Welcome!

QUANTITATIVE RISK MANAGEMENT IN PYTHON

Dr. Jamsheed Shorish
Computational Economist

About Me

- Computational Economist
- Specializing in:
  - asset pricing
  - financial technologies ("FinTech")
  - computer applications to economics and finance
- Co-instructor, "Economic Analysis of the Digital Economy" at the ANU
- **Shorish Research** (Belgium): computational business applications

What is Quantitative Risk Management?

- **Quantitative Risk Management**: Study of *quantifiable uncertainty*
- **Uncertainty**:
  - Future outcomes are unknown
  - Outcomes impact planning decisions
- **Risk management**: mitigate (reduce effects of) adverse outcomes
- **Quantifiable uncertainty**: identify factors to measure risk
  - **Example**: Fire insurance. What factors make fire more likely?
- **This course**: focus upon risk associated with a *financial portfolio*

Risk management and the Global Financial Crisis

- Great Recession (2007 - 2010)
  - **Global** growth loss more than $2 trillion
  - **United States**: nearly $10 trillion lost in household wealth
  - U.S. stock markets lost c. $8 trillion in value
- Global Financial Crisis (2007-2009)
  - Large-scale changes in fundamental asset values
  - Massive uncertainty about future returns
  - High asset returns volatility
  - Risk management critical to success or failure

Quick recap: financial portfolios

- Financial portfolio
  - Collection of assets with uncertain future returns
  - Stocks
  - Bonds
  - Foreign exchange holdings ('forex')
  - Stock options
- Challenge: quantify risk to manage uncertainty
  - Make optimal investment decisions
  - Maximize portfolio return, conditional on risk appetite

Quantifying return

- Portfolio return: weighted sum of individual asset returns
  - `Pandas` data analysis library
  - DataFrame `prices`
  - `.pct_change()` method
  - `.dot()` method of `returns`

```
prices = pandas.read_csv("portfolio.csv")
returns = prices.pct_change()
weights = (weight_1, weight_2, ...)
portfolio_returns = returns.dot(weights)
```

Quantifying risk

- Portfolio return volatility = **risk**
- Calculate volatility via **covariance matrix**
- Use `.cov()` DataFrame method of `returns` and annualize

```
covariance = returns.cov()*252

print(covariance)
```

	Asset 1	Asset 2	Asset 3	Asset 4
Asset 1	1.010823	0.406477	0.503497	0.573644
Asset 2	0.406477	0.373898	0.308224	0.472868
Asset 3	0.503497	0.308224	0.480904	0.398519
Asset 4	0.573644	0.472868	0.398519	0.917529

Quantifying risk

- Portfolio return volatility = **risk**
- Calculate volatility via **covariance matrix**
- Use `.cov()` DataFrame method of `returns` and annualize
- *Diagonal* of `covariance` is individual asset variances

```
covariance = returns.cov()*252
print(covariance)
```

	Asset 1	Asset 2	Asset 3	Asset 4
Asset 1	1.01082	0.406477	0.503497	0.573644
Asset 2	0.406477	0.373898	0.308224	0.472868
Asset 3	0.503497	0.308224	0.480904	0.398519
Asset 4	0.573644	0.472868	0.398519	0.917529

Quantifying risk

- Portfolio return volatility = **risk**
- Calculate volatility via **covariance matrix**
- Use `.cov()` DataFrame method of `returns` and annualize
- *Diagonal* of `covariance` is individual asset variances
- *Off-diagonals* of `covariance` are covariances between assets

```
covariance = returns.cov()*252
print(covariance)
```

	Asset 1	Asset 2	Asset 3	Asset 4
Asset 1	1.01082	0.406477	0.503497	0.573644
Asset 2	0.406477	0.373898	0.308224	0.472868
Asset 3	0.503497	0.308224	0.480904	0.398519
Asset 4	0.573644	0.472868	0.398519	0.917529

