction[7]
        return net_amount

    def get_currency(self, transaction: Series) -> str:
        """Returns the currency used for transaction.

        >>> investment = Investment()
        >>> transaction = pd.Series(
                {'Action': 'Buy',
                 'Symbol': 'FLNC',
                 'Description': ('FLUENCE ENERGY INC CLASS A '
                                 + 'COMMON STOCK WE ACTED AS AGENT'),
                 'Quantity': 50.0,
                 'Price': 9.05,
                 'Gross Amount': -452.5,
                 'Commission': -4.95,
                 'Net Amount': -457.45,
                 'Currency': 'USD'},
                dtype=object,
                name=dt.date(2022, 5, 24)
                )
        >>> investment.cur = investment.get_currency(transaction)
        >>> investment.cur
        'USD'
        """
        currency = transaction[8]
        return currency


class Option(Investment):
    """Information about an option including type, multiplier, expirery date,
    underlying asset ticker, strike price, days until expirery,
    intrinsic value, time value, and present value which is calculated to be
    the mid point of last bid and last ask. Option extends Investment if and
    only if there exists transaction data collected from brokerage account
    transaction history.

    Parameters
    ----------
    security : str, default ''
        Either 'CALL' or 'PUT'.
    transaction : DataFrame or None, default None.
        One dimensional ndarray with name Timestamp and dtype Object.
    market : Dict or None, default None.
        Dict with key of str representing ticker and value of object
        Business containing key financial data pertaining to ticker.

    Examples
```

```
--------
>>> blank_call = Option('CALL')
>>> isinstance(blank_call, Option)
True
>>> blank_call.__dict__
{'date': None,
 'act': None,
 'sym': None,
 'des': None,
 'qty': None,
 'prc': None,
 'grs': None,
 'com': None,
 'net': None,
 'cur': None,
 'security': 'CALL',
 'mult': 100,
 'exp': None,
 'under': None,
 'strk': None,
 'dte': None,
 'i_val': None,
 't_val': None,
 'p_val': None}

>>> market_data = get_market_data()
>>> market_data
{'AQN.TO':              Open   High    Low  Close  Adj Close   Volume
 Date
 2020-11-05  20.80  20.92  20.67  20.72      19.43  1153000
 2020-11-06  20.76  20.90  20.65  20.85      19.55   763400
 2020-11-09  21.33  21.58  20.93  20.96      19.65  1616300
 2020-11-10  21.12  21.47  21.09  21.22      19.90  2365600
 2020-11-11  21.39  21.73  21.31  21.39      20.06  1739100
 ...           ...    ...    ...    ...        ...      ...
 2022-05-17  18.17  18.49  18.15  18.43      18.43  1644900
 2022-05-18  18.41  18.70  18.27  18.36      18.36  1798500
 2022-05-19  18.25  18.58  18.21  18.47      18.47  2182000
 2022-05-20  18.45  18.53  18.31  18.51      18.51  1523300
 2022-05-24  18.62  18.75  18.55  18.64      18.64   555214

 [388 rows x 6 columns],
 ...:          ...    ...    ...    ...        ...      ...

 [... rows x 6 columns],

 'XLU':              Open   High    Low  Close  Adj Close    Volume
 Date
 2020-11-05  64.37  65.44  64.22  64.27      61.40  15762600
 2020-11-06  64.31  64.88  63.96  64.13      61.27   9477700
 2020-11-09  65.99  67.93  65.26  65.32      62.41  23036100
 2020-11-10  65.67  66.45  65.40  66.26      63.30  15586000
 2020-11-11  66.60  67.10  66.25  66.51      63.54   9729500
 ...           ...    ...    ...    ...        ...       ...
 2022-05-18  72.51  72.66  71.50  71.69      71.69  18728300
 2022-05-19  71.36  71.86  70.53  71.54      71.54  18276100
 2022-05-20  71.73  71.94  70.75  71.74      71.74  15678600
 2022-05-23  72.56  72.93  71.88  72.60      72.60  13528700
 2022-05-24  72.74  73.04  72.18  72.92      72.92   4230956

