FINANÇAS QUANTITATIVAS

ANÁLISE DE RISCO E OTIMIZAÇÃO DE PORTFÓLIO COM PYTHON

FERRAMENTAS DE GERENCIAMENTO DE RISCO E OTIMIZAÇÃO PORTFÓIOS COM PYTHON

ARTHUR RUFINO DE CAMARGO

Sumário

Qual a proposta desse livro?

O principal objetivo desse livro é que você compreenda como uma alocação pode fazer diferença em uma carteira em questão de risco e rentabilidade.

Apresentarei ferramentas para medir o risco individual de cada ativo e assim pavimentar o caminho para que você escolha, de acordo com seu apetite ao risco, os ativos mais interessantes para sua carteira e depois disso montar seu portfólio dentro de uma estratégia que faça sentido para o nível de risco/retorno desejado.

Obviamente não posso deixar de citar a linguagem Python que será nossa principal ferramenta nessa jornada: excelente para os cálculos necessários deixando qualquer planilha de acompanhamento de investimentos para trás em termos de facilidade e escalabilidade.

Introdução

Uma Breve Introdução à História do Risco

O risco e como mensurá-lo são questões que acompanham a humanidade há muito tempo. Nem sempre recorremos a métodos científicos. O oráculo de *Delphos* é um dos episódios da História que remete à consulta a deuses através de rituais e transes místicos para saber o desfecho de batalhas, decisões políticas ou mesmo da vida cotidiana dos poderosos da época. Com o avanço da matemática e do cálculo probabilístico permitiu-se que já na época das navegações as primeiras apólices de seguro fossem emitidas em caso da carga ser roubada durante a expedição e demorou um pouco para humanidade aproveitar os benefícios de quantificar o risco do lado dos ganhos e não somente pelo lado das perdas.

A definição de risco está incorporada em diversas áreas: seguros, indústria, mercado financeiro ou saúde. Todas essas áreas trabalham de alguma forma com risco e tentam de alguma maneira quantificá-lo.

No chinês, o ideograma que representa o risco é formado por dois termos: "perigo e oportunidade". Essa definição se aproxima bastante do que conhecemos, mesmo que intuitivamente, sobre risco no mercado financeiro.

Já dentro do mercado financeiro a palavra risco comumente vem acompanhada também da palavra retorno, seja nas notícias ou nas cartas de gestores de grandes fundos de investimento. Sob a ótica das finanças risco é definido como a variabilidade dos retornos observados de um investimento em comparação com seus retornos esperados.

Isso implica em duas questões: primeiro ter um método adequado para quantificar essa "variabilidade dos retornos observados" e segundo ter outro método também para estimar os retornos futuros.

Esse livro como objetivo apresentar técnicas quantitativas para uma melhor tomada de decisão na aquisição de ações e como compor uma carteira que possua uma estratégia orientada para gestão da relação risco e retorno.

Risco

Risco e mercado de ações

Data de 1909 a *Financial Review of Reviews*, uma publicação que examinava carteiras de ação, desvio-padrão e estimativa de retornos de ativos nos Estados Unidos e quase na mesma época em 1915 as agências *Moody's* e *Standard & Poor's* (que na época chamava *Stdanard Statistics Bureau*), processavam informações contábeis para oferecer notas as empresas e consequentemente influenciar seus preços e também seu crédito frente ao mercado.

Nessa época acreditavam que o valor do ativo era o valor presente dos dividendos a serem pagos por ele.

Em 1934 Benjamim Graham relatava em seus estudos que a volatilidade baseada no histórico dos preços de nada adiantava como proxy para o risco, pois as quedas nos preços dos ativos podem ser passageiras e não representar o verdadeiro valor do ativo (*Value Investing*), evidenciando dessa maneira uma oportunidade de compra independente da volatilidade histórica atual do seus retornos. Essa tese é incorporada até hoje por investidores como Warren Buffet nos Estados Unidos e Luis Paes de Barros aqui no Brasil.

Fronteira Eficiente

Em 1952, Harry Markowitz mudou completamente o panorama de portfólio de investimento dando um passo à frente em relação a diversificação de ativos dentro de uma carteira. O conceito atual da época de que o valor de uma ação era o valor presente de seus dividendos esperados, era da autoria de John Burr William no seu livro *Theroy of Investment Value*.

Markowitz argumentou que se o valor presente de uma ação era o dividendo que estava por vir, o investidor deveria alocar todo seu dinheiro na ação que tivesse o maior retorno esperado, o que empiricamente ia contra a ideia de diversificar os ativos. Indo além na ideia de carteira, advogava que o risco da carteira deveria ser menor que o risco individual de cada ativo, caso contrário a diversificação não estava sendo eficiente. Matematicamente ele afirmava que a variância dos retornos de uma carteira poderia ser descrita como função não apenas dos pesos de cada ativo e da variância dos ativos individualmente, mas também da correlação entre os ativos.

Dessa maneira, ao estabelecer essa relação, Markowitz consegue afirmar um conceito que já era posto em prática, entretanto não parou por aí: formulou um método onde os investidores poderiam otimizar a diversificação em seu portfólio de ações.

Ora, se os investidores podem escolher os ativos de sua carteira, os pesos de cada ativo em seu portfólio são as alavancas que estão sob seu controle, já que por mais que sejam feitas estimativas, os retornos dependem de diversos fatores que muitas vezes não estão ao alcance dos investidores

Primeiramente iremos analisar o comportamento de um ativo separadamente, para então, posteriormente fazermos uma análise em conjunto. Para isso, iremos importar as seguintes bibliotecas:

```
#Python Gestão de Portfolio

import pandas as pd
import numpy as np
from pandas_datareader import data as wb
import matplotlib.pyplot as plt
```

- **Pandas** servirá para manipularmos os data frames referentes aos preços históricos das ações analisadas.
- **Numpy** nos ajudará com a parte de estatística e alguns outros cálculos algébricos.
- **Pandas DataReader** é a biblioteca que importa os dados para nós através de uma API ligada diretamente ao site do *Yahoo Finance*, outras fontes também são suportadas pela biblioteca, recomendo que você invista um pouco do seu tempo dando uma olhada na documentação.
- **MatplotLib** para podermos visualizar os dados

Para fins de análise escolheremos a holding ITAUSA (ITSA4). Os parâmetros do DataReader são: primeiro o nome do ativo entre aspas simples, a fonte que será utilizada (data_source) e por opção start onde você pode definir um limite, caso não restrinja o range de data, a função irá trazer todos os preços da série histórica disponível.

Na sequência já pedimos para ver o cabeçalho do *dataset* importado:

```
18 #Risco Individual do Ativo ITSA.4
19 itau_prices = wb.DataReader('ITSA4.SA', data_source = 'yahoo', start='2017-1-1')
20 itau_prices.head()
```

Temos como resultado dos códigos a seguinte saída:

```
Out[2]:
                High      Low     Open    Close      Volume  Adj Close
Date
2017-01-02  7.45070  7.36942  7.40554  7.41457   4589559.0   5.960370
2017-01-03  7.73970  7.45070  7.45070  7.73970  18882423.0   6.221735
2017-01-04  7.79388  7.66745  7.68551  7.78485   9621033.0   6.258029
2017-01-05  7.93838  7.73066  7.80291  7.85710  17324814.0   6.316109
2017-01-06  7.90226  7.76679  7.85710  7.83001  12229671.0   6.294332
```

Estamos interessados apenas na ***última*** coluna do *dataset* que é [Adj Close], pois essa nos oferece dados ajustados conforme o pagamento de dividendos. Para isso iremos eliminar as demais colunas (High, Low, Open, Close e Volume), e renomear para ITAU:

```
22 #renomeia a coluna adj close
23 itau_prices.rename(columns = {"Adj Close":"ITAU"}, inplace = True)
24
25 #remove as colunas
26 itau = itau_prices.drop(itau_prices.columns[[0,1,2,3,4]], axis =1)
```

Retornos

Para calcular os retornos da série histórica de preços da ITSA4, precisamos entender a fórmula do retorno linear que nada mais é que (Preço Final – Preço inicial)/Preço Final, ou simplesmente podemos utilizar o método pct_change para conseguir as variações percentuais.

```
40 #variação retornos diarios
41 itau_returns = itau.pct_change()
```

Com o método .head( ) conseguimos o resultado abaixo:

```
In [12]: itau_returns.head()
Out[12]:
                ITAU
Date
2019-01-02       NaN
2019-01-03  0.008696
2019-01-04 -0.003135
2019-01-07 -0.000786
2019-01-08  0.013375
```

Aqui vale a pena abrirmos um parêntese para explicar por que utilizamos pct_change ao invés do método logarítmico para obtenção dos retornos. Muitos códigos e artigos acadêmicos vemos os retornos sendo calculados através do famoso "log-return", que são os retornos logarítmicos dos preços, esses representam melhor os retornos acumulados que o retorno linear.

```
35 #logretornos diarios
36 itau_log_returns = np.log(itau/itau.shift(1))
```

Como estamos trabalhando com retornos diários, a diferença é mínima:

```
In [14]: itau_log_returns.head()
Out[14]:
                ITAU
Date
2019-01-02       NaN
2019-01-03  0.008658
2019-01-04 -0.003139
2019-01-07 -0.000787
2019-01-08  0.013287
```

Qual usar? Fica a seu critério decidir qual melhor caminho seguir. A primeira parte do quebra-cabeça está concluída, a seguir veremos como calcular a volatilidade dos retornos.

Como funciona na prática?