Portfolio risk

- Depends upon asset `weights` in portfolio
- Portfolio variance σ_p^2 is

$$\sigma_p^2 := w^T \cdot \mathrm{Cov}_p \cdot w$$

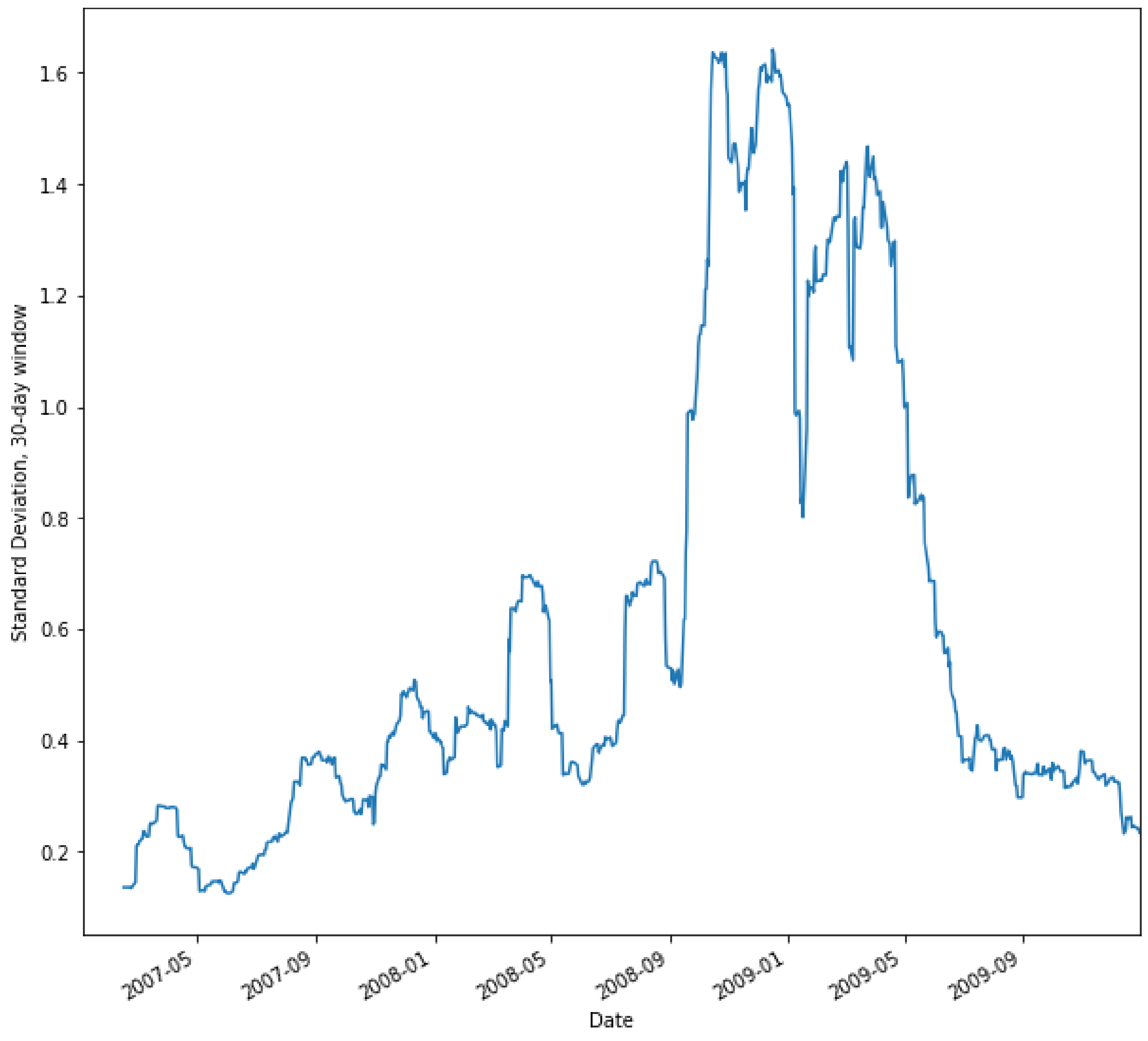
- Matrix multiplication can be computed using `@` operator in Python
- Standard deviation is usually used instead of variance

```
weights = [0.25, 0.25, 0.25, 0.25] # Assumes four assets in portfolio

portfolio_variance = np.transpose(weights) @ covariance @ weights

portfolio_volatility = np.sqrt(portfolio_variance)
```

Volatility time series

- Can also calculate portfolio volatility over time
- Use a 'window' to compute volatility over a fixed time period (e.g. week, 30-day 'month')
- `Series.rolling()` creates a window
- Observe volatility **trend** and possible extreme events

```
windowed = portfolio_returns.rolling(30)
volatility = windowed.std()*np.sqrt(252)
volatility.plot()
    .set_ylabel("Standard Deviation...")
```



Let's practice!

