# Overview

# Getting Started

# risk_dash



## Overview

risk_dash is a framework to help simplify the data flow for a portfolio of assets and handle market risk metrics at the asset and portfolio level. If you clone the source repository, included is a Dash application to be an example of some of the uses for the package. To run the Dash app, documentation is here

## Getting Started

### Installation

Since the package is in heavy development, to install the package fork or clone the repository and run `pip install -e risk_dash/` from the directory above your local repository.

To see if installation was sucessful run `python -c 'import risk_dash; print(*dir(risk_dash), sep="\n")'` in the command line, currently the output should match the following:

```
$ python -c 'import risk_dash; print(*dir(risk_dash), sep="\n")'
__builtins__
__cached__
__doc__
__file__
__loader__
__name__
__package__
__path__
__spec__
market_data
name
```

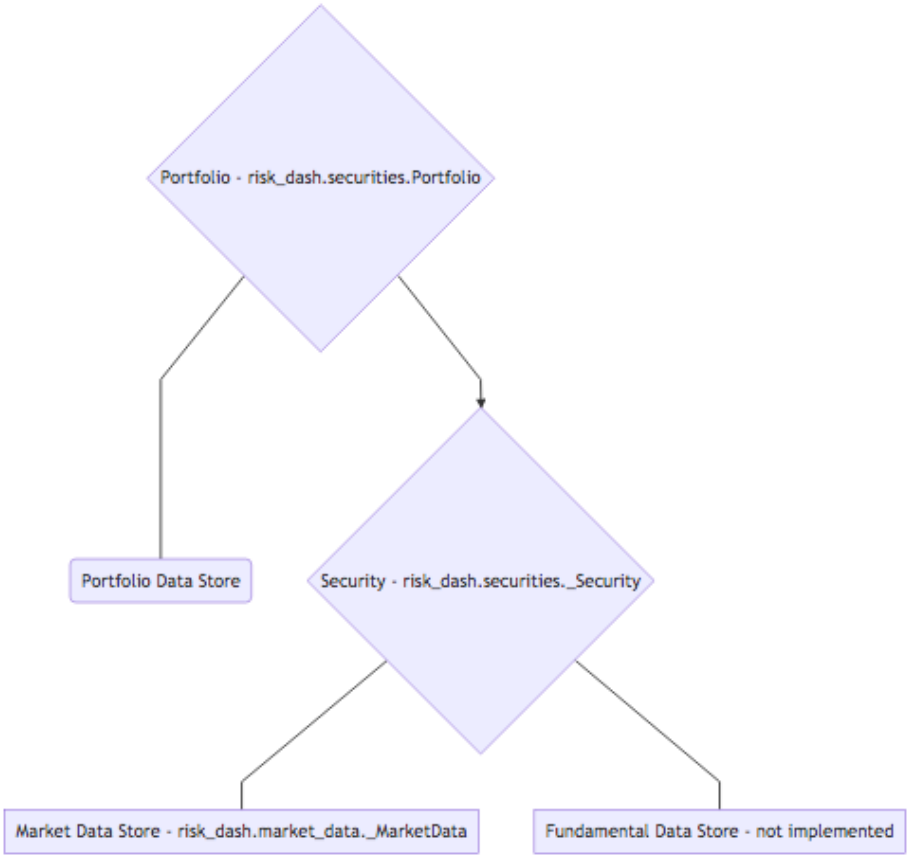
```
securities
simgen
```

### Getting Started

Now that we have the package installed, let's go through the object workflow to construct a simple long/short equity portfolio.

High level, we need to specify:

1. Portfolio Data
    - Security weights, what securities are in the portfolio
2. Security data
    - Identification data
    - Market data
3. Portfolio/security constructors to handle the above data




To do so, we'll need subclasses for the _Security and _MarketData classes

to model specific types of securities. Currently supported is the Equity class. Once we have the portfolio constructed, we will specify and caclulate parameters to simulate or look at historic distrubtions. We'll then create a subclass of _Simulation and _RandomGen

### Security data, _Security objects, and creating Security Subclasses

The core of the package is in the _Security and Portfolio objects. Portfolio objects are naturally a collection of Securities, however we want to specify the type of securities that are in the portfolio. Since we're focusing on a long/short equity portfolio we want to create a Equity subclass.

Subclasses of _Security classes must have the following methods:

- valuation(current_price)
- mark_to_market(current_price)
- get_marketdata()

In addition, we want to pass them the associated _MarketData object to represent the security's historic pricing data. To build the Equity class, we first want to inherit any methods from the _Security class:

```
class Equity(_Security):

    def __init__(self, ticker, market_data : md.QuandlStockData, ordered_price, quantity, da
        self.name = ticker
        self.market_data = market_data
        self.ordered_price = ordered_price
        self.quantity = quantity
        self.initial_value = ordered_price * quantity
        self.date_ordered = date_ordered
        self.type = 'Equity'
```

To break down the inputs, we want to keep in mind that the goal of this subclass of the _Security object is to provide an interface to model the Equity data.