Os retornos são os principais pilares de um bom investimento, buscar ativos que tenham maiores retornos deve ser o objetivo de todo investidor que pensa em construir uma carteira que tenha bom desempenho no longo prazo, entretanto como vamos ver a seguir, não é a única métrica que importa.

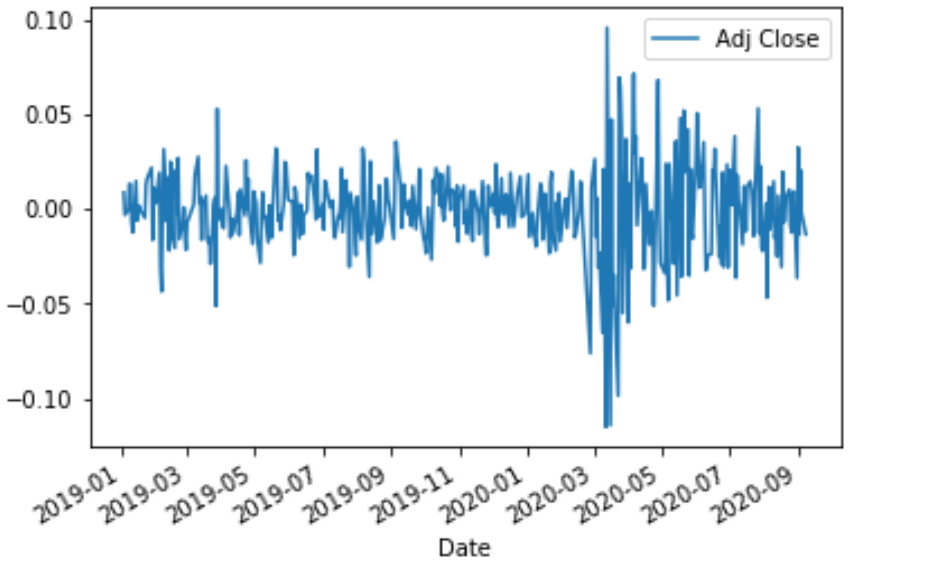
Visualizando os retornos

Iremos plotar também o gráfico dos retornos para observar o seu comportamento:

```
43 #plota o grafico de retornos
44 itau_log_returns.plot()
```

Para o gráfico simples, conseguimos observar a abrangência dos retornos (positivos e negativos):

```
Out[7]: <matplotlib.axes._subplots.AxesSubplot at 0x101a0b70>
```

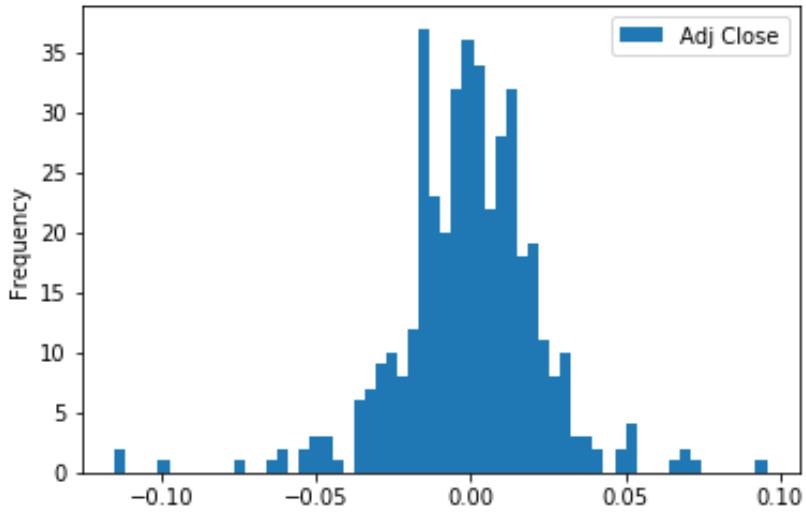


Além do gráfico de linhas, podemos plotar os retornos através de sua distribuição de frequência em um histograma, isso nos permite observar onde estão concentrados as maiores incidências de retornos:

```
56 #distribuição dos retornos
57 itau_log_returns.plot.hist(bins = 60)
58 plt.show()
```

Observe que a maior parte das incidências de retorno estão + 5% e – 5%

```
In [8]: itau_log_returns.plot.hist(bins = 60)
   ...: plt.show()
   ...:
```



Fatores de Risco

Como já fora descrito anteriormente temos que o risco de um ativo é representado pela volatilidade dos retornos apresentados ao longo do tempo. A melhor forma de calcular essa volatilidade é através do desvio-padrão.

$$\sigma = \sqrt{\frac{\sum (x_i - \mu)^2}{N}}$$

σ = population standard deviation

N = the size of the population

x_i = each value from the population

μ = the population mean

O desvio-padrão é uma medida de dispersão, ou seja, nos auxilia a enxergar quão longe da média estão os valores da amostra. No mercado financeiro utilizamos a medida do desvio padrão dos retornos como um indicador do quão volátil é aquele ativo, ou seja, quão disperso são os retornos daquele ativo positiva ou negativamente. Um desvio padrão alto dos retornos quer dizer que temos retornos cada vez mais longe da média.

Na fórmula temos a raiz do somatório entre a diferença das observações e da média elevado ao quadrado, dividido pelo tamanho da amostra.

Importante ressaltar que no mercado todo ativo está exposto a basicamente **dois tipos** de risco:

Risco sistemático – afeta os ativos do portfólio da mesma maneira (não na mesma intensidade), exemplo disso é a economia em si. Quando temos um cenário econômico ruim, isso reflete em diferentes intensidades nos ativos, entretanto dificilmente algum ativo escapa ileso da desvalorização.

Risco intrínseco – afeta especificamente um ativo (risco relacionado ao negócio). Aumento da matéria-prima relativa ao negócio (aumento de custos), queda do valor de mercado do produto que a empresa comercializa (queda do preço do minério afetando a Vale), são exemplos de riscos que podem afetar unicamente o desempenho dos retornos dos ativos.

Calculando o risco individual de cada ativo através do desvio padrão

Como já temos calculado a variação diária dos retornos, para calcular o desvio padrão em Python utilizaremos a biblioteca Numpy com a função std (*standard deviation*)

```
63 #volatilidade diária
64 volatilidade_diaria = np.std(itau_returns['ITAU'])
```

"Printando" a volatilidade diária temos:

```
In [5]: print(volatilidade_diaria)
0.023102188948326222
```

Para calcularmos em bases anuais, basta multiplicarmos pela raiz (252 dias úteis para trade):

```
69 #volatilidade anual
70 volatilidade_anual = volatilidade_diaria*np.sqrt(252)
```

Teremos então o resultado abaixo:

```
In [7]: print(volatilidade_anual)
0.366735880190976
```

Existe uma limitação importante no método de calcular risco de um ativo através do desvio padrão: a fórmula leva em consideração tanto os retornos positivos quanto negativos, atribuindo para os dois o mesmo peso. Quando falamos de investimento temos a premissa que o investidor é avesso a perdas, ou seja, ninguém gosta de ver seu dinheiro indo pelo ralo. Portanto na próxima seção abordaremos uma maneira mais ponderada de observar os retornos a partir de uma perspectiva de perdas.

Risco de Perda Potencial ou *Downside Risk*

Downside Risk é um termômetro da volatilidade relacionada às perdas do ativo. Obtendo o desvio padrão das perdas podemos entender como funciona o risco do lado negativo dos retornos desse ativo e mais uma vez a premissa da alta volatilidade implica em maiores riscos. Temos dois objetivos aqui nesse tópico: descobrir o *drawndown* do investimento, ou seja, baseado em seu histórico, qual foi a maior queda ocorrida no período e calcular a volatilidade dos retornos negativos.

Como funciona na prática?

Ativos que possuem alto desvio-padrão tendem a ser mais voláteis, ou seja, seus retornos variam bastante em relação a média observada, tanto para cima como para baixo. Um alto desvio-padrão nem sempre indica que você pode apenas perder dinheiro naquele ativo por ele ser arriscado, mas que você pode tanto perder quanto ganhar dinheiro com ele, pois o desvio-padrão tradicional não faz distinção de *loss* e *gain*.

Semivariância

A semivariância é uma alternativa ao desvio padrão tradicional, que calcula a dispersão entre os valores que estão abaixo da média da amostra.

$$\sigma_{semi} = \sqrt{\frac{1}{N} \sum_{x_t \leq \bar{x}} (x_t - \bar{x})^2}$$

N = number of entries which fall below the mean.

No caso dos ativos, se formos considerar a volatilidade apenas dos itens abaixo da média, podemos ainda estar considerando retornos positivos, ou ainda deixando alguns retornos negativos de fora da amostra, para isso fazemos uma adaptação, iremos calcular o desvio padrão dos retornos que estão abaixo de zero, ou seja, aqueles retornos que representam as perdas diárias:

```
77 #semivariância
78 semivariancia = np.std(itau_returns['ITAU']<0)
```

Como resultado temos um número bem maior do que a alternativa do desvio padrão:

```
In [10]: print(semivariancia)
0.49985551721644134
```

Como funciona na prática?

O downside risk é uma medida de dispersão dos valores negativos da amostra, ou seja, ele nos mostra como está a dispersão entre os valores de *loss*. Um alto downside risk nos informa que as perdas podem variar bastante, posteriormente é interessante avaliar qual a distribuição e amplitude dessas perdas com outros indicadores como o *value-at-risk*.

Drawdown

Drawdown pode ser expresso pela máxima queda que os retornos acumulados tiveram em um determinado período, podendo ser analisado como a queda máxima da semana, do mês, do ano ou de um período específico.

