Tutorial de Utilização do Código

Este código Python, em formato de **Jupyter Notebook**, implementa a **Teoria Moderna de Portfólios (MPT)** de Markowitz para otimizar uma carteira de investimentos usando o método de **Simulação de Monte Carlo** com uma validação de consistência.

Bloco 1: Instalação e Importação de Bibliotecas

Este é o primeiro passo para garantir que o ambiente de execução tenha todas as ferramentas necessárias.

import numpy as np

import pandas as pd

import yfinance as yf

import matplotlib.pyplot as plt

from sklearn.linear_model import LinearRegression

- **O que faz:** Importa as bibliotecas essenciais para o projeto.
    - **numpy**: Usada para realizar operações matemáticas complexas, especialmente com matrizes e vetores, que são a base da MPT.
    - **pandas**: Fundamental para a manipulação e organização de dados em tabelas (DataFrames), como os preços e retornos dos ativos.
    - **yfinance**: Uma biblioteca poderosa para baixar dados históricos de ações, moedas e outros ativos do Yahoo Finance.
    - **matplotlib.pyplot**: Utilizada para a criação de gráficos e visualizações, como a fronteira eficiente.
    - **sklearn.linear_model.LinearRegression**: Embora o núcleo da otimização seja Monte Carlo, esta biblioteca é incluída para a etapa opcional de regressão linear para uma análise extra do modelo.
- **Como usar:** Você não precisa alterar nada neste bloco. Basta executá-lo. Certifique-se de que as bibliotecas estão instaladas em seu ambiente (pip install numpy pandas yfinance matplotlib scikit-learn).

Bloco 2: Coleta de Dados e Preparação

Aqui, o código se conecta a uma fonte de dados real e prepara as informações para a análise.

ativos = ["WEGE3.SA","ITUB4.SA",...]

precos = yf.download(ativos, start="2006-01-01", end="2024-12-31")["Close"]

 # Ajustar preços antes do IPO...

retornos = precos.pct_change().fillna(0)

 # Divisão treino / teste

metade = len(retornos) // 2

retornos_treino = retornos.iloc[:metade].copy()

retornos_teste = retornos.iloc[metade:].copy()

 # Parâmetros da simulação

num_ativos = len(retornos.columns)

num_carteiras_simuladas = 30000

- **O que faz:**
    - **Sub-bloco 1 (ativos)**: Define a lista de ativos que você deseja analisar.
    - **Sub-bloco 2 (precos)**: Baixa os preços históricos de fechamento dos ativos no período especificado.
    - **Sub-bloco 3 (Ajuste e Retornos)**: O código ajusta os preços de ativos que foram listados após a data de início do período, preenchendo os valores anteriores com o preço da primeira data válida. Em seguida, calcula os retornos percentuais diários e preenche os valores ausentes (NaN) com 0.
    - **Sub-bloco 4 (Divisão)**: Divide os retornos em duas metades: **retornos_treino** para a simulação e **retornos_teste** para validar a consistência.
    - **Sub-bloco 5 (Parâmetros)**: Define o número de ativos e a quantidade de carteiras que serão simuladas.
- **Como usar:**
    1. **Altere os ativos:** Você pode mudar a lista ativos para incluir os ativos de sua preferência. Certifique-se de usar os tickers corretos do Yahoo Finance.

2. **Mude o período:** Altere as datas de início (**start**) e fim (**end**) para o período que você deseja analisar.
3. **Ajuste o número de simulações:** O valor **num_carteiras_simuladas** (padrão 30000) pode ser alterado. Um número maior aumentará a precisão, mas também o tempo de execução.

Bloco 3: Definição de Funções de Desempenho

Este bloco define as fórmulas matemáticas para calcular o desempenho de qualquer portfólio.

def desempenho_portfolio(pesos, retornos, taxa_livre_risco=0.06):

retorno_portfolio = np.dot(pesos, retornos.mean()) * 252

volatilidade_portfolio = np.sqrt(np.dot(pesos.T, np.dot(retornos.cov() * 252, pesos)))

sharpe_ratio = (retorno_portfolio - taxa_livre_risco) / volatilidade_portfolio

return {...}

def desempenho_portfolio_treino(...)

def desempenho_portfolio_real(...)