- ticker is going to be the ticker code for the equity, such as 'AAPL'
- market_data is going to be a subclass of the _MarketData object
- ordered_price is going to be the price which the trade occured
- quantity for Equity will be the number of shares
- date_ordered should be the date the order was placed

Currently the implemented _MarketData subclass is QuandlStockData, which is a wrapper for this Quandl dataset api. This data is no longer being updated, for current market prices you must create a _MarketData subclass for your particular market data. Information to construct the subclass is below.

```
class Equity(_Security):
  # ...
```

```
  def valuation(self, current_price):
      value = (current_price - self.ordered_price) * self.quantity
      return(value)

class Equity(_Security):
  # ...
  def mark_to_market(self, current_price):
    self.marketvalue = self.quantity * current_price
    change = (current_price - self.ordered_price) * self.quantity
    return(change)
```

Since the Equity class is already implemented in the package, we can create an instance that represents an order of 50 shares at close on March 9th, 2018. The code to call the instance is as follows:

```
>>> import risk_dash.market_data as md
>>> import risk_dash.securities as sec
>>> from datetime import datetime
>>> apikey = 'valid-quandl-apikey'
>>> aapl_market_data = md.QuandlStockData(
  apikey = apikey,
  ticker = 'AAPL'
)
>>> aapl_stock = sec.Equity(
  ticker = 'AAPL',
  market_data = aapl_market_data,
  ordered_price = 179.98,
  quantity = 50,
  date_ordered = datetime(2018,3,9)
)
>>> aapl_stock.valuation(180.98) # $1 increase in value
50.0
```

> Note: Another important distinction is that the Equity class will only keep a reference to the underlying QuandlStockData, which will minimize dulpication of data. However, at scale, you'd want minimize price calls to your data source, so that could probably be better dealt at the Portfolio level.

### Portfolio Data and creating a Portfolio Object

To have an equity position in your portfolio you need what quantity you ordered, what price, and when you ordered the position. In this example, we'll use the following theoretical portfolio found in `portfolio_example.csv`:

| Type | Ticker | Ordered Price | Ordered Date | Quantity |
|---|---|---|---|---|
| Equity | AAPL | 179.98 | 3/9/18 | 50 |

| Type | Ticker | Ordered Price | Ordered Date | Quantity |
|---|---|---|---|---|
| Equity | AMD | 11.7 | 3/9/18 | 100 |
| Equity | INTC | 52.19 | 3/9/18 | -50 |
| Equity | GOOG | 1160.04 | 3/9/18 | 5 |

With this example, the portfolio is static, or just one snap shot of the weights at a given time, in practice your portfolio would be dynamically and have a time dimensionality. The Portfolio class could be easily adapted to handle that information to accurately plot historic performance by remarking through time, however, risk metrics looking forward would probably still only want to account for the current positions in the portfolio. Due to this, presently the package only looks at one snap shot in time.

With a portfolio so small, it is very easily stored in a csv and each security can store the reference to the underlying market data independently. As such, there is an included portfolio constructor in the portfolio class from csv

```
>>> current_portfolio = sec.Portfolio()
>>> port_dict = current_portfolio.construct_portfolio_csv(
  input='.portfolio_example.csv',
  apikey=apikey
)
>>>type(port_dict)
dict
```

The `Portfolio` object is main portfolio handler for the portfolio data, however, depending on how your data is structured you will want to create a portfolio constructor. If you have a list of securities you can also just pass the list into the Portfolio instance. The following code creates a portfolio of just the AAPL equity that we created earlier:

```
>>>aapl_portfolio = sec.Portfolio([aapl_stock])
```

## Building Custom Classes

**_MarketData**

**_Security**

**_RandomGen**

**_Simulation**

## Simulating Distributions

**** _Security(*name*, *market_data*, *kwargs*)**

Generic class for Security objects, shouldn't externally be used

**Parameters**

**name** : string identifier for security, i.e. 'AAPL'

**market_data** : MarketData object reference for the Market Data associated with the Security

****kwargs**** : Generic arguments

**Attributes**

| Attribute | Description |
|---|---|

**Portfolio(*securities=None*, *input=None*, *apikey=None*)**

Main handler for portfolio data. A Portfolio is a collection of Security objects in the `port` dictionary.

**Parameters**

**securities** : array-like of Security objects - should contain all securities in portoflio

**input** : either `pandas.core.frame.DataFrame` or string for csv input path

**apikey** : String for quandl apikey

**Attributes**

| Attribute | Description |
|---|---|
| port | dict of all securities in Portfolio |

**Methods**

| Method | Description |
|---|---|
| value() | calculate the current valuation of the portfolio |
| mark() | mark the portfolio at the current market price |
| get_date() | grab the max date available |
| set_portfolio_historicmarketdata() | combine individual security market_data into one pandas dataframe |
| set_port_variance() | set the portfolio variance using weights and covariance matrix |
| set_weights() | set the portfolio weights by invested value |
| construct_portfolio_from_csv() | create portfolio from a .csv matching format portfolio_example.csv |
| get_portfolio_returns() | return market_data object |

| Method | Description |
|---|---|
| get_weights() | return portfolio weights |
| get_port_variance() | return weighted portfolio |

**value()**

**mark()**

**get_date()**

**set_portfolio_marketdata()***