Para calcularmos o *drawdown* iremos primeiramente:

1) Calcular os retornos acumulados em base 100:

```
97 retorno_acumulado = itau_log_returns*100
```

2) Retirar os NaNs e substituí-los por zero:

```
100 retorno_acumulado = retorno_acumulado['ITAU'].fillna(0)
```

3) Calcular o ponto máximo dos acumulados utilizando o Numpy e garantir que não fique abaixo de 1:

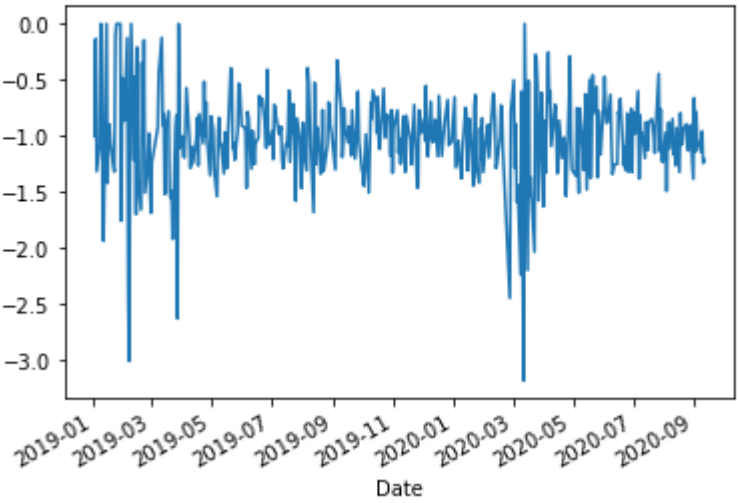
```
102 ponto_maximo = np.maximum.accumulate(retorno_acumulado)
103 ponto_maximo[ponto_maximo<1] = 1
```

4) Calcular o drawdown

```
105 drawdown = (retorno_acumulado)/running_max - 1
```

5) Plotar para ver o resultado:

```
In [13]: drawdown.plot()
Out[13]: <matplotlib.axes._subplots.AxesSubplot at 0x1236d5f8>
```



O Modelo CAPM e o Beta

O modelo CAPM (Capital Asset Pricing Model) foi desenvolvido em 1961 por Bill Sharpe e John Lintner e Jack Treynor, e pretende estabelecer uma relação entre o ativo analisado e sua *sensibilidade* ao risco sistemático de mercado. O modelo tem o prêmio sobre o risco como seu principal pilar, somamos a taxa livre de risco com o produto entre beta o prêmio pelo risco.

O **beta** serve como um índice de sensibilidade do ativo ao mercado. A taxa livre de risco expressa a taxa de remuneração do ativo que o investidor tem certeza que receberá sua remuneração na data e na quantia prometida e o prêmio pelo risco expressa quanto o investidor está disposto a pagar "a mais" por esse ativo tendo em vista que possui a alternativa livre de risco para investir, ou seja, o ágio sobre a taxa livre de risco.

As premissas do modelo são bastante simples: o modelo considera a existência de um mercado de capitais eficiente, em que os investidores buscam a diversificação máxima em sua carteira, a existência da taxa de juros livre de risco, não existem custos transacionais (taxas de compra ou venda de ativos), não existem impostos e que não existe assimetria de informação, ou seja, um investidor tem mais informações do que os demais dentro do mercado.

Observemos a fórmula do CAPM:

$$Re = Rf + \beta \times (Rf-Rm)$$

Re = Retorno estimado pelo modelo;

Rf = Taxa livre de risco;

β = Coeficiente beta de risco;

Rm = Risco médio de mercado

Beta

Nessa seção iremos focar bastante no beta por ser um indicador muito utilizado dentro do mercado financeiro e fácil de interpretar e expressa a relação entre os retornos do ativo e o risco de mercado.

Observe abaixo a fórmula do beta:

β = Covariância do ativo com a carteira de mercado/ Variância da carteira de mercado

Primeiro passo é obter a covariância entre os retornos do ativo com os retornos do que chamamos de carteira de mercado. Essa carteira segundo a teoria deve envolver todos os ativos do mercado em uma cesta. O mais próximo que possuímos disso hoje no Brasil é o Ibovespa, com suas principais de limitações de pouco número de empresas e alta concentração de capital em determinados setores. A covariância é uma medida que expressa quanto do ativo se move quando a carteira de mercado se move também. Como segundo passo calculamos a variância dos retornos do Ibovespa.

Ao dividirmos a covariância dos retornos de um ativo com o mercado e dividindo pela variância da carteira de mercado obtemos um número ao redor de 1, onde betas acima desse valor são investimentos considerados mais arriscados que o mercado e o oposto também é verdadeiro, betas com valores inferiores a 1 são menos arriscados.

Outra forma de calcular o beta é através da regressão linear entre os retornos do ativo e o retorno do mercado, onde o beta nada mais é que a inclinação da reta de regressão. O método de regressão linear busca estabelecer relação entre duas variáveis a dependente ou resposta (no caso os retornos do ativo analisado) e a independente ou preditora (os retornos do Ibovespa).

No exemplo abaixo iremos utilizar o Ibovespa como referência de carteira de mercado e para calcular o beta iremos utilizar os seguintes passos:

1) Importar o Ibovespa e dar o mesmo tratamento de dados anterior que demos a ITAUSA:

```
123 #Importando o Ibovespa
124 ibov = wb.DataReader('^IBVSP', data_source = 'yahoo', start='2017-1-1')
125 ibov.rename(columns = {"Adj Close":"IBOV"}, inplace = True)
126 ibov = ibov.drop(ibov.columns[[0,1,2,3,4]], axis =1)
127 ibov_returns = np.log(ibov/ibov.shift(1))
```

2) Calcular a variância do ibov

```
#Variancia Ibov"
ibov_var = ibov_returns.var()
```

3) Juntar os dataframes

```
#Juntar ITAUSA e IBOV"
from functools import reduce
join_itau_ibov = ([itau_log_returns,ibov_returns])
cov_itau = reduce(lambda left, right: pd.merge(left, right, on=['Date'], how = 'inner'), join_itau_ibov)
```

4) Montar a matriz de covariância [ITSA,IBOV] e encontrar o coeficiente

```
#Montar a matriz de covariância
cov_itau = cov_itau[['ITAU','IBOV']].cov()
cov_itau_coef = cov_itau.iloc[0,1]
```

5) Calcular o beta dividindo o coeficiente de covariância dividindo pela variância do Ibov

```
#Calcular o Beta de 2019 a 2020YTD
beta_itau = cov_itau_coef/ibov_var
print(beta_itau)
```

Calculando o beta através da regressão linear

A regressão linear das taxas de retorno de uma ação (ITSA4) e dos retornos do índice de mercado (IBOV) é uma abordagem alternativa ao quociente entre covariância e variância. O beta é calculado a partir da inclinação da reta de regressão. Retas mais positivamente inclinadas indicam betas maiores, ou seja, mais sensíveis as variações do mercado.

1) Importar a biblioteca statsmodels

```
import statsmodels.api as sm
```

2) Definir a variável beta no método OLS (*Ordinary Least Squares*)

```
65 beta = sm.OLS(itau_retornos, ibov_retornos).fit()
```

3) Chamar o método summary para observar os resultados

```
67 beta.summary()
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                   ITAU   R-squared (uncentered):                   0.696
Model:                            OLS   Adj. R-squared (uncentered):              0.695
Method:                 Least Squares   F-statistic:                              1008.
Date:                Fri, 09 Oct 2020   Prob (F-statistic):                   7.46e-116
Time:                        23:39:37   Log-Likelihood:                          1303.5
No. Observations:                 441   AIC:                                     -2605.
Df Residuals:                     440   BIC:                                     -2601.
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
IBOV           0.8413      0.027     31.743      0.000       0.789       0.893
==============================================================================
Omnibus:                       27.400   Durbin-Watson:                   1.957
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               91.584
Skew:                          -0.088   Prob(JB):                     1.30e-20
Kurtosis:                       5.226   Cond. No.                         1.00
==============================================================================
```

A resposta será a tabela acima e o beta do nosso ativo é o coeficiente da regressão 0,8413. Na tabela temos também o R^2 que nos diz quanto das variações do retorno do Ibovespa explicam as variações nos retornos da ITAUSA, nesse caso 69,6%.

Como funciona na prática?

O beta indica como seu ativo se comporta de acordo com o mercado. Na prática, betas acima de 1 são empresas mais sucetíveis as variações do mercado, tanto para cima como para baixo, do contrário temos betas inferiores a 1 que demonstram empresas que tem menos influência do mercado.

No caso da interpretação do Beta da ITAUSA temos que para cada 1% de variação do Ibovespa temos uma variação de 0,8414% no

retorno das ações dentro do período analisado. Buscar o equilíbrio dentro de sua carteira através dos betas faz sentido quando você está buscando certa proteção ou alavancagem a favor do mercado.

Value At Risk

O value at risk é uma medida que mostra para o investidor qual o potencial de perda daquele ativo ou de sua carteira como um todo. O indicador mede o pior cenário que esse ativo ou carteira pode atingir, dadas as condições normais de mercado e um determinado nível de confiança.

Uma carteira de investimentos com VaR de R$5 mil em um mês à um intervalo de significância de 95% nos indica que existe 5% de chance dessa carteira perder mais que R$5 mil no período de trinta dias, dentro de condições normais de mercado.

O VaR pode ser utilizado para um ativo isoladamente, uma carteira de ativos ou até para análise de fluxos de caixa de uma empresa. O conceito de VaR está intimamente ligado a quantificar uma probabilidade de perda se as coisas ocorrerem como sempre ocorreram.

