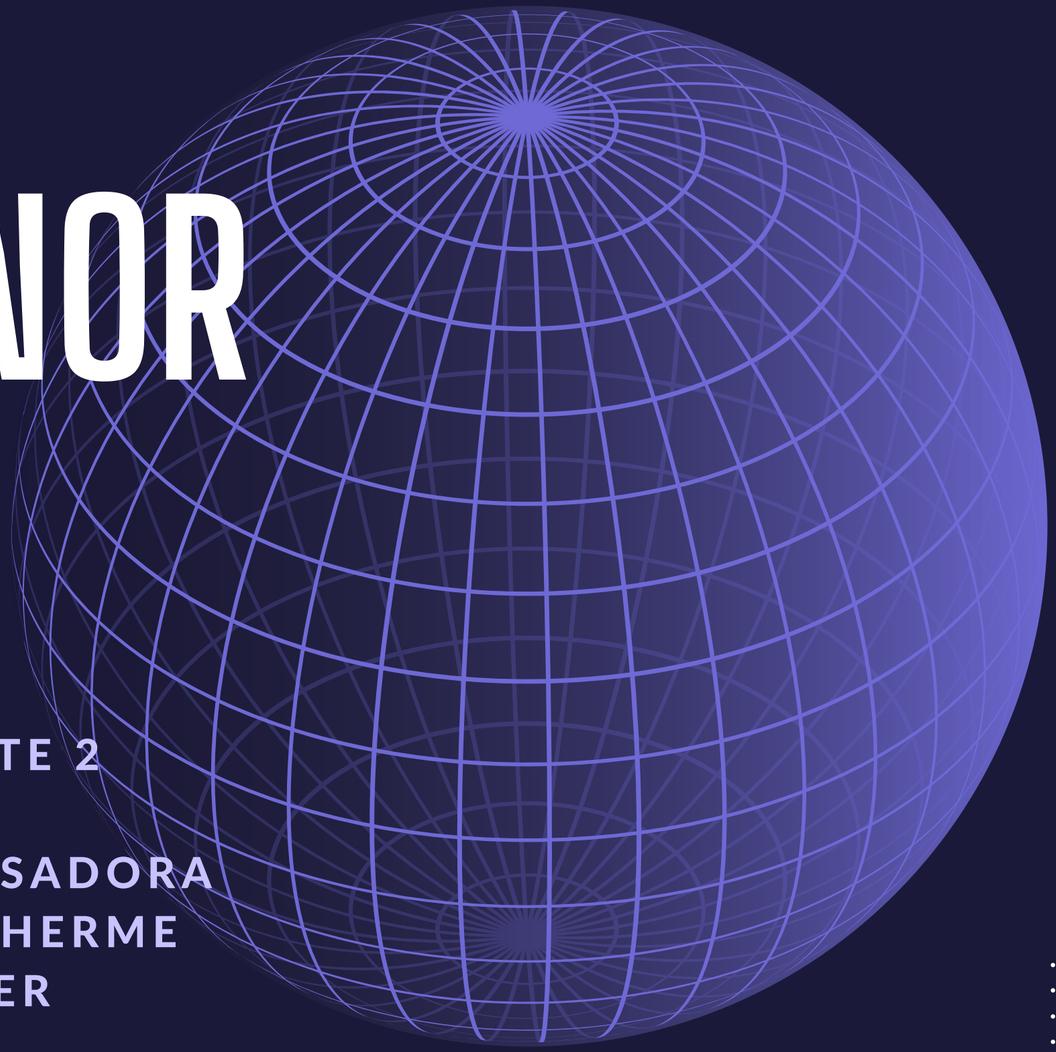
001

DESAFIO FINOR
GRUPO 1

CASE 2 - INDEX TRACKING - PARTE 2

JOÃO VICTOR LENZI CARDOSO, ISADORA MULLER, GABRIEL NOVAES, GUILHERME FREITAS E RAQUEL GUTSCHWAGER



INDEX TRACKING:

O que é?