 [390 rows x 6 columns]}

>>> market = construct_market(market_data)
>>> market
{'AQN.TO': <__main__.Business at 0x23a128f50d0>,
 'BIP': <__main__.Business at 0x23a0d5a54f0>,
 'EIX': <__main__.Business at 0x23a12ba3610>,
 'ENB': <__main__.Business at 0x23a1270ed00>,
 'ERX': <__main__.Business at 0x23a126e08b0>,
 'FAS': <__main__.Business at 0x23a12ad1760>,
 'FLNC': <__main__.Business at 0x23a12863af0>,
 'GNE': <__main__.Business at 0x23a12c3dac0>,
 'INO-UN.TO': <__main__.Business at 0x23a12c1a910>,
 'LABU': <__main__.Business at 0x23a12ac21c0>,
 'LAC.TO': <__main__.Business at 0x23a1290a6a0>,
 'PPL': <__main__.Business at 0x23a12b88af0>,
 'SJI': <__main__.Business at 0x23a12cd5490>,
 'SPG': <__main__.Business at 0x23a12ad86a0>,
 'STEM': <__main__.Business at 0x23a128c2490>,
 'TWO': <__main__.Business at 0x23a12b58310>,
 'UTSL': <__main__.Business at 0x23a1261fdf0>,
 'VICI': <__main__.Business at 0x23a12ad5f40>,
 'XLU': <__main__.Business at 0x23a12ad5190>}

>>> account_data = get_account_data()
>>> account_data
                 Action               Symbol  ... Net Amount  Currency
Transaction Date                              ...                     
2022-05-12         Sell                  TWO  ...    3067.18       USD
2022-05-11         Sell               AQN.TO  ...     868.38       CAD
2022-05-09          Buy                 FLNC  ...    -457.45       USD
2022-04-22          Buy            INO-UN.TO  ...    -892.95       CAD
2022-04-06         Sell  AQN16Sep22C17.00.MX  ...     962.05       CAD
2022-03-31         Sell     XLU19Jan24C60.00  ...    3248.03       USD
2022-03-07          Buy                  TWO  ...   -1012.53       USD
2022-02-25          Buy                 STEM  ...    -634.95       USD
2022-02-10          Buy                  TWO  ...   -1084.95       USD
2022-02-09          Buy                 FLNC  ...    -339.95       USD
2022-02-03          Buy                 STEM  ...    -299.83       USD
2022-02-03          Buy                 FLNC  ...    -494.75       USD
2022-01-26         Sell    STEM19Jan24P10.00  ...    1186.04       USD
...                 ...                  ...  ...        ...       ...
2021-02-22          Buy               LAC.TO  ...     -56.62       CAD
2021-02-10          Buy                 LABU  ...    -361.12       USD

[59 rows x 9 columns]

>>> transaction_call = account_data.iloc[4]
>>> transaction_call
Action                                                       Sell
Symbol                                        AQN16Sep22C17.00.MX
Description     CALL .AQN 09/16/22 17 ALGONQUIN POWER & UTILIT...
Quantity                                                     -3.0
Price                                                        3.25
```

```
Gross Amount                                                 975.0
Commission                                                  -12.95
Net Amount                                                  962.05
Currency                                                       CAD
Name: 2022-04-06 00:00:00, dtype: object

>>> security = transaction_call[2].split()[0]
>>> security
'CALL'
>>> aqnto_call = Option(security, transaction_call, market)
>>> aqnto_call.__dict__
{'date': Timestamp('2022-04-06 00:00:00'),
 'act': 'Sell',
 'sym': 'AQN16Sep22C17.00.MX',
 'des': 'CALL .AQN 09/16/22 17 ALGONQUIN POWER &
         UTILITIES WE ACTED AS AGENT',
 'qty': -3.0,
 'prc': 3.25,
 'grs': 975.0,
 'com': -12.95,
 'net': 962.05,
 'cur': 'CAD',
 'type': 'CALL',
 'mult': 100,
 'exp': '2022-09-16',
 'under': 'AQN.TO',
 'strk': 17.0,
 'dte': 117,
 'i_val': 1.5100000000000016,
 't_val': 0,
 'p_val': 151.00000000000017}