Quando fatores de risco extrapolam tudo o que já vimos antes, o VaR cai por terra e as perdas podem ser severas. E o mercado continua e continuará nos surpreendendo sempre:

Ibovespa cai 30% em março, maior queda mensal em 22 anos

Mês termina com seis circuit breakers em apenas oito pregões, petróleo afundando 60%, epidemia de coronavírus virando pandemia, e EUA, com contração prevista em 30% no próximo trimestre, se tornando o novo epicentro da doença

[1]

O VaR é normalmente medido em três intervalos de confiança: 90%, 95% e 99% e o período analisado pode ser anual, mensal, semana ou diário.

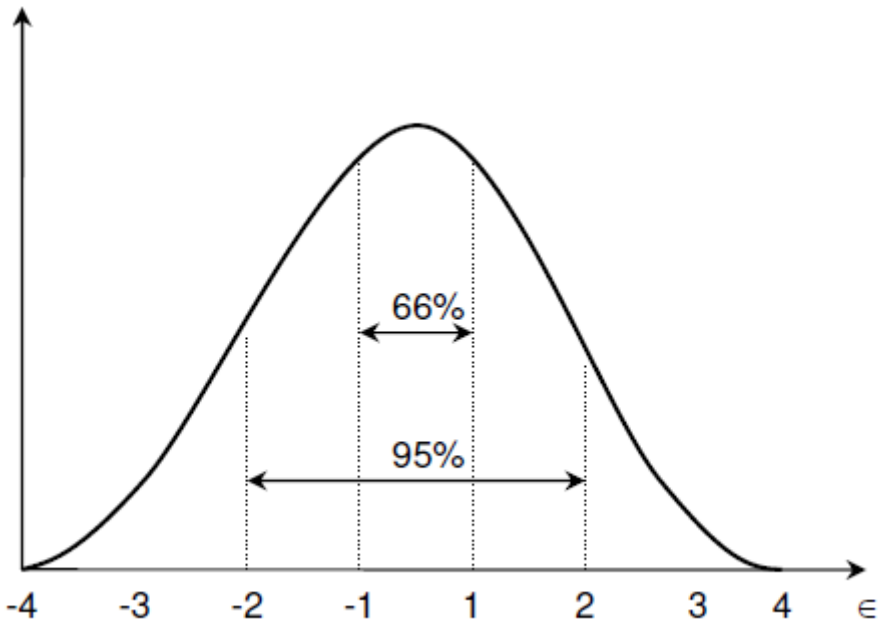
Distribuição Normal

Para compreender melhor as bases do VaR precisamos entender as particularidades de uma distribuição normal.

A partir do teorema do limite central Laplace comprovou que muitos tipos de populações conforme o número de amostras ia aumentando a média gradativamente convergia para uma distribuição que tinha característica de uma normal com média 0 e desvio-padrão = 1.

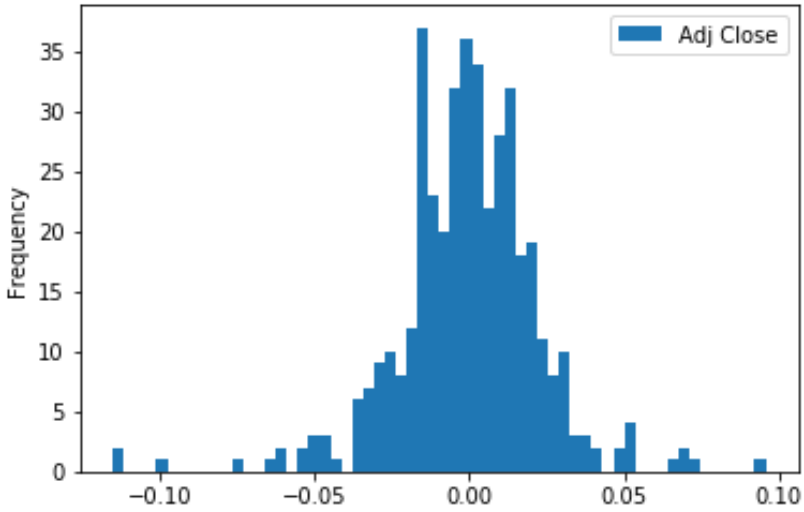
A distribuição normal é utilizada para estimar probabilidades em diversos eventos como seguros, risco de default em crédito, fluxos de caixa e no mercado financeiro os retornos de ações.

Em termos simplificados, temos um gráfico de frequência das observações com essa característica:



Aqui nesse livro nós já vimos algo parecido, lembram?

```
In [8]: itau_log_returns.plot.hist(bins = 60)
   ...: plt.show()
   ...:
```



A distribuição dos retornos das ações da ITAUSA4. Similar de certa forma, certo?

Pois bem. Nesse caso, na regra da distribuição normal temos que 95% das observações estão em um espaço de -2 e + 2 desvio-padrão e 66% está entre -1 e +1. Dessa forma podemos estimar probabilidades de ocorrência de resultados, tanto positivo quanto negativos se a distribuição dos retornos das ações tiver comportamento normal.

Vamos analisar duas abordagens para o VaR: Simulação histórica, Variância-Covariância e paramétrico.

VaR Histórico

Para o VaR da metodologia de simulação histórica pressupõe que tenhamos a série de retornos já calculada, variável que já está armazenada em itau_retornos. Adicionalmente iremos declarar denominando o intervalo de confiança para calcular os percentis.

VaR_90 – 90% de confiança

VaR_95 – 95% de confiança

VaR_99 – 99% de confiança

Iremos então selecionar o percentil de cada retorno correspondente ao VaR e nível de confiança desejado, conforme abaixo:

```
88 #VaR simulação historica
89
90 var_90 = np.percentile(itau_retornos, 10)
91 var_95 = np.percentile(itau_retornos, 5)
92 var_99 = np.percentile(itau_retornos, 1)
```

Cada uma das variáveis var armazena a probabilidade de uma queda nos retornos a um determinado nível de confiança.

VaR Paramétrico

Como o próprio nome já sugere, essa metodologia necessita de parâmetros para ser estimada. A função *norm.ppf()* do Scipy nos apresenta um número gerado de forma aleatória dentro de uma função de probabilidade normalizada. São executadas simulações dentro dos parâmetros que nós passamos de média de retornos e volatilidade. No caso do parâmetro relacionado aos retornos

podemos aplicar algumas outras técnicas para dar peso maior às ultimas observações ou mesmo você pode inserir o retorno estimado a partir de alguma outra técnica (iremos abordar retornos estimados mais a frente).

1) Primeiro iremos importar a função norm do pacote Scipy

```
70 from scipy.stats import norm
```

2) Depois calcular as médias dos retornos através da variável media:

```
72 #declarar variavel media
73 media = np.mean(itau_retornos['ITAU'])
```

3) Definir as variáveis dos intervalos de confiança:

```
80 #calcular o VaR
81 VaR_90 = norm.ppf(conf_90, media, vol_itau)
82 VaR_95 = norm.ppf(conf_95, media, vol_itau)
83 VaR_99 = norm.ppf(conf_99, media, vol_itau)
```

Pronto. Cada variável tem um resultado armazenado correspondente a uma perda máxima percentual dentro do intervalo de confiança parametrizado. conta

Como funciona na prática?

VaR_95% gerou um resultado de -2%? Quer dizer que existe uma chance de 5% (1-95%) de ocorrer uma perde de -2% ou mais no dia nas ações do ITAUSA4.

O VaR é utilizado por bancos e fundos de investimento para saber qual o custo de carrego das posições tomadas em ativos, pois sabendo dentro de um intervalo de confiança quanto posso perder,

consigo gerenciar meu risco com um colchão de liquidez para cobrir eventuais perdas.

VaR Simulação de Monte Carlo

A simulação de Monte Carlo aplicada ao VaR é uma mistura das ferramentas contidas no VaR paramétrico com o histórico. O método da simulação de Monte Carlo surgiu em 1946 com Stanislaw Ulam a partir de um problema de probabilidades e análise combinatória. O modelo faz simulações dentro de uma distribuição de probabilidades dos retornos que detêm média = mu e desvio-padrão = desv_pad e a partir disso podemos calcular o percentil que a perda está inserida, assim como fizemos como VaR histórico.

Iremos ver como fica a simulação de Monte Carlo agora em Python:

1) Definir o dataset retornos_simulados em branco para abrigar as iterações da simulação

```
204 retornos_simulados = []
```

2) Criar a variável dias_trade para 252 dias pois estamos analisando retornos diários

```
207 dias_trade = 252
```

3) Criar um loop de 1000 iterações para a Simulação de Monte Carlo

```
210 for i in range (1000):
211     sim_retornos = np.random.normal(media, vol_itau, dias_trade)
```

4) Anexar as iterações no df retornos_simulados

```
213 retornos_simulados.append(sim_retornos)
```

5) Calcular o percentil relacionado ao VaR desejado

```
215 var_99 = np.percentile(retornos_simulados,1)
```


Novamente a distribuição normal...

Como vimos anteriormente a premissa sobre a distribuição normal aparece diversas vezes durante a aplicação do VaR. Quando os retornos fogem a regra da distribuição normal ou apresentam as famosas caudas gordas, com possibilidades de perdas ou ganhos altos, a premissa da distribuição normal pode nos levar a subestimar as perdas potenciais, o que dentro da gestão de risco se torna um grande problema.

Dados históricos

Observe que tanto no VaR Paramétrico como na Simulação de Monte Carlo utilizamos a média histórica dos retornos e o desvio-padrão histórico, no VaR Simulação Histórica o passado acompanha inclusive o nome da ferramenta, portanto todos os métodos aqui dispostos compartilham da mesma fraqueza: histórico não é certeza de futuro. Caso o período analisado for de estabilidade, possíveis mudanças estruturais na série de dados ao longo do tempo podem se tornar um desafio à acurácia do VaR.