- Consiste em montar carteiras de investimentos que acompanhem as variações de índices de bolsas de valores;
- As carteiras devem possuir um número de ativos menor do que o índice original;
- Esta prática serve para reduzir o custo de manutenção e acompanhamento do portifólio;
- No desafio proposto, deve-se acompanhar a variação dos índices BOVESPA e S&P100 para os últimos 7 anos.

PARTE 2 – MVP

Minimum Viable Product

Após feita a EDA, chegou a hora de aplicar o modelo de otimização proposto. Na etapa atual, o foco está em desenvolver uma versão preliminar do projeto que já possua o mínimo das funcionalidades essenciais implementadas.

Realizamos a preparação dos dados para serem utilizados como inputs no algoritmo, bem como obtivemos nossos primeiros resultados - muito animadores.

Para tanto, foi utilizada a linguagem python junto das bibliotecas Pandas, numpy, matplotlib e yfinance, além do solver comercial Gurobi - utilizando de uma licença acadêmica concedida pela UFRGS.

MVP

IMPORTAÇÃO E PREPARO DE DADOS

Novamente os dados do índice de interesse - neste caso, o S&P100 - foram importados. Também importamos os dados de todos os seus tickers em sua última composição e adicionamos alguns ativos que podem ser interessantes, como Bitcoin, ouro e commodities.

```python
# Baixando dados dos Indice desejados (S&P100 e IBOV)
tickers = ['^BVSP', '^SP100']

# Baixar e arrumar o DataFrame
indices = yf.download(tickers, start = '2017-01-01', end = '2024-04-30')
indices = indices.stack().reset_index(level=1)
indices.columns.name = None

# Filtrando para SP100
filtro_SP100 = indices['Ticker'] == '^SP100'
SP100 = indices.copy()
SP100 = SP100[filtro_SP100]
SP100
```

✓ 2.5s Python

```python
## Importando ações do S&P100 e outros ativos interessantes

# Essa lista foi definida através da Wikipedia
lista_sp100 = ['AAPL', 'ABBV', 'ABT', 'ACN', 'ADBE', 'AIG', 'AMD', 'AMGN', 'AMT', 'AMZN', 'AVGO', 'AXP', 'BA', 'BAC',
               'BK', 'BKNG', 'BLK', 'BMY', 'BRK-B', 'C', 'CAT', 'CHTR', 'CL', 'CMCSA', 'COF', 'COP', 'COST', 'CRM', 'CS
               'DHR', 'DIS', 'DOW', 'DUK', 'EMR', 'F', 'FDX', 'GD', 'GE', 'GILD', 'GM', 'GOOG', 'GOOGL', 'GS', 'HD', 'H
               'JNJ', 'JPM', 'KHC', 'KO', 'LIN', 'LLY', 'LMT', 'LOW', 'MA', 'MCD', 'MDLZ', 'MDT', 'MET', 'META', 'MMM',
               'MSFT', 'NEE', 'NFLX', 'NKE', 'NVDA', 'ORCL', 'PEP', 'PFE', 'PG', 'PM', 'PYPL', 'QCOM', 'RTX', 'SBUX', '
               'T', 'TGT', 'TMO', 'TMUS', 'TSLA', 'TXN', 'UNH', 'UNP', 'UPS', 'USB', 'V', 'VZ', 'WFC', 'WMT', 'XOM', 'S
               'SI=F', 'CC=F', 'CL=F', 'KC=F', 'CT=F', 'SB=F', 'OJ=F', 'NG=F', 'HG=F']

dfsp = yf.download(lista_sp100, start = '2017-01-01', end = '2024-04-30')
dfsp = dfsp.stack().reset_index(level=1)
dfsp.columns.name = None
dfsp
```

✓ 9.1s Python

MVP

IMPORTAÇÃO E PREPARO DE DADOS

Foram calculados os retornos de todos os Tickers importados, onde os mesmos já foram organizados em um DataFrame próprio para ser inserido no solver.