QUANTITATIVE RISK MANAGEMENT IN PYTHON

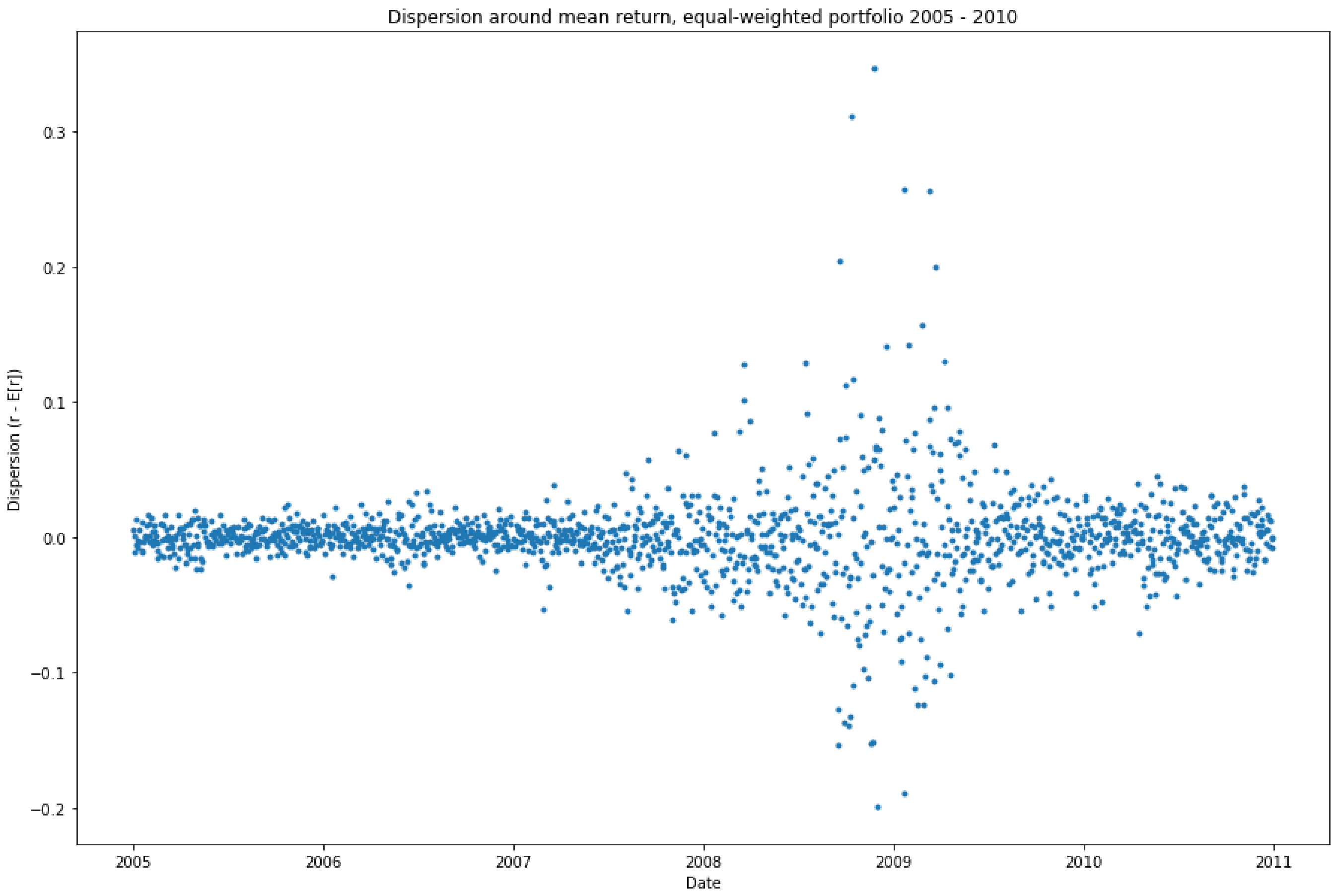
Risk factors and the financial crisis

QUANTITATIVE RISK MANAGEMENT IN PYTHON

Jamsheed Shorish
Computational Economist

Risk factors

- Volatility: measure of **dispersion** of returns around expected value
- Time series: expected value = sample average
- What drives expectation and dispersion?
- **Risk factors**: variables or events driving portfolio return and volatility