Parâmetros não-estacionários

Outra vez o VaR Paramétrico e o VaR Simulação de Monte Carlo compartilham mais uma fraqueza. Quando realizamos a simulação ou definimos um parâmetro dentro de suas fórmulas estamos admitindo que por exemplo o desvio-padrão utilizado naquela

análise é constate, o que pode por muitas vezes não ser verdade. Você acredita que a volatilidade na semana do "Joesley Day" (dia em que os irmãos Batista delataram esquema de corrupção que supostamente envolvia o presidente em exercício Michel Temer), foi a mesma do que a volatilidade do mês anterior ao fato? A média de retornos muda, a volatilidade se altera devido a diversos fatores que podem ser internos à empresa como também externos.

Diversos trabalhos sugerem adaptações em relação aos parâmetros utilizados no cálculo do VaR, tanto em relação às médias dos retornos sendo substituídas por médias móveis

Otimização de portfólios

Particularidades de um portfólio

Um portfólio é caracterizado por um conjunto de ativos. Nessa seção trataremos exclusivamente de um portfólio de ações de empresas listadas no IBOVESPA, que já possuem histórico de pelo menos quatro anos de cotações para que possamos construir análises mais robustas. Nada impede você de escolher empresas que abriram capital recentemente, mas alguns ajustes deverão ser feitos, pois com pouco histórico não será possível utilizar os parâmetros relacionados a média de retornos ou desvio-padrão histórico.

Objetivos de um portfólio

Muito além da diversificação em si, um portfólio deve ter alguns pilares que fundamentam sua estratégia dentro do prazo definido por você investidor e administrador de sua carteira.

Acredito ser particularmente importante definir *stop-loss* dentro de suas posições, sejam compradas ou vendidas, mesmo com intuito de investimento de longo prazo. *Stop* nada mais é que um limite que você admite de perda dentro dessa posição. Como você já aprendeu anteriormente, poderia utilizar como uma base o VaR como perda máxima admitida dentro da posição assumida dada um cenário de probabilidades.

Objetivos de rentabilidade também são necessários. Podem ser mensais, trimestrais, mensais ou anuais, quem define a temporalidade aqui é quem administra a carteira e quão confortável você se sente com isso. Tenha sempre em mente que: 30 dias são um mês, 12 meses são um ano, 10 anos são uma década. São os tijolos

que constroem o muro, independente do seu tamanho, ter uma consistência nos resultados traz um melhor desempenho no longo prazo. Observe os fundos vencedores que já gozam de um certo tempo de maturidade no mercado, o que eles possuem em comum? Constância. Ganhar Sempre. Nunca perder dinheiro.

A otimização de portfólio

A otimização de média-variância que foi primeiramente proposta por Harry Markowitz e já abordada aqui nesse livro no capítulo X é um problema de otimização com algumas variáveis já conhecidas por nós:

(Wp) = Wb + x

Onde:

- Wp = pesos do portfólio de benchmark (sempre somando 1);

- Wb = vetor de ações em posições compradas ou vendidas;

- x = pesos das ações no portfólio.

Para otimizar esse portfólio temos como objetivo minimizar a função como ocorre abaixo:

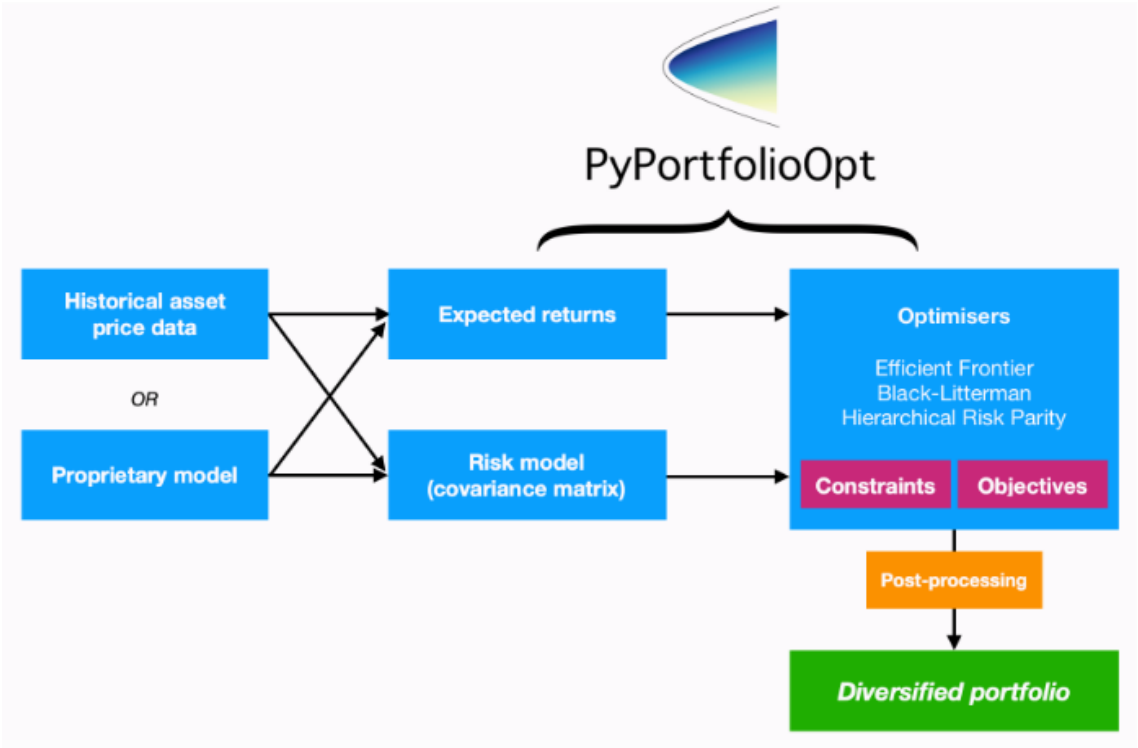
$$\begin{array}{ll} \underset{x}{\text{minimize}} & x^T \Sigma x \\ \text{subject to} & x^T \alpha \geq g \\ & x^T \mathbf{1} = 0 \\ & x \geq -w_B \\ & x \leq c\mathbf{1} - w_B \end{array}$$

Portanto, os retornos devem ser maiores que o retorno alvo da carteira xT α ≥g, os pesos da carteira devem somar 1

Otimizando portfólio com a biblioteca PyPortfólio

A biblioteca utilizada para otimização de portfólio será a PyPortfolioOpt [2], existem outras soluções, entretanto trabalham muito bem com os dados do S&P500 ou Nyse, para dados do Ibovespa a PyPortfolioOpt é mais indicada por uma gama maior de parâmetros customizáveis

Conforme podemos observar na figura abaixo temos um pipeline para construção de um modelo de otimização de carteira que segue alguns passos até chegar de fato na otimização do seu portfólio:



Primeiro passo precisamos de uma matriz de preços históricos que servirão como base para criar os retornos esperados. A biblioteca inclui alguns métodos para calcular retornos esperados, entretanto esse é um tema bastante controverso dentro das finanças quantitativas. Prever retorno não é tarefa fácil e existe uma ampla discussão sobre os melhores métodos a serem utilizados para isso. A biblioteca permite tanto que você use um modelo proprietário

(somente passando um vetor de previsões estimadas por você) ou que utilize algum dos métodos de estimar retorno do pacote.

O segundo momento é construir a matriz de covariância ou modelo de risco, como é chamado na biblioteca, a partir dos retornos sejam históricos ou do seu modelo proprietário. Nem todos os tipos de otimizadores necessitam de um modelo risco.

A terceira etapa é selecionar o método de otimização desejado e a partir disso ou adicionar alguma restrição ou objetivo desejado. O modelo permite ainda que você faça algumas alterações depois de processada a otimização para uma melhor calibração dos pesos da carteira. Veremos melhor na prática cada situação a partir dos exemplos.

Seleção de Ações

Iremos utilizar para o exemplo as seguintes ações:

- ITUB4

- ABEV3

- PETR4

- VALE3

Em Python seguimos os passos:

1) Criar uma lista com os tickers das ações (da mesma maneira que está escrito no Yahoo Finance):

```
20  #Seleção de Ativos para Carteira
21
22  ativos = ['ITSA4.SA', 'PETR4.SA', 'ABEV3.SA', 'VALE3.SA']
```

2) Criar um dataframe vazio para abrigar os dados que vamos coletar:

```
26  df = pd.DataFrame()
```