- **O que faz:**
  - **desempenho_portfolio**: Recebe um vetor de pesos e os retornos dos ativos.
    - Calcula o **retorno anualizado** do portfólio: a média dos retornos diários ponderada pelos pesos, multiplicada por 252 (aproximadamente o número de dias úteis em um ano).
    - Calcula a **volatilidade anualizada**: usa a matriz de covariância dos retornos para determinar o risco total da carteira. A matriz de covariância é um componente central da MPT.
    - Calcula o **Índice de Sharpe**: a diferença entre o retorno do portfólio e a taxa livre de risco, dividida pela volatilidade.
  - **Funções auxiliares**: As funções **desempenho_portfolio_treino** e **desempenho_portfolio_real** são apenas "invólucros" que chamam a função principal com os dados de treino ou teste, respectivamente.

- **Como usar:** Você não precisa alterar este bloco, a menos que queira mudar a **taxa_livre_risco**. O valor padrão de 0.06 (6% ao ano) pode ser ajustado para refletir a taxa de juros atual (por exemplo, a taxa Selic no Brasil ou a taxa de juros do Fed nos EUA).

Bloco 4: Otimização e Filtro de Consistência

Este bloco de código é o ponto central da metodologia de otimização. Ele vai além de simplesmente encontrar a carteira com o maior Índice de Sharpe, introduzindo uma fase crucial de **validação de consistência** para garantir que a carteira ótima é robusta e não apenas um resultado da sorte no período de treino.

1) Checagens Rápidas e Padronização

Esta subseção do código é dedicada a garantir que as variáveis necessárias existem e que os dados estão no formato esperado. É uma etapa de segurança fundamental.

- **R2_THRESHOLD = 0.3**: Define o parâmetro principal para o filtro de consistência. Este valor representa a diferença máxima permitida entre o Índice de Sharpe no treino e no teste para que um portfólio seja considerado consistente.
- **TAXA_LIVRE_RISCO_ANUAL** = 0.06: Define a taxa de retorno livre de risco a ser usada no cálculo do Sharpe.
- **if 'df_treino' not in globals() or ...**: Realiza uma verificação de pré-execução. Se qualquer uma das variáveis (**df_treino, df_teste** ou **pesos_treino_df**) não tiver sido criada nos blocos anteriores, o código para e lança um erro (RuntimeError), informando ao usuário que o ambiente não está pronto.
- **if len(df_treino) != len(df_teste) or ...**: Garante que todos os **DataFrames** de resultados e de pesos têm o mesmo número de linhas, o que é vital para garantir que a correspondência entre eles está correta.
- **def _standardize_columns(df)**: Esta é uma função utilitária para padronizar os nomes das colunas nos **DataFrames**. Ela procura por **substrings** como **'ret', 'vol'** ou **'sharpe'** e renomeia as colunas para retorno, volatilidade e **sharpe_ratio**, respectivamente. Isso evita erros caso as colunas tenham nomes ligeiramente diferentes, tornando o código mais robusto.
- **df_treino = _standardize_columns(df_treino)**: Aplica a função de padronização ao **DataFrame** de treino.
- **df_teste = _standardize_columns(df_teste)**: Aplica a função de padronização ao **DataFrame** de teste.

- **for name, df in ...**: Uma checagem final para garantir que as colunas essenciais (retorno e volatilidade) existem após a padronização. Se não existirem, o código interrompe a execução com um erro, impedindo que a próxima etapa de cálculo falhe.

2) Calcular a Diferença (Consistência) Entre Treino e Teste por Carteira

Esta seção realiza os cálculos necessários para a análise de consistência.