Risk exposure

- **Risk exposure**: measure of possible portfolio loss
  - Risk factors determine risk exposure
- **Example**: Flood Insurance
  - *Deductible*: out-of-pocket payment regardless of loss
  - 100% coverage still leaves deductible to be paid
  - So deductible is *risk exposure*
  - Frequent flooding => more volatile flood outcome
  - Frequent flooding => higher risk exposure

Systematic risk

- **Systematic risk**: risk factor(s) affecting volatility of all portfolio assets
  - **Market risk**: systematic risk from general financial market movements
- **Airplane engine failure**: systematic risk!
- Examples of financial systematic risk factors:
  - Price level changes, i.e. *inflation*
  - Interest rate changes
  - Economic climate changes

Idiosyncratic risk

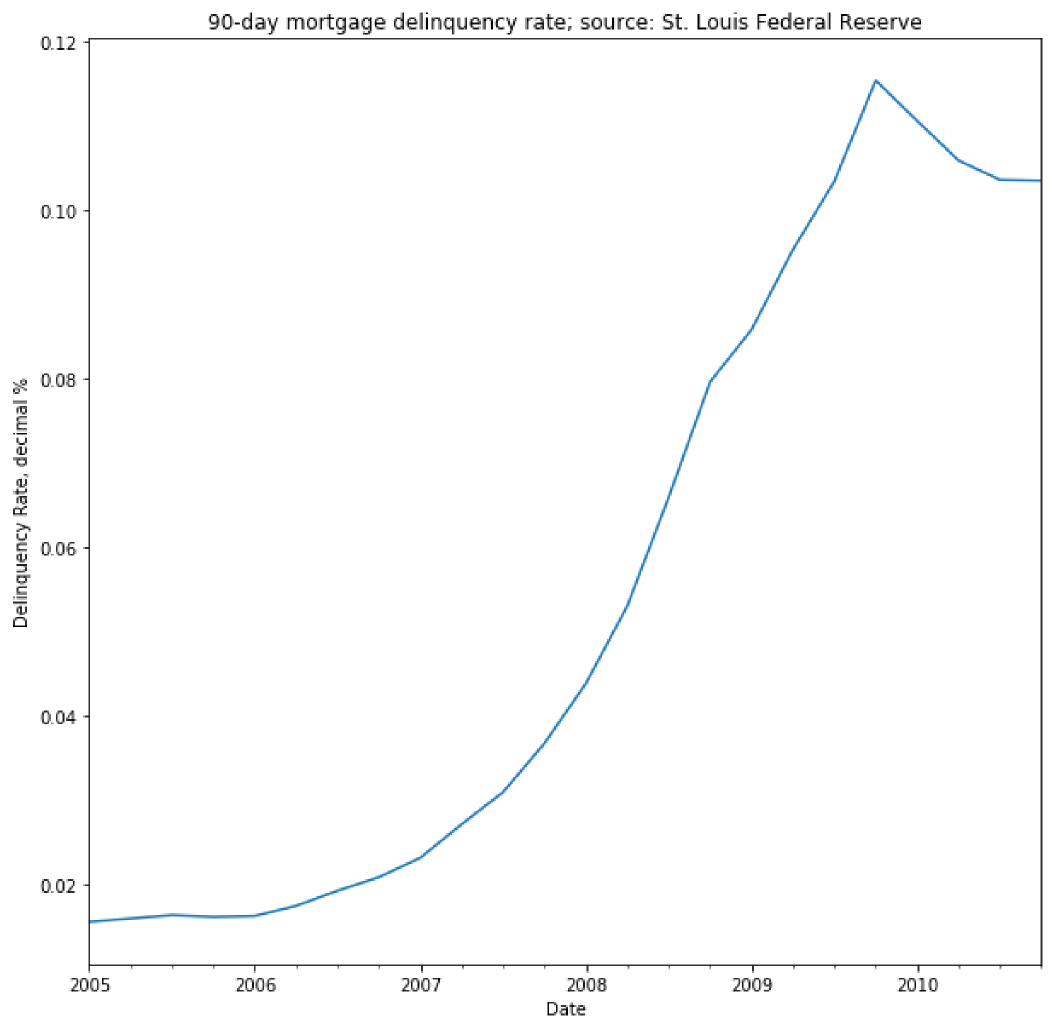
- **Idiosyncratic risk**: risk specific to a particular asset/asset class.
- **Turbulence and the unfastened seatbelt**: idiosyncratic risk!
- Examples of idiosyncratic risk:
  - Bond portfolio: issuer risk of default
  - Firm/sector characteristics
    - Firm size (market capitalization)
    - Book-to-market ratio
    - Sector shocks