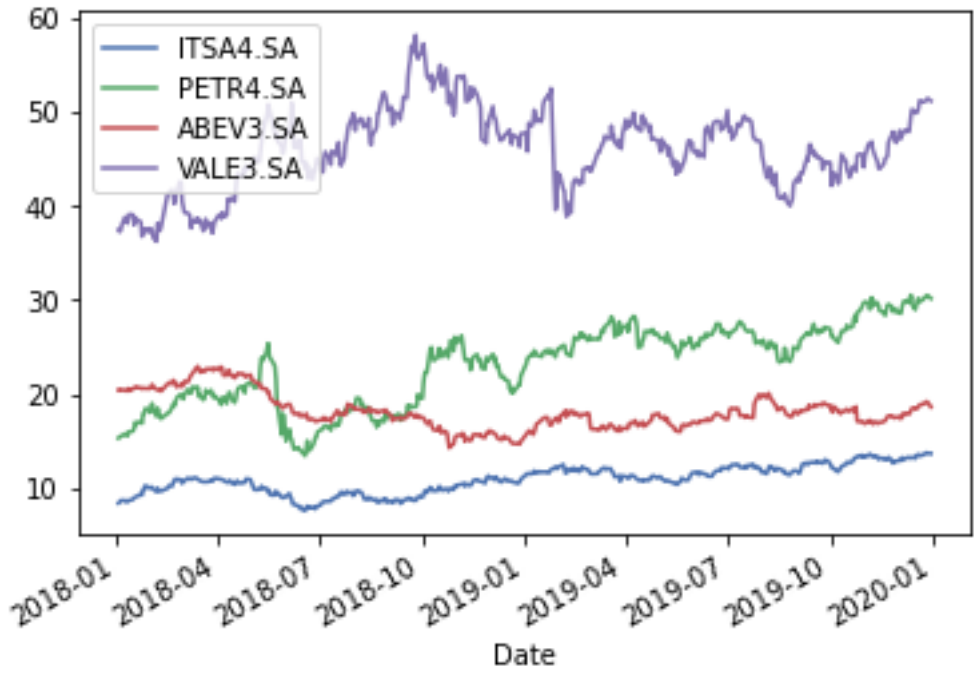
3) Fazer um loop para preencher o df com os dados das ações com o campo 'Adj Close'

```
#Fazer um loop para popular o df com os ativos e campos desejados
 for t in ativos:
   df[t] = wb.DataReader(t, data_source='yahoo', start='2018-01-01',end= '2020-01-01') ['Adj Close']
```

A partir das ações escolhidas iremos analisar o comportamento dos preços entre o período Janeiro-2018 e Janeiro-2020 graficamente

```
#Visualizar os preços
df.plot()
```

Como resultado temos:



Se o intuito foi entender o comportamento dos preços ao longo do período o gráfico acima é satisfatório. Mostra como os preços evoluíram, se há ou não alguma tendência e podemos ver também em cada ação as quedas e altas mais significativas.

Da mesma maneira que fizemos com a ITAUSA4 iremos calcular os retornos diários das ações do dataframe df:

```
#calcular retornos diários
retorno = df.pct_change()
```

Entretanto para comparar desempenho entre as ações devemos utilizar o crescimento composto que coloca os retornos em base 100

Retornos Esperados

Como input de todos os métodos de retorno esperado que iremos analisar agora, utilizaremos como base o df, dataframe do Pandas que contém os preços "crus" e que não foram convertidos em retornos ainda, outros tipos não serão aceitos como parâmetro. Por definição todos os métodos irão entregar valores anualizados apesar de receberem como inputs preços diários.

Iremos separar o df em dois anos (250 dias de pregão) para podermos avaliar qual melhor método de previsão de retornos que utilizaremos para a série de dados do ITAUSA4.

```
#Retornos Esperados
#Separando o dataframe
df_fut, df_past = df.iloc[:-250], df.iloc[-250:]
```

Dessa maneira poderemos utilizar o df_past para prever os valores e o df_fut para comparar os valores reais com os preditos para depois comparar qual o método mais efetivo. Para isso utilizaremos o erro médio absoluto:

$$\mathrm{MAE} = \frac{\sum_{i=1}^{n} |y_i - x_i|}{n}$$

MAE = mean absolute error

y_i = prediction

x_i = true value

n = total number of data points

O erro médio absoluto calcula a média da diferença entre o valor real e o valor estimado em módulo, dessa forma evita-se os resultados negativos.

Retorno Médio Histórico - Mean Historical Return - pypfopt.expected_returns.mean_historical_return(*prices, returns_data=False, compounding=True, frequency=252*)

Calcula a média aritmética de retornos anualizada

Parâmetros:

Prices – Pandas dataframe com os fechamentos ajustados de cada dia de cotação das ações;

Returns_data – padrão é falso, se deixar como true indica que você está passando retornos ao invés de preços;

Compounding – se você quer retornos compostos marque essa opção como True

Frequency – número de negociações no ano (padrão é 252 dias)

Em Python:

```
#Retorno Historico
future_rets_rh = expected_returns.mean_historical_return(df_fut)
mu_rh = expected_returns.return_model(df_past, method="mean_historical_return")
mean_abs_errors_rh = []
mean_abs_errors_rh.append(np.sum(np.abs(mu_rh-future_rets))/ len(mu_rh))
```

Média Móvel Exponencial - EMA Historical Returns - pypfopt.expected_returns.ema_historical_return(*prices, returns_data=False, compounding=True, span=500, frequency=252*)

Calcula a média exponencial ponderada, ou seja, a média dos últimos n preços com a ponderação sendo reduzida exponencialmente a cada evento, dando pesos maiores aos eventos mais recentes, seguindo a fórmula:

Média Móvel Exponencial = [Preço de Fechamento – Média Móvel Exponencial t+1] x (2/n+1) + Média Móvel Exponencial t-1

Parâmetros

Prices - Pandas dataframe com os fechamentos ajustados de cada dia de cotação das ações;

Returns_data - padrão é falso, se deixar como True indica que você está passando retornos ao invés de preços;

Compounding - se você quer retornos compostos marque essa opção como True

Frequency – número de negociações no ano (padrão é 252 dias)

Span – intervalo de tempo da EMA default é 500 períodos.

```
#EMA
future_rets_ema = expected_returns.ema_historical_return(df_fut)
mu_ema = expected_returns.return_model(df_past, method="ema_historical_return")
mean_abs_errors_ema = []
mean_abs_errors_ema.append(np.sum(np.abs(mu_ema-future_rets))/ len(mu_ema))
```

Capital Asset Pricing Model (CAPM) - expected_returns.capm_return(*prices, market_prices=None, returns_data=False, risk_free_rate=0.02, compounding=True, frequency=252*)

O Modelo CAPM foi abordado nas seções anteriores e parte do pressuposto da covariância de um ativo ou carteira de ativos com a carteira de mercado divido pela variância da carteira de mercado, gerando o beta que é um numero ao redor de 1, onde investimentos mais arriscados possuem beta > 1 e o contrário também é verdadeiro. Nesse método os retornos esperados são baseados na fórmula do CAPM:

$$Ri=Rf+\beta i(E(Rm)-Rf)$$

Parâmetros

prices - Pandas dataframe com os fechamentos ajustados de cada dia de cotação das ações;

market_prices – Pandas dataframe dos preços ajustados de fechamento do benchmarking de mercado (Ibovespa)

returns_data - padrão é falso, se deixar como true indica que você está passando retornos ao invés de preços;

risk_free_rate – taxa livre de risco default é 0.02. Se estiver utilizando retornos diários a taxa livre de risco deverá ser diária e assim sucessivamente para cada periodicidade que for analisar;

compounding - se você quer retornos compostos marque essa opção como True

frequency – número de negociações no ano (padrão é 252 dias)

```
#CAPM
future_rets_capm = expected_returns.capm_return(df_fut)
mu_capm = expected_returns.return_model(df_past, method="capm_return")
mean_abs_errors_capm = []
mean_abs_errors_capm.append(np.sum(np.abs(mu_capm-future_rets))/ len(mu_capm))
```

Comparando os métodos

Através do erro médio absoluto podemos identificar quais foram os métodos que melhor performaram no dataset de teste e performance. Intuitivamente podemos escolher o método que obteve melhor resultado no teste de MAE, entretanto isso não garante que seu modelo terá uma performance futura à exemplo do passado.

Caso queira trabalhar com um dataset próprio de retornos basta passar um pandas Series que contenha os tickers e também os valores estimados por você através de algum outro método desejado como o *valuation*.

Demais métodos

Outros métodos da classe expected_returns do pacote PyPortfolio:

Retornos - expected_returns.returns_from_prices(prices, log_returns=False)

Calcula os retornos a partir de uma série de preços

Parâmetros:

Prices - Pandas dataframe com os fechamentos ajustados de cada dia de cotação das ações;

Log_returns – True or False (default) para ativar log-retornos

Log-Retornos - expected_returns.log_returns_from_prices(prices)

Calcula os log-retornos de uma série de preços

Parâmetros:

Prices - Pandas dataframe com os fechamentos ajustados de cada dia de cotação das ações;

Todos os métodos de estimativa de retornos utilização como base de previsão retornos históricos e as limitações de métodos como esses são muitas em relação a processos estocásticos.

Modelos de Risco

A estimação de matrizes de covariância é um passo necessário para a otimização de um portfólio, seja em modelos otimização de portfólio voltados a rentabilidade ou redução de volatilidade e cresce em complexidade conforme aumenta o número de ativos dentro dessa carteira.

Estimativas através da base histórica ou *sample covariance methods* são métodos similares retornos esperados que se valem de dados do passado. Mais uma vez nos deparamos com o problema de performance passada pode não ser uma boa forma de estimar o futuro, entretanto quando falamos em estimar volatilidade estamos em um campo.

Matrizes de covariância são matrizes que contém variância e covariância das variáveis analisadas. Nas diagonais temos a variância de suas variáveis e nos demais elementos da tabela temos a covariância entre os itens da matriz. Quando transformamos a matriz de covariância em uma matriz de correlação temos a diagonal da matriz contendo valor igual a 1.

Na biblioteca PyPortfolio temos o módulo risk_models para estimar os modelos de risco e seus métodos cada um explicado detalhadamente abaixo:

pypfopt.risk_models.risk_matrix(*prices, method='sample_cov', **kwargs*)

Parâmetros:

- prices: Pandas dataframe com os fechamentos ajustados de cada dia de cotação das ações;

- returns_data: boolean default é falso se quiser imputar retornos marcar como true

- method: string com o método desejado dentro da lista:

{sample_cov, semicovariance, exp_cov, min_cov_determinant, ledoit_wolf, ledoit_wolf_constant_variance, ledoit_wolf_single_factor, ledoit_wolf_constant_correlation, oracle_approximating}

O PyPortfolio também traz cada um dos métodos com seus parâmetros que podem ser alterados:

Sample Covariance

pypfopt.risk_models.sample_cov(*prices, returns_data=False, frequency=252, **kwargs*)

Essa função retorna a matriz de covariância dos retornos dos ativos diários em termos anuais em um formato de Pandas DataFrame.