>>> stem_put = account_data.iloc[12]
>>> stem_put
Action                                                      Sell
Symbol                                         STEM19Jan24P10.00
Description     PUT STEM 01/19/24 10 STEM INC WE ACTED AS AGENT
Quantity                                                    -4.0
```

```
    Price                                                       3.0
    Gross Amount                                             1200.0
    Commission                                               -13.96
    Net Amount                                              1186.04
    Currency                                                    USD
    Name: 2022-01-26 00:00:00, dtype: object

    >>> security = transaction_put[2].split()[0]
    >>> security
    'PUT'
    >>> stem_put = Option(security, transaction_put, market)
    >>> stem_put.__dict__
    {'date': Timestamp('2022-01-26 00:00:00'),
     'act': 'Sell',
     'sym': 'STEM19Jan24P10.00',
     'des': 'PUT STEM 01/19/24 10 STEM INC WE ACTED AS AGENT',
     'qty': -4.0,
     'prc': 3.0,
     'grs': 1200.0,
     'com': -13.96,
     'net': 1186.04,
     'cur': 'USD',
     'type': 'PUT',
     'mult': 100,
     'exp': '2024-01-19',
     'under': 'STEM',
     'strk': 10.0,
     'dte': 607,
     'i_val': 2.41,
     't_val': 0.040000000000000036,
     'p_val': 245.00000000000003}
    """

    def __init__(
            self,
            security: str = '',
            transaction: U[DataFrame, None] = None,
            market: U[dict[str, DataFrame], None] = None
            ):
        super().__init__(transaction)

        self.security = security
        self.mult = 100
        self.exp = None
        self.under = None
        self.strk = None
        self.dte = None
        self.i_val = None
        self.t_val = None
        self.p_val = None

        if isinstance(self.des, str):
            description = self.des.split()
            self.under = self.get_underlying(description)
            self.exp = self.get_expirery_date(description)
            self.strk = self.get_strike_price(description)
            self.dte = self.get_days_to_expirery()

        if isinstance(market, dict):
            business = market[self.under]
            self.i_val = self.get_intrinsic_value(business.last_p)
            self.t_val = self.get_time_value(business)
            self.p_val = self.get_present_value_estimate()

    def get_underlying(self, description: list[str]) -> str:
        """Return the underlying ticker within the description of an option.
        Formats ticker if there is a misplaced '.'.

        >>> call = Option('CALL')
        >>> description = ('CALL .AQN 09/16/22 17 ALGONQUIN POWER & '
                           + 'UTILITIES WE ACTED AS AGENT').split()
        >>> call.get_underlying(description)
        'AQN.TO'
        """
        ticker = description[1]
        if '.' in ticker:
            return ticker.lstrip('.') + '.TO'
        return ticker

    def get_expirery_date(self, description: list[str]) -> dt.date:
        """Return the expirery date in ISO format yyyy-mm-dd based on
        date in description.

        >>> put = Option('PUT')
        >>> description = ('PUT STEM 01/19/24 10 STEM '
                           + 'INC WE ACTED AS AGENT').split()
        >>> put.get_expirery_date(description)
        datetime.date(2024, 1, 19)
        """
        expirery = dt.datetime.strptime(description[2], '%m/%d/%y').date()
        return expirery

    def get_strike_price(self, description: list[str]) -> float64:
        """Return the strike price of an option determined by description

        >>> call = Option('CALL')
        >>> description = ('CALL .AQN 09/16/22 17 ALGONQUIN POWER & '
                           + 'UTILITIES WE ACTED AS AGENT').split()
        >>> call.get_strike_price(description)
        17.0
        """
        return float64(description[3])

    def get_days_to_expirery(self) -> int:
        """Return the integer number of days to expirery. Return 0 if already
        expired.

        >>> put = Option('PUT')
        >>> description = ('PUT STEM 01/19/24 10 STEM '
                           + 'INC WE ACTED AS AGENT').split()
        >>> put.exp = put.get_expirery_date(description)
        >>> put.get_days_to_expirery()
        605
        """
```

```python
        today = dt.date.today()
        total = self.exp - today
        return max(0, total.days)

    def get_intrinsic_value(self, last_p: float64) -> float64:
        """Return the intrinsic value of an option based on last_p.