Factor models

- **Factor model**: assessment of risk factors affecting portfolio return
- Statistical regression, e.g. Ordinary Least Squares (OLS):
  - dependent variable: returns (or volatility)
  - independent variable(s): systemic and/or idiosyncratic risk factors
- **Fama-French** factor model: combination of
  - market risk and
  - idiosyncratic risk (firm size, firm value)

Crisis risk factor: mortgage-backed securities

- Investment banks: borrowed heavily just before the crisis
- Collateral: mortgage-backed securities (MBS)
- MBS: *supposed* to diversify risk by holding many mortgages of different characteristics
    - Flaw: mortgage default risk in fact was **highly correlated**
    - Avalanche of delinquencies/default destroyed collateral value
- **90-day mortgage delinquency**: risk factor for investment bank portfolio during the crisis



Crisis factor model

- Factor model regression: portfolio returns vs. mortgage delinquency
- Import `statsmodels.api` library for regression tools
- Fit regression using `.OLS()` object and its `.fit()` method
- Display results using regression's `.summary()` method

```
import statsmodels.api as sm

regression = sm.OLS(returns, delinquencies).fit()

print(regression.summary())
```

Regression .summary() results

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.190
Model:                            OLS   Adj. R-squared:                  0.154
Method:                 Least Squares   F-statistic:                     5.174
Date:                Tue, 31 Dec 2019   Prob (F-statistic):             0.0330
Time:                        08:13:21   Log-Likelihood:                 60.015
No. Observations:                  24   AIC:                            -116.0
Df Residuals:                      22   BIC:                            -113.7
Df Model:                           1
Covariance Type:            nonrobust
============================================================================================
                                coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------------------
const                         0.0100      0.007      1.339      0.194      -0.006       0.026
Mortgage Delinquency Rate     0.2558      0.112      2.275      0.033       0.023       0.489
==============================================================================
Omnibus:                       19.324   Durbin-Watson:                   0.517
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               23.053
Skew:                           1.814   Prob(JB):                     9.87e-06