Parâmetros:

- prices: Pandas Dataframe com matriz de preços ajustados podendo agrupar os retornos dos ativos em cada coluna

- returns_data: booleano, default é falso, colocar verdadeiro caso você queira passar os retornos calculados ao invés dos preços diretamente;

- frequency: número de dias de pregão no ano (default é 252)

Semicovariance

pypfopt.risk_models.semicovariance(*prices, returns_data=False, benchmark=7.9e-05, frequency=252, **kwargs*)

Como já tratado anteriormente nesse livro, a semivariância é uma forma de medir o *downside risk ou* risco de perda. No caso dessa função retorna um Pandas DataFrame com a covariância dos preços

abaixo de um parâmetro utilizado como benchmark onde normalmente é utilizada a taxa livre de risco da economia.

Parâmetros: Pandas Dataframe com matriz de preços ajustados podendo agrupar os retornos dos ativos em cada coluna

- prices: Pandas Dataframe com matriz de preços ajustados podendo agrupar os retornos dos ativos em cada coluna

Métodos de achatamento de matrizes de covariância

Os métodos de achatamento das matrizes de covariância são um campo relativamente novo que tem como objetivo reduzir o erro das previsões da matriz de covariância dentro dos modelos de otimização.

A metodologia consiste em combinar dois estimadores para achar um meio termo entre eles fazendo com que a matriz de covariância se achate para ficar mais próximo do objetivo. Para maiores informações checar as referências bibliográficas dos artigos de Ledoit -Wolf.

Covariance-Shrinkage

Método para realizar o achatamento da matriz de covariância utilizando a matriz de covariância da amostra permitindo que o viés de achatamento seja imputado como parâmetro

Variáveis de instancia:

X – pd.DataFrame (retornos);

S – np.ndarray (matriz de covariância);

Delta – float (constante de achatamento);

Frequency – integer (padrão diário 252 dias de negociação)

Parâmetros da função:

Prices – formato pd.Dataframe são os preços de fechamento do ativo analisado

Returns_data – default é False, colocar verdadeiro se o dataframe acima passado for de retornos percentuais

Frequency – default é diário considerando 252 dias de negociação em um ano

Ledoit-Wolf

Calcula o achatamento da matriz de covariância através do método Ledoit-Wolf

Parâmetros da função:

Shrinkage_target: tipo string, parâmetro opcional que pode adicionar um objetivo de achatamento na função – opções: constant_variance (default), single_factor or constant_correlation

Funções Objetivo

Dentro do módulo objective_functions encontramos os objetivos de otimização encontrados no método EfficientFrontier. Necessitam como parâmetros weights (pesos), expected_returns (retornos esperados) ou cov_matrix (matriz de covariância).

Função de Regularização L2

Funções de regularização também conhecidas como funções de decaimento de peso dentro da biblioteca PyPortfolio tem como objetivo reduzir o número de pesos-zero dentro do seu modelo de otimização. Os métodos de otimização tendem a zerar os pesos de

alguns ativos dentro da sua carteira quando esses destoam muito do objetivo de otimização que está sendo aplicado.

Métodos de pós-processamento

Os métodos de pós-processamento da biblioteca PyPortfolio consistem em como alocar recursos financeiros a partir de determinados métodos que seguem uma lógica.

Se você tem R$1000,00 disponível e o otimizador te forneceu que o peso ideal para você alocar em ações da Petrobrás é 10% então você deve alocar R$100 reais em ações da Petrobrás. Entretanto as ações da Petrobrás são comercializadas em valores discretos (R$18,90) ou seja, você não irá conseguir comprar R$100,00 de ações exatamente.

Discrete Allocation

Pypfopt.discrete_allocation.DiscreteAllocation

Parâmetros:

Weights – dicionário de pesos

Latest_prices – pd.Series dos valores mais recentes de preços das ações que estão sendo otimizadas

Total_portfolio_value – integer ou float

Short_ratio – float colocar o valor de proporção caso o portfolio seja long and short (exemplo: 1.3)

Algoritmo Greedy

Pypfopt.discrete_allocation.DiscreteAllocation.greedy_allocation()

Algoritmo vai agir em duas rodadas: na primeira ela irá comprar a quantidade máxima possível de Petro e seguirá para os outros ativos Exemplo: com R$100,00 conseguirá comprar 5 ações inteiras de Petro (R$94,50). Ao finalizar a primeira iteração terá uma "sobra" (R$100-R$94,50 = R$5,50). As diferenças irão variar de ativo para ativo algumas serão maiores e outras menores, o algoritmo irá distribuir as "sobras" primeiro para as "sobras" maiores.

Parâmetros:

Reinvest – reinvestir ou não o dinheiro ganho nas posições short

Verbose – imprime erros e o resultado da alocação

LP_Portfólio

Converte os valores contínuos dos pesos em quantidade de ações dado um determinado valor de portfólio

Parâmetros:

Reinvest - reinvestir ou não o dinheiro ganho nas posições short

Verbose - imprime erros e o resultado da alocação

Solver – default é o solver GLPK_MI da biblioteca CVXPY (consultar referências para maiores informações)

Modelos de Otimização de Portfólio

As decisões de alocação de investimento podem são influenciadas por fatores externos como taxa de juros, dólar e outros fatores econômicos. Mas também podem sofrer restrições em relação ao nível de risco desejado pelo investidor, nível de retorno esperado, restrições de capital a ser investido ou reinvestido ou ainda o gestor pode ter uma estratégia de limitar sua exposição em determinado setor ou ativo, por exemplo ficar exposto somente em no máximo 10% no setor de saúde pois ele acredita que essa é uma exposição segura dentro desse setor específico do mercado

Para isso a biblioteca PyPortfolio conta com métodos para que facilitam inclusão de restrições dentro de modelos de otimização. Dentro da biblioteca por trás do modelo CVXPY foram construídos método e classes que possibilitam deixar a construção de modelos com restrições mais complexas como as citadas acima.

Utilizaremos o dataframe dos exemplos anteriores “df” para estimar retornos e matriz de covariância.

Retorno Esperado

A próxima etapa do framework é preparar o vetor de retornos previstos, iremos utilizar para essa tarefa o módulo expected_returns da biblioteca PyPortfolio:

```
26
27  from pypfopt import expected_returns
28
```

Seguindo para o próximo passo, iremos utilizar neste projeto o modelo CAPM como previsão de retornos. Para isso precisamos preparar os parâmetros de Selic diária e do Ibovespa como carteira de mercado:

```
29  #Calcular a selic diária
30  selic_aa = 0.019
31  selic_diaria = (1+selic_aa)**(1/252)-1
32
33  #Ibovespa para o período passado
34  ibov = wb.DataReader('^BVSP', data_source='yahoo', start='2016-01-01', end='2018-12-31')
35  ibov = ibov.drop(ibov.columns[[0,1,2,3,4]], axis=1)
36  ibov
```

Para realizar a previsão dos retornos através do modelo CAPM, três parâmetros são necessários: os preços (df), market_prices (ibovespa), risk_free_rate (selic_diária)

```
37
38  re = expected_returns.capm_return(df, market_prices=ibov,
39                                    risk_free_rate=selic_diaria)
```

O resultado desse objetivo **re** é um pandas.core.series.Series com as previsões em taxa anualizada:

```
In [7]: re
Out[7]:
ITSA4.SA    0.309250
PETR4.SA    0.499773
ABEV3.SA    0.131194
VALE3.SA    0.333165
Name: mkt, dtype: float64
```

Matriz de Covariância

Nesta etapa faremos uma previsão da matriz de covariância dos ativos ela é parte fundamental dos modelos de otimização em relação ao risco. Primeiro iremos importar o módulo risk_models:

```
44
45  from pypfopt import risk_models
46
```

Para esse exemplo utilizaremos o método disponível de achatamento da matriz de covariância e o redutor de erros de Ledoit-Wolf:

```
47  sample_cov = risk_models.CovarianceShrinkage(df).ledoit_wolf()
48
```

Portfólio de Mínima-Volatilidade

Passada as etapas de estimar retornos e matriz de covariância, nosso primeiro objetivo de otimização será a redução de volatilidade, ou seja, uma combinação de pesos que indique o menor ponto de volatilidade do portfólio. Para isso iremos importar o módulo

Efficient Frontier que trará os otimizadores que utilizaremos nessa seção:

```
49
50    from pypfopt import EfficientFrontier
51
```

A próxima etapa é construir o objeto mv na EfficientFrontier e aplicar o método de min_volatility() para realizar a otimização:

```
55
56    mv = EfficientFrontier(re, sample_cov)
57    mv.min_volatility()
```

O método clean_weights nos ajuda a criar um objeto lista para os pesos do portfólio otimizado:

```
59
60    pesos_vol = mv.clean_weights()
61    pesos_vol
```

O output dessa função são os pesos que o algoritmo de otimização nos dá para cada ativo:

```
In [16]:
    ...:
    ...: pesos_vol = mv.clean_weights()
    ...: pesos_vol
Out[16]:
OrderedDict([('ITSA4.SA', 0.19474),
             ('PETR4.SA', 0.0),
             ('ABEV3.SA', 0.76268),
             ('VALE3.SA', 0.04258)])
```

Podemos notar que o ativo PETR4.SA está com peso zerado, muitas vezes isso pode acontecer devido a disparidade de volatilidade que ele tem perante os outros ativos da carteira, o algoritmo age até o limite da otimização e como não há restrições de peso mínimo ele irá passar exatamete a melhor solução matemática para o problema de otimização.