- **df_treino['sharpe_ratio'] = ...**: Recalcula o Índice de Sharpe para o **DataFrame** de treino. Embora essa métrica já tenha sido calculada antes, esta linha a garante que ela seja calculada com a **TAXA_LIVRE_RISCO_ANUAL** definida no topo do bloco, padronizando o cálculo.
- **df_teste['sharpe_ratio'] = ...**: Faz o mesmo cálculo para o **DataFrame** de teste.
- **df_treino['sharpe_ratio'].replace(...)**: Remove valores inválidos (**inf ou -inf**) do Índice de Sharpe, substituindo-os por **NaN**. Isso é crucial porque esses valores podem ocorrer em portfólios com volatilidade zero e causariam erros nos cálculos subsequentes.
- **df_teste['sharpe_ratio'].replace(...)**: Aplica a mesma substituição no **DataFrame** de teste.
- **sharpe_diff = (df_treino['sharpe_ratio'] - df_teste['sharpe_ratio']).abs()**: Esta linha é o coração do filtro. Ela calcula a **diferença absoluta** entre o Índice de Sharpe de treino e de teste para cada uma das 30.000 carteiras simuladas. O método **.abs()** garante que o resultado é sempre positivo.

3) Aplicar Filtro de Consistência

Esta etapa usa a diferença de Sharpe calculada para selecionar os portfólios robustos.

- **mask_selected = sharpe_diff <= R2_THRESHOLD**: Cria uma **máscara booleana**. Para cada carteira, esta máscara armazena **True** se a **sharpe_diff** for menor ou igual ao **R2_THRESHOLD**, e **False** caso contrário. Apenas as carteiras com **True** serão consideradas na próxima etapa.
- **selected_count = int(mask_selected.sum())**: Soma os valores **True** na máscara (**True** é interpretado como 1, **False** como 0) para obter o número total de carteiras que passaram no filtro.
- **print(...)**: Exibe o número de carteiras que foram consideradas consistentes, dando um feedback instantâneo ao usuário.

4) Selecionar a Melhor Carteira Entre as Filtradas

Esta seção finaliza o processo, escolhendo a melhor carteira entre o subconjunto de portfólios consistentes.

- **if selected_count == 0**:: Verifica se alguma carteira passou no filtro. Se **selected_count** for zero, o código informa que não é possível continuar e não executa o restante do bloco, prevenindo erros.
- **sharpe_teste_selected = df_teste['sharpe_ratio'].loc[mask_selected].dropna()**: Esta linha é crucial. Ela seleciona os valores do Índice de Sharpe do período de teste **apenas para as carteiras que passaram no filtro (mask_selected).** O **.dropna()** remove qualquer valor **NaN** que possa ter sobrado, garantindo que o cálculo do máximo seja preciso.
- **if len(sharpe_teste_selected) == 0:**: Uma checagem adicional. Caso o **.dropna()** tenha removido todas as carteiras, o código informa que não há portfólios válidos para seleção.
- **best_index = sharpe_teste_selected.idxmax()**: Encontra o índice (a posição) do maior valor de Sharpe no subconjunto de carteiras consistentes. Este é o índice da carteira ótima e robusta.
- **best_sharpe_value = sharpe_teste_selected.loc[best_index]**: Obtém o valor real do Sharpe para a carteira ideal.
- print(...): Imprime o índice e o valor do Índice de Sharpe da carteira selecionada.
- **if best_index in pesos_treino_df.index**:: Esta verificação é uma medida de segurança para garantir que o índice encontrado corresponde a um índice real no **DataFrame** de pesos.
- **best_weights = pesos_treino_df.loc[best_index]**: Usa o índice da carteira ideal para extrair a alocação de pesos correspondente do **DataFrame pesos_treino_df**. A variável **best_weights** agora contém a composição ideal da carteira.

Bloco 5: Análise Adicional de Carteiras de Destaque

Este bloco é dedicado a identificar outras carteiras importantes que foram simuladas, mas que não foram necessariamente a melhor carteira consistente. O objetivo é fornecer um ponto de comparação para entender a diferença entre uma carteira otimizada para o **Índice de Sharpe consistente** e carteiras que apenas maximizam ou minimizam uma única métrica.

max_sharpe_idx = df_teste["Sharpe"].idxmax()

min_vol_idx = df_teste["Volatilidade"].idxmin()

max_vol_idx = df_teste["Volatilidade"].idxmax()

max_retorno_idx = df_teste["Retorno"].idxmax()