Kurtosis:                       6.145   Cond. No.                         26.7
==============================================================================
```

Let's practice!

QUANTITATIVE RISK MANAGEMENT IN PYTHON

Modern portfolio theory

QUANTITATIVE RISK MANAGEMENT IN PYTHON

Jamsheed Shorish
Computational Economist

The risk-return trade-off

- Risk factors: sources of uncertainty affecting return
- Intuitively: **greater uncertainty** (more risk) compensated by **greater return**
- Cannot *guarantee return*: need some measure of **expected return**
  - average (mean) historical return: proxy for expected future return

Investor risk appetite

- Investor survey: *minimum* return required for given level of risk?
- Survey response creates (risk, return) risk profile "data point"
- Vary risk level => **set** of (risk, return) points
- Investor **risk appetite**: defines one quantified relationship between risk and return

Choosing portfolio weights

- Vary **portfolio weights** of *given* portfolio => creates set of (risk, return) pairs
- Changing weights = beginning risk management!
- **Goal**: change weights to maximize expected return, *given* risk level
    - Equivalently: minimize risk, *given* expected return level
- Changing weights = adjusting investor's *risk exposure*

Modern portfolio theory

- **Efficient portfolio**: portfolio with weights generating highest expected return for given level of risk
- **Modern Portfolio Theory (MPT)**, 1952
  - H. M. Markowitz (Nobel Laureate 1990)
- Efficient portfolio weight vector $w^\star$ solves:

$$\max_w \mathbb{E}[w^T r]$$

with

$$w^T \Sigma w = \bar{\sigma}^2$$

The efficient frontier

- Compute many efficient portfolios for different levels of risk
- **Efficient frontier**: locus of (risk, return) pairs created by efficient portfolios
- `PyPortfolioOpt` library: optimized tools for MPT
    - `EfficientFrontier` class: generates one optimal portfolio at a time
    - **Constrained Line Algorithm** (`CLA`) class: generates the entire efficient frontier
        - Requires covariance matrix of returns
        - Requires proxy for expected future returns: mean historical returns

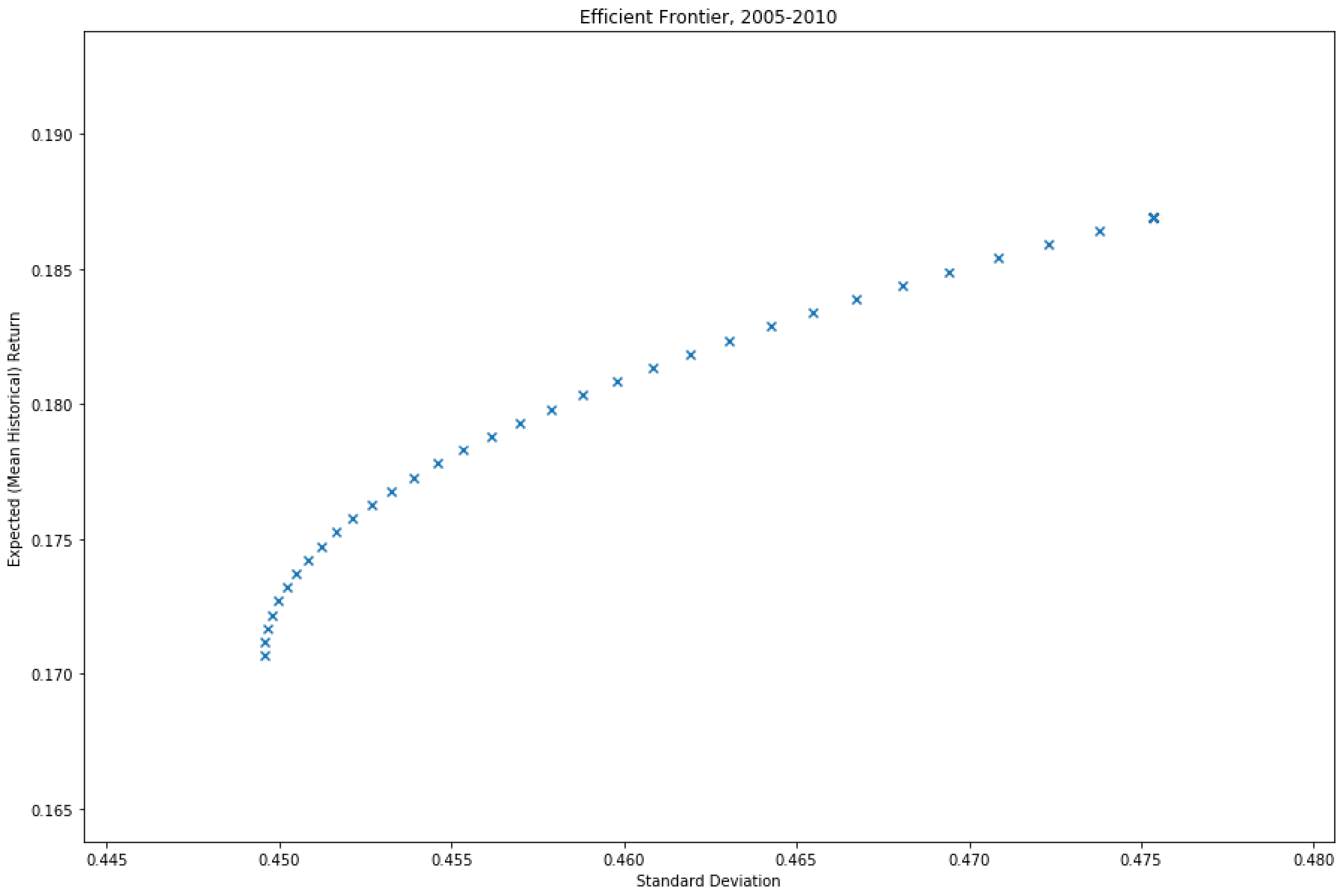
Investment bank portfolio 2005 - 2010

- **Expected returns**: historical data
- **Covariance matrix:** `Covariance Shrinkage` improves efficiency of estimate
- **Constrained Line Algorithm** object `CLA`
- **Minimum variance portfolio:** `cla.min_volatility()`
- **Efficient frontier**: `cla.efficient_frontier()`