Uma maneira de contornar esses problemas de alocação é através da função de regularização L2 que está dentro do módulo objective_functions:

```
63
64    from pypfopt import objective_functions
65
```

Vamos construir novamente a estrutura de mínima volatilidade dessa vez adicionando como objetivo da otimização a função regularizadora com fator gamma de 10%, dessa maneira ela irá tratar esses pesos zero de maneira diferente conferindo a eles alguma distribuição de percentual:

```
66    mv_2 = EfficientFrontier(re, sample_cov)
67    mv_2.add_objective(objective_functions.L2_reg, gamma=0.1)
68    mv_2.min_volatility()
69    pesos_2 = mv_2.clean_weights()
70    pesos_2
```

O output desse bloco de código será o dicionário de pesos, note que já temos algum percentual para o ativo que outrora estava zerado:

```
    ...: pesos_2
Out[18]:
OrderedDict([('ITSA4.SA', 0.30597),
             ('PETR4.SA', 0.06065),
             ('ABEV3.SA', 0.48311),
             ('VALE3.SA', 0.15027)])
```

Maximização do índice de Sharpe

O otimizador tem como objetivo otmizar a relação prêmio-pelo-risco-retorno através da clássica fórmula de Índice de Sharpe (Taxa livre de risco – Retorno da Carteira)/Volatilidade da Carteira

```
#Maximização de Sharpe
msharpe = EfficientFrontier(re,sample_cov)
msharpe.max_sharpe(risk_free_rate=selic_aa)
sharpe_pesos = msharpe.clean_weights()
sharpe_pesos
```

O output do objeto sharpe_pesos será um dicionário com os pesos otimizados:

```
Out[9]:
OrderedDict([('ITSA4.SA', 0.4986),
             ('PETR4.SA', 0.22803),
             ('ABEV3.SA', 0.12986),
             ('VALE3.SA', 0.14351)])
```

Podemos transformar o dicionário em lista para auxiliar nos cálculos. Abaixo extraímos os valores dos pesos do dicionário e depois o transformamos em lista:

```
#Transformando o dicionário em lista de pesos
lista_sharpe = sharpe_pesos.values()
lista_sharpe = list(lista_sharpe)
lista_sharpe
```

Risco Eficiente

O modelo de otimização risco eficiente permite que você estabeleça um objetivo de volatilidade para o seu portfólio dessa maneira permitindo que se calcule a máxima rentabilidade dado esse objetivo. Nesse caso teremos como objetivo 20% de volatilidade:

```
#Risco Eficiente
r_eficiente = EfficientFrontier(re, sample_cov)
r_eficiente.efficient_risk(target_volatility=0.2)
r_eficiente_pesos = r_eficiente.clean_weights()
r_eficiente_pesos
```

O output de r_eficiente_pesos será:

```
Out[12]:
OrderedDict([('ITSA4.SA', 0.29767),
             ('PETR4.SA', 0.00212),
             ('ABEV3.SA', 0.62226),
             ('VALE3.SA', 0.07795)])
```

Retorno Eficiente

Seguindo a mesma ideia do risco eficiente, só que com papéis trocados: queremos a mínima volatilidade para um objetivo de retorno específico. Nesse caso nosso objetivo com o portfólio é um retorno anualizado de 25% passado no argumento da função **target_return**:

```
85    #Retorno Eficiente
86    re_eficiente = EfficientFrontier(re, sample_cov)
87    re_eficiente.efficient_return(target_return=0.25)
88    re_eficiente_pesos = re_eficiente.clean_weights()
89    re_eficiente_pesos
```

No output teremos os pesos otimizados dado nosso objetivo de retorno:

```
Out[14]:
OrderedDict([('ITSA4.SA', 0.37252),
             ('PETR4.SA', 0.08628),
             ('ABEV3.SA', 0.43883),
             ('VALE3.SA', 0.10237)])
```

Modelos de Otimização com restrições

Quando estamos definindo uma estratégia de portfólio muitas vezes podemos ter algumas regras de ouro observadas a partir de outros inputs de dados que não gostaríamos de quebrar. Imagine que você

tem uma carteira e criou um racional que você tem que ter pelo menos 10% de ações ligadas a commodities a ela, pois você acredita que o cenário econômico atual é favorável a um aumento de preços em commodities de forma geral.

Na biblioteca PyPortfolio temos funções prontas para compreender essas situações a partir de parâmetros facilmente passados através dos argumentos das funções de otimização, saindo dos modelos de solver complexos que teríamos que construir para modelos com múltiplas restrições funcionarem.

Restrições Setoriais

Nesse exemplo iremos segregar nossa carteira em setores e construir um modelo onde queremos no máximo 10% de exposição no setor de varejo e mais que 5% de exposição em commodities. Para isso essa função de otimização espera 3 dicionários: um com os ativos e os determinados setores (sector_mapper), outra com o parâmetro maior igual (sector_lower) e o parâmetro menor igual (sector_upper)

```
93    sector_mapper = {
94        "ITSA4.SA" : "FINANCEIRO",
95        "PETR4.SA" : "COMMODITIES",
96        "ABEV3.SA" : "VAREJO",
97        "VALE3.SA" : "COMMODITIES"
98    }
99
100   sector_lower = {"COMMODITIES": 0.05} #>= 5% em commodities
101   sector_upper = {"VAREJO": 0.10} #<=10% em varejo
102
103
104   rest_setor = EfficientFrontier(re, sample_cov)
105   rest_setor.add_sector_constraints(sector_mapper, sector_lower, sector_upper)
106   rest_setor.max_sharpe()
107   rest_setor_pesos = rest_setor.clean_weights()
108   rest_setor_pesos
```

O output desse bloco de código através o objeto rest_setor_pesos:

```
Out[5]:
OrderedDict([('ITSA4.SA', 0.51622),
             ('PETR4.SA', 0.23507),
             ('ABEV3.SA', 0.1),
             ('VALE3.SA', 0.14871)])
```

Restrição de ativo específico

Depois de analisar como se dá um modelo de otimização com restrições setoriais construídas a partir de dicionários, temos agora as restrições de ações específicas. Nesse modelo é passado a quantidade exata ou dentro de regra de qual a exposição que teremos em cada ativos específico. Veja o exemplo abaixo onde limitamos a exposição em PETR4.SA em 10%:

Primeiro iremos construir o objeto a ser otimizado:

```
110    #Restrição de ação Específica
111
112    restricao_acao = EfficientFrontier(re, sample_cov)
```

Depois iremos construir as restrições de otimização, note que buscamos o ticker da ação através de tickers.index para trazer exatamente o papel PETR4.SA:

```
114    #restrição
115    petr = restricao_acao.tickers.index("PETR4.SA")
116    restricao_acao.add_constraint(lambda w: w[petr] <=0.10)
117
```

E por último seguimos com a estrutura anterior de realizar a otimização e extrair os pesos selecionados:

```
118     #otimização
119     restricao_acao.max_sharpe()
120
121     #pesos
122     pesos_acao = restricao_acao.clean_weights()
123     pesos_acao
```

O output de pesos_acao será:

```
Out[8]:
OrderedDict([('ITSA4.SA', 0.56387),
             ('PETR4.SA', 0.1),
             ('ABEV3.SA', 0.16816),
             ('VALE3.SA', 0.16797)])
```

Considerações Finais

Modelos de otimização de portfólio baseado na teoria da média-variância sofrem das limitações amplamente discutidas nesse volume.

Na prática vemos que os modelos de otimização que utilizam como input principal os retornos esperados calculados a partir dos métodos oferecidos pela biblioteca PyPortfolio tendem a ter resultados piores justamente pela dificuldade de estimar retornos através das ferramentas que levam em conta histórico de retornos. Soluções alternativas a esses problemas seria estimar retornos através de outros métodos de precificação de ativos como o valuation através do fluxo de caixa descontado. Na prática somente substituiria o parâmetro retorno esperado dos objetos de Efficiente Frontier por pesos estimados por modelos fora da

Referências

- Strategic Risk Taking – Damoradan – Prentice Hall, 2009

- Sharpe, William F. "Capital Asset Pricing Model: A Theory of Market Equilibrium under Conditions of Risk. Journal of Finance, 19 (3) – 425-442. Lintner, J. 1965. Disponível em: https://psc.ky.gov/pscecf/2012-00221/rateintervention@ag.ky.gov/10252012f/sharpe_-_CAPM.pdf

-Estrada (2006), Mean-Semivariance Optimization: A Heuristic Approach

- Ledoit, O., & Wolf, M. (2003). Honey, I Shrunk the Sample Covariance Matrix The Journal of Portfolio Management, 30(4), 110–119. https://doi.org/10.3905/jpm.2004.110

- Ledoit, O., & Wolf, M. (2001). Improved estimation of the covariance matrix of stock returns with an application to portfolio selection, 10, 603–621.

[1] https://valorinveste.globo.com/objetivo/hora-de-investir/noticia/2020/03/31/ibovespa-tem-maior-queda-mensal-em-22-anos-dolar-maior-alta-desde-ataque-as-torres-gemeas-em-2011.ghtml

[2] PyPortfolio Library Documentation: https://pyportfolioopt.readthedocs.io/en/latest/About.html

Autoria de Robert Martin – Cambridge University