        >>> market_data = get_market_data()
        >>> market = construct_market(market_data)
        >>> account_data = get_account_data()
        >>> transaction_put = account_data.iloc[12]
        >>> stem_put = Option('PUT', transaction_put, market)
        >>> underlying_price = market['STEM'].last_p
        >>> underlying_price
        7.11
        >>> stem_put.get_intrinsic_value(underlying_price)
        2.8899999999999997
        """
        if self.security == 'CALL':
            return max(0, last_p - self.strk)
        else:
            return max(0, self.strk - last_p)

    def get_time_value(self, business: Business) -> float64:
        """Return the time value of an option on underlying business.

        >>> market_data = get_market_data()
        >>> market = construct_market(market_data)
        >>> account_data = get_account_data()
        >>> transaction_call = transaction_call = account_data.iloc[5]
        >>> xlu_call = Option('CALL', transaction_call, market)
        >>> business = market['XLU']
        >>> xlu_call.get_time_value(business) # strike price not in range.
        0
        """
        if business.options is None:
            business.get_option_chain(self.exp)
            if business.options is None:
                return 0
        if self.security == 'CALL':
            chain = business.options.calls
        else:
            chain = business.options.puts
        option = chain[chain['strike'] == self.strk]
        bid = option['bid'].sum()
        ask = option['ask'].sum()
        mid = (bid + ask) / 2
        return max(0, mid - self.i_val)

    def get_present_value_estimate(self) -> float64:
        """Return the sum intrinsic value and time value of an option,
        multiplied by 100

        >>> an_option = Option('CALL')
        >>> an_option.i_val = float64(10)
        >>> an_option.t_val = float64(5)
        >>> an_option.get_present_value_estimate()
        1500.0
        """
        return (self.i_val + self.t_val) * self.mult


class Share(Investment):
    """Information about a share including type, present value, dividend
    history, and dividend yield. Share extends Investment if and
    only if there exists transaction data collected from brokerage account
    transaction history.

    Parameters
    ----------
    security : string, default 'SHARE'
        Always 'Share'
    transaction : DataFrame or None, default None.
        One dimensional ndarray with name Timestamp and dtype Object.
    market : Dict of string : Dataframe or None, default None
        Dict with key of str representing ticker and value of object
        Business.

    Examples
    --------
    >>> blank_share = Share()
    >>> isinstance(blank_share, Share)
    True

    >>> market_data = get_market_data()
    >>> market = construct_market(market_data)
    >>> account_data = get_account_data()
    >>> account_data
                     Action               Symbol  ... Net Amount  Currency
    Transaction Date                              ...
    2022-05-12         Sell                  TWO  ...    3067.18       USD
    2022-05-11         Sell               AQN.TO  ...     868.38       CAD
    2022-05-09          Buy                 FLNC  ...    -457.45       USD
    2022-04-22          Buy            INO-UN.TO  ...    -892.95       CAD
    2022-04-06         Sell  AQN16Sep22C17.00.MX  ...     962.05       CAD
    2022-03-31         Sell     XLU19Jan24C60.00  ...    3248.03       USD
    2022-03-07          Buy                  TWO  ...   -1012.53       USD
    2022-02-25          Buy                 STEM  ...    -634.95       USD
    2022-02-10          Buy                  TWO  ...   -1084.95       USD
    2022-02-09          Buy                 FLNC  ...    -339.95       USD
    2022-02-03          Buy                 STEM  ...    -299.83       USD
    2022-02-03          Buy                 FLNC  ...    -494.75       USD
    2022-01-26         Sell    STEM19Jan24P10.00  ...    1186.04       USD
    ...                 ...                  ...  ...        ...       ...
    2021-02-22          Buy               LAC.TO  ...     -56.62       CAD
    2021-02-10          Buy                 LABU  ...    -361.12       USD

    [59 rows x 9 columns]

    >>> transaction_share = account_data.iloc[0]
    >>> transaction_share
    Action                                                     Sell
    Symbol                                                      TWO
    Description     TWO HARBORS INVESTMENT CORP COMMON STOCK WE AC...
```