```
expected_returns = mean_historical_return(prices)
efficient_cov = CovarianceShrinkage(prices).ledoit_wolf()
cla = CLA(expected_returns, efficient_cov)
minimum_variance = cla.min_volatility()
(ret, vol, weights) = cla.efficient_frontier()
```

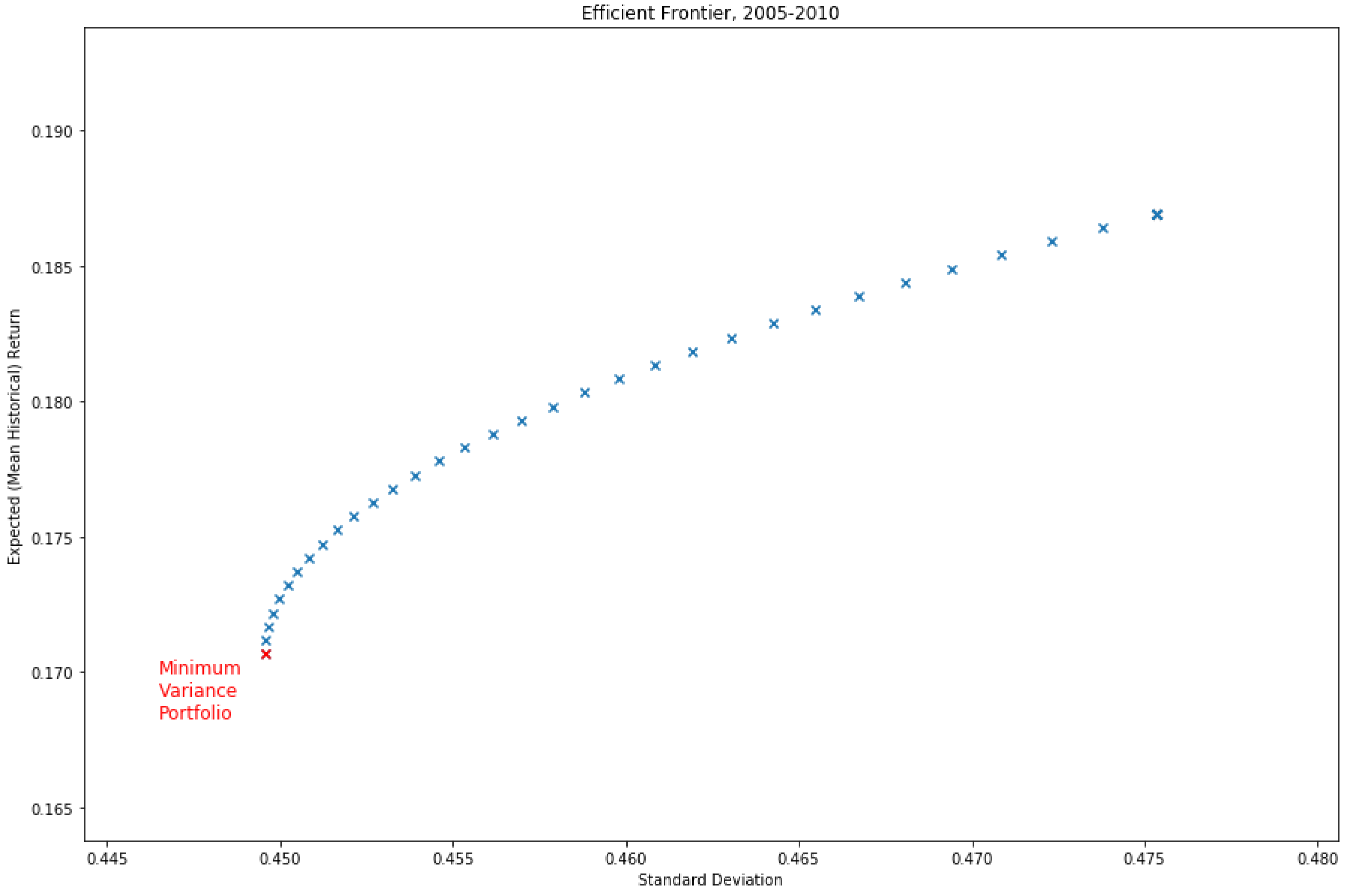
Visualizing the efficient frontier

- Scatter plot of (vol, ret) pairs



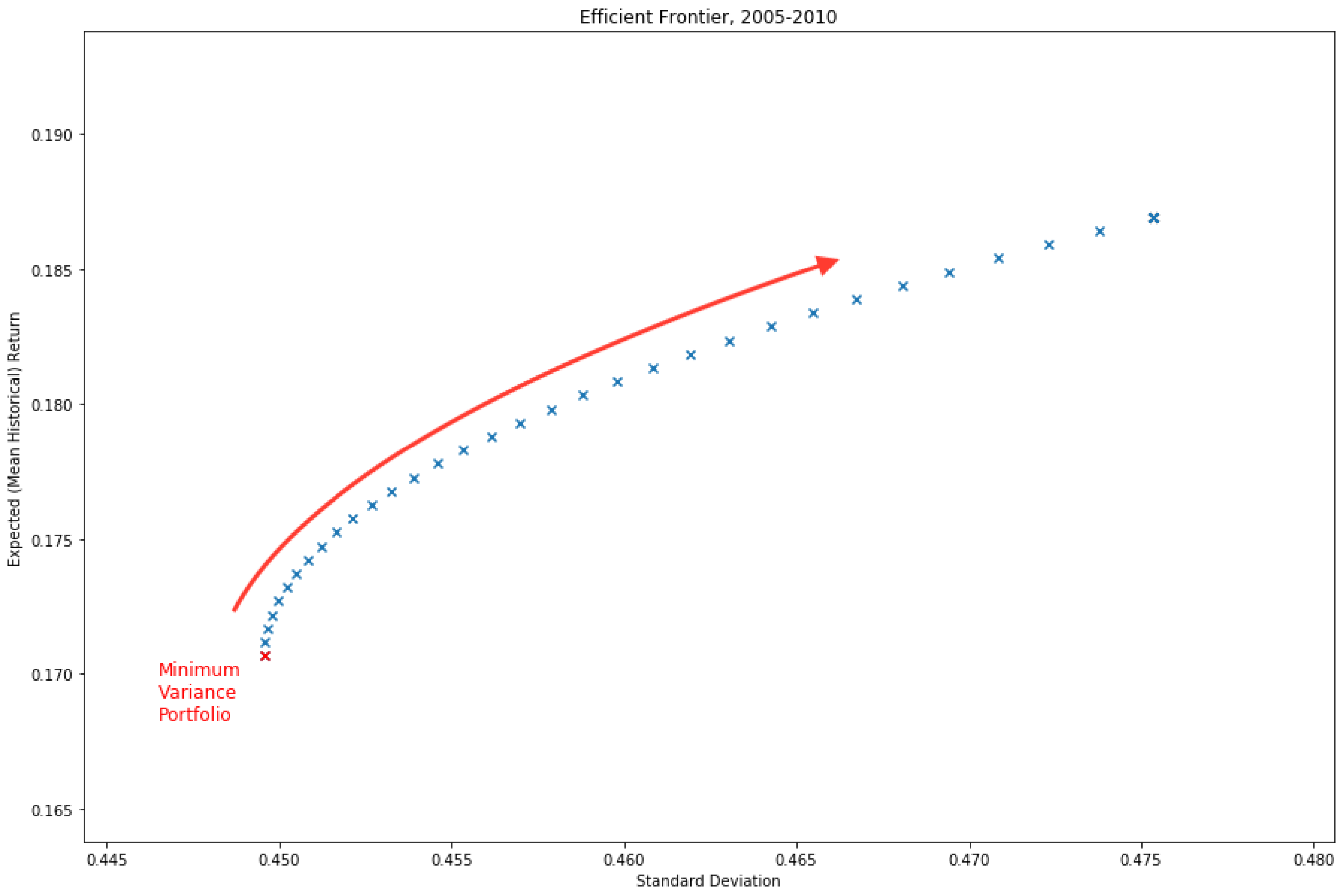
Visualizing the efficient frontier

- Scatter plot of (vol, ret) pairs
- **Minimum variance portfolio**: smallest volatility of all possible efficient portfolios



Visualizing the efficient frontier

- Scatter plot of (vol, ret) pairs
- **Minimum variance portfolio**: smallest volatility of all possible efficient portfolios
- Increasing risk appetite: move *along* the frontier



Let's practice!

QUANTITATIVE RISK MANAGEMENT IN PYTHON