- **O que faz:**
    - **max_sharpe_idx**: Encontra o índice da carteira que teve o **maior Índice de Sharpe** no período de teste, sem considerar o filtro de consistência.
    - **min_vol_idx**: Encontra o índice da carteira que teve a **menor volatilidade** no período de teste.
    - **max_vol_idx**: Encontra o índice da carteira que teve a **maior volatilidade** no período de teste.
    - **max_retorno_idx**: Encontra o índice da carteira que teve o **maior retorno** no período de teste.
- **Como usar:** Este bloco não requer alterações. Você pode usar os índices resultantes para extrair informações sobre essas carteiras e compará-las com a carteira ótima selecionada no Bloco 6. Isso ajuda a visualizar a importância da consistência na seleção.

Bloco 6: Análise e Visualização do Modelo de Regressão (Validação de Sanidade)

Este é o último bloco e é crucial para validar visualmente a robustez dos resultados. Ele utiliza um modelo de regressão para verificar se as métricas de desempenho do portfólio são previsíveis com base em seus pesos.

 # ----------

 # Bloco de regressão linear para validar o modelo

 # ----------

alvos = ["Retorno", "Volatilidade", "Sharpe"]

modelo_regressao = LinearRegression()

X_treino = pesos_treino_df.loc[mask_selected].values

y_treino = df_treino.loc[mask_selected, alvos].values

modelo_regressao.fit(X_treino, y_treino)

X_teste = pesos_treino_df.loc[mask_selected].values

y_model = df_teste.loc[mask_selected, alvos].values

```
y_pred = modelo_regressao.predict(X_teste)
fig, axes = plt.subplots(1, 3, figsize=(18, 5))
for i, alvo in enumerate(alvos):
    ax = axes[i]
    ax.scatter(y_model[:, i], y_pred[:, i], alpha=0.5, s=20)
    mn = np.nanmin([y_model[:, i].min(), y_pred[:, i].min()])
    mx = np.nanmax([y_model[:, i].max(), y_pred[:, i].max()])
    ax.plot([mn, mx], [mn, mx], linestyle="--", color="k", linewidth=1)
    ax.set_title(f"{alvo}: y_true vs y_pred")
    ax.set_xlabel(f"{alvo} (teste)")
    ax.set_ylabel(f"{alvo} (previsto)")
    ax.grid(True)
plt.tight_layout()
plt.show()
```

- **O que faz:**
    - **alvos = [...]**: Define as métricas que o modelo de regressão tentará prever.
    - **modelo_regressao = LinearRegression()**: Cria uma instância do modelo de regressão linear.
    - **X_treino = ...:** Seleciona os pesos das carteiras que passaram no filtro de consistência. Estes serão os dados de entrada (X) para o modelo.
    - **y_treino = ...:** Seleciona as métricas de desempenho (retorno, volatilidade, Sharpe) das carteiras consistentes do período de treino. Estes serão os dados de saída (y) para o modelo.
    - **modelo_regressao.fit(X_treino, y_treino):** Treina o modelo de regressão, ensinando-o a encontrar a relação entre os pesos de uma carteira e seu desempenho.
    - **X_teste = ...** e **y_model = ...:** Prepara os dados de teste. **X_teste** são os pesos das carteiras consistentes, e **y_model** são as métricas de desempenho reais no período de teste.

- **y_pred = modelo_regressao.predict(X_teste):** O modelo treinado faz uma previsão **(y_pred)** das métricas de desempenho usando os pesos do período de teste.
- **Sub-bloco de Plotagem (fig, axes = ...)**: Esta parte cria três gráficos de dispersão, um para cada métrica (Retorno, Volatilidade, Sharpe).
    - Em cada gráfico, o eixo **x** representa o valor real da métrica no período de teste (**y_model**).
    - O eixo **y** representa o valor previsto pelo modelo de regressão **(y_pred).**
    - A linha tracejada **(ax.plot(...)**) é a **linha de identidade (y=x)**. Se os pontos se alinharem a essa linha, significa que o modelo de regressão previu o desempenho da carteira com grande precisão.