```
Quantity                                                   -600.0
Price                                                      5.1221
Gross Amount                                              3073.26
Commission                                                  -6.08
Net Amount                                                3067.18
Currency                                                      USD
Name: 2022-05-12 00:00:00, dtype: object

>>> two = Share(transaction=transaction_share, market=market)
>>> two.__dict__
{'date': Timestamp('2022-05-12 00:00:00'),
 'act': 'Sell',
 'sym': 'TWO',
 'des': 'TWO HARBORS INVESTMENT CORP COMMON STOCK WE ACTED AS AGENT',
 'qty': -600.0,
 'prc': 5.1221,
 'grs': 3073.26,
 'com': -6.08,
 'net': 3067.18,
 'cur': 'USD',
 'type': 'SHARE',
 'p_val': 4.94,
 'div': Date
 2009-12-29    0.396537
 2010-03-29    0.549051
 2010-06-28    0.503297
 2010-09-28    0.594805
 ...                ...
 2020-12-29    0.170000
 2021-03-26    0.170000
 2021-06-28    0.170000
 2021-09-30    0.170000
 2021-12-28    0.170000
 2022-04-01    0.170000
 Name: Dividends, dtype: float64,
 'yld': 0.14}
"""
```

```
    def __init__(
            self,
            security: str = 'SHARE',
            transaction: U[DataFrame, None] = None,
            market: U[dict[str, DataFrame], None] = None
            ):
        super().__init__(transaction)

        self.security = security
        self.p_val = None
        self.div = None
        self.yld = None

        if isinstance(market, dict) and isinstance(security, str):
            business = market[self.sym]
            self.p_val = self.get_present_value(business)
            self.div = self.get_dividend(business)
            self.yld = self.get_yield()

    def get_present_value(self, business: Business) -> float64:
        """Return the last traded share price of a given business.

        >>> market_data = get_market_data()
        >>> market = construct_market(market_data)
        >>> market['FLNC'].last_p
        8.78
        >>> flnc = Share()
        >>> flnc.get_present_value(market['FLNC'])
        8.78
        """
        return business.last_p

    def get_dividend(self, business: Business) -> Series:
        """Return the trailing twelve month dividend.

        >>> two = Share()
        >>> market_data = get_market_data()
        >>> market = construct_market(market_data)
        >>> two.get_dividend(market['TWO'])
        Date
        2009-12-29    0.396537
        2010-03-29    0.549051
        2010-06-28    0.503297
        2010-09-28    0.594805
        ...                ...
        2020-12-29    0.170000
        2021-03-26    0.170000
        2021-06-28    0.170000
        2021-09-30    0.170000
        2021-12-28    0.170000
        2022-04-01    0.170000
        Name: Dividends, dtype: float64
        """
        if business.dividends is None:
            dividends = business.get_dividends()
            return dividends
        return business.dividends

    def get_yield(self):
        """Return the dividend yield all dividends within the last 12 months.

        >>> two = Share()
        >>> market_data = get_market_data()
        >>> market = construct_market(market_data)
        >>> two.p_val = two.get_present_value(market['TWO'])
        >>> two.div = two.get_dividend(market['TWO'])
        >>> two.get_yield()
        0.14
        """
        payout_days = self.div.index
        number_of_payouts = len(payout_days)
        i = -1
        if number_of_payouts == 0:
            return 0
        elif number_of_payouts > 1:
            ttm_start = dt.timedelta(0)
            ttm_end = dt.timedelta(365)
            delta = payout_days[i] - payout_days[i-1]
            while ttm_start + delta < ttm_end:
                ttm_start += delta
                i -= 1
                try:
                    delta = (payout_days[i] - payout_days[i-1])
                except IndexError:
                    delta = ttm_end
        ttm = self.div[i:].sum()
        yld = ttm / self.p_val
        return round(yld, 2)


def print_portfolio(portfolio: dict[dict[str, float]]) -> None:
    """Print the current holdings of a portfolio along with quantity,
    book value, market value, average price, last price, and % P&L.

    >>> market_data = get_market_data()
    >>> market = construct_market(market_data)
    >>> account_data = get_account_data()
    >>> account = construct_account(market, account_data)
    >>> portfolio = get_portfolio(account, account_data)
    >>> print_portfolio(portfolio)
                       Quantity  Book Value   ...   Last Price   % P&L
    FLNC                  150.0    -2957.30   ...         8.95  -54.60
    INO-UN.TO             100.0     -888.00   ...         7.87  -11.37
    STEM                  175.0    -2363.69   ...         7.59  -43.81
    STEM19Jan24C15.00       6.0    -4020.00   ...         0.00 -100.00

    Note that the above was consolidated to fit within character limit.
    """
    df = DataFrame.from_dict(portfolio).transpose()
    print(df.to_string())


def get_portfolio(
        account: list[U[Option, Share]],
        account_data: DataFrame
        ) -> dict[dict[str, float]]:
    """Return a dictionary containing the current holdings within an account.

    >>> market_data = get_market_data()
    >>> market = construct_market(market_data)
    >>> account_data = get_account_data()
    >>> account = construct_account(market, account_data)
    >>> portfolio = get_portfolio(account, account_data)
    >>> portfolio
    {'FLNC': {'Quantity': 150.0,
      'Book Value': -2957.3,
      'Market Value': 1342.5,
      'Avg Price': 19.72,
      'Last Price': 8.95,
      '% P&L': -54.6},
     'INO-UN.TO': {'Quantity': 100.0,
      'Book Value': -888.0,
      'Market Value': 787.0,
      'Avg Price': 8.88,
      'Last Price': 7.87,
      '% P&L': -11.37},
     'STEM': {'Quantity': 175.0,
      'Book Value': -2363.69,
      'Market Value': 1328.25,
      'Avg Price': 13.51,
      'Last Price': 7.59,
      '% P&L': -43.81},
     'STEM19Jan24C15.00': {'Quantity': 6.0,
      'Book Value': -4020.0,
      'Market Value': 0.0,
      'Avg Price': 670.0,
      'Last Price': 0,
      '% P&L': -100.0}}
    """
    portfolio = {}
    for inv in account:
        if inv.sym not in portfolio:
            subset = account_data[account_data['Symbol'] == inv.sym]
            quantity = subset['Quantity'].sum()
            if quantity > 0:
                book_val = subset['Gross Amount'].sum()
                avg = round(-1 * book_val / quantity, 2)
                market_val = quantity * inv.p_val
                p_l = round(100 * (market_val + book_val) / (-1 * book_val), 2)
                portfolio[inv.sym] = {
                    'Quantity': quantity,
                    'Book Value': book_val,
                    'Market Value': market_val,
                    'Avg Price': avg,
                    'Last Price': inv.p_val,
                    '% P&L': p_l,
                    }
    return portfolio


def construct_account(
        market: dict[str, Business],
        account_data: DataFrame
        ) -> list[U[Option, Share]]:
    """Return a list of past and current investments with real-time data
    provided by market and investment history contain in account_data.

    >>> market_data = get_market_data()
    >>> market = construct_market(market_data)
    >>> account_data = get_account_data()
    >>> account = construct_account(market, account_data)
    >>> account
    [<__main__.Share at 0x159e13298b0>,
     <__main__.Share at 0x159e13299d0>,
     <__main__.Share at 0x159e1329d90>,
     <__main__.Share at 0x159e1329b80>,
     <__main__.Option at 0x159e1211610>,
     <__main__.Option at 0x159e0966520>,
     ...
     <__main__.Share at 0x159e1345cd0>,
     <__main__.Share at 0x159e1345670>,
     <__main__.Share at 0x159e1345250>,
     <__main__.Share at 0x159e13459d0>,
     <__main__.Share at 0x159e13104f0>,
     <__main__.Share at 0x159e13103a0>]
    """
    account = []
    for i in range(len(account_data.index)):
        transaction = account_data.iloc[i]
        security = transaction[2].split()[0]
        if security in ('CALL', 'PUT'):
            account.append(
                Option(
                    security=security,
                    transaction=transaction,
                    market=market
                    )
                )
        else:
            account.append(
                Share(transaction=transaction, market=market)
                )
    return account
```