- **Como usar:** Este bloco é para análise. Sua função é validar se a relação entre a composição da carteira e seu desempenho é previsível. Se os pontos nos gráficos de dispersão estiverem próximos da linha tracejada, seu modelo é robusto. Uma grande dispersão sugere que há outros fatores não capturados que influenciam o desempenho da carteira.

Bloco 7: Gráfico de Comparação de Sharpe (Treino vs. Teste)

Este bloco é a visualização direta do filtro de consistência do Bloco 6. Ele cria um gráfico para mostrar como o Índice de Sharpe de cada portfólio se comporta em ambos os períodos, treino e teste.

plt.figure(figsize=(10, 6))

plt.scatter(df_treino['sharpe_ratio'], df_teste['sharpe_ratio'], alpha=0.5, s=20)

plt.title('Sharpe Ratio: Período de Treino vs. Período de Teste')

plt.xlabel('Sharpe Ratio (Treino)')

plt.ylabel('Sharpe Ratio (Teste)')

plt.grid(True)

plt.show()

- **O que faz:**
    - Ele plota um gráfico de dispersão (**scatter**).

- O eixo **X** representa o Índice de Sharpe de cada uma das 30.000 carteiras no período de **treino**.
- O eixo **Y** representa o Índice de Sharpe dessas mesmas carteiras no período de **teste**.
- Cada ponto no gráfico é uma carteira simulada.
- O gráfico permite uma visualização intuitiva da consistência: carteiras que são consistentes se agrupam perto de uma linha diagonal (y=x). Carteiras com grande diferença de Sharpe aparecem longe dessa linha.

- **Como usar:** Este bloco não precisa de modificações. Simplesmente execute-o para gerar a visualização. Ele serve como uma ferramenta de diagnóstico para entender a relação entre o desempenho histórico e a consistência.

Bloco 8: Análise Detalhada de Regressão (MSE e MAE)

Este bloco expande a análise de sanidade do Bloco 8, calculando métricas de erro para quantificar a precisão do modelo de regressão linear. Ele avalia o quão bem o modelo consegue prever o desempenho do portfólio.

```
from sklearn.metrics import mean_squared_error, mean_absolute_error


mse = mean_squared_error(y_model, y_pred)
mae = mean_absolute_error(y_model, y_pred)


print(f"MSE (Erro Quadrático Médio): {mse:.6f}")
print(f"MAE (Erro Absoluto Médio): {mae:.6f}")


print("\nAnálise Detalhada do Modelo de Regressão:")
print("----------------------------------------")
for i, alvo in enumerate(alvos):
    mse_alvo = mean_squared_error(y_model[:, i], y_pred[:, i])
    mae_alvo = mean_absolute_error(y_model[:, i], y_pred[:, i])
    print(f"  {alvo}:")
    print(f"    MSE: {mse_alvo:.6f}")
    print(f"    MAE: {mae_alvo:.6f}")
```

- **O que faz:**
    - **from sklearn.metrics ...:** Importa as funções para calcular o **Erro Quadrático Médio (MSE)** e o **Erro Absoluto Médio (MAE)**.
    - **mse = ...** e **mae = ...:** Calcula o MSE e o MAE para todas as métricas combinadas, fornecendo um valor geral do erro de previsão do modelo.
        - **MSE**: Penaliza erros maiores de forma mais severa (por elevar ao quadrado a diferença), sendo sensível a outliers.
        - **MAE**: Representa a média das diferenças absolutas entre os valores reais e previstos. É mais intuitivo e menos sensível a outliers.
    - **for i, alvo in enumerate(alvos)::** Um loop que calcula e exibe o MSE e o MAE **individualmente** para cada métrica (Retorno, Volatilidade e Sharpe). Isso permite uma análise mais precisa da performance de previsão do modelo para cada variável.
- **Como usar:** Este bloco fornece uma análise quantitativa para complementar os gráficos do Bloco 8. Um MSE e um MAE próximos de zero indicam que o modelo de regressão linear está fazendo boas previsões, o que reforça a confiabilidade dos seus resultados de consistência.