Este loop calcula os seus retornos acumulados de forma individual em um DataFrame temporário e depois os anexa ao DataFrame principal.

```python
# Calculando o retorno de todos os Tickers e juntando em um DataFrame

# Inicializacao de variaveis para o loop for
retornos = pd.DataFrame()
temp = []

# loop que cria um unico Data Frame com todos os retornos
for nome in dfsp['Ticker'].unique():

    # Esta primeira parte cria um dataframe temporario apenas com o ticker da iteracao atual do loop
    filtro = dfsp['Ticker'] == nome
    temp_lag = len(temp) # armazenando o numero de linhas do ticker anterior
    temp = dfsp[filtro]

    # Condicional pra remover tickers com menos linhas que os demais (dados faltantes)
    # e indexar corretamente o número de linhas em tickers com dados demais.
    if len(temp) < temp_lag:
      continue
    elif len(temp) > temp_lag:
      filtro2 = temp.index.isin(dfsp.query('Ticker == "AAPL"').index)
      temp = temp[filtro2]

    # Esta segunda parte calcula o retorno acumulado do ticker atual e depois o anexa ao dataframe principal de retornos, test_retornos
    calculo = temp.groupby('Ticker').apply(lambda x: calculo_retorno(x['Close']), include_groups=False).T
    retornos = retornos.join(calculo, how = 'outer')

retornos.dropna(inplace=True)
retornos
```

✓ 2.5s Python

MVP

IMPORTAÇÃO E PREPARO DE DADOS

Foram calculados os retornos de todos os Tickers importados, onde os mesmos já foram organizados em um DataFrame próprio para ser inserido no solver.

Este loop calcula os seus retornos acumulados de forma individual em um DataFrame temporário e depois os anexa ao DataFrame principal.

Date	BTC-USD	AAPL	ABBV	ABT	ACN	ADBE	AIG	AMD	AMGN	AMT	...	TXN	UNH	UNP	UPS	U
2017-01-04	0.106233	-0.001119	0.014100	0.007939	0.002404	0.006378	0.013053	0.000000	0.014198	0.001790	...	-0.001223	0.002849	0.006048	0.000261	0.007
2017-01-05	-0.122410	0.005085	0.007584	0.008638	-0.014991	0.016996	-0.005609	-0.016623	0.000720	-0.003479	...	-0.007756	0.001668	-0.009793	0.000521	-0.016
2017-01-06	-0.109711	0.011148	0.000314	0.027204	0.011392	0.022566	0.018445	0.007117	0.024840	-0.006606	...	0.016868	0.001418	0.010379	0.001997	0.006
2017-01-09	0.000695	0.009159	0.006584	-0.000981	-0.011178	0.002493	-0.006436	0.015018	0.013139	-0.002375	...	0.002562	-0.002832	-0.007462	-0.005459	-0.004
2017-01-10	0.005373	0.001009	-0.002180	0.013500	0.000522	-0.002855	0.012353	-0.004352	-0.000504	-0.014473	...	0.003497	-0.002285	0.009764	-0.007406	0.002
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
2024-04-23	-0.006440	0.006392	0.009828	0.004857	-0.001859	0.012872	-0.000400	0.024421	0.005995	0.010911	...	0.012482	-0.010280	0.007762	0.024147	-0.001
2024-04-24	-0.032080	0.012702	-0.010263	-0.006506	-0.010384	0.008924	0.000534	-0.003481	-0.001938	-0.004765	...	0.056445	0.002304	-0.018240	-0.015114	0.015
2024-04-25	0.003186	0.005147	-0.003039	-0.000281	-0.014480	-0.007713	-0.004535	0.013312	-0.013296	-0.002596	...	0.002517	0.013462	0.049875	0.005252	-0.014
2024-04-26	-0.011265	-0.003473	-0.045849	0.006270	-0.003204	0.008702	-0.001340	0.023673	0.002227	-0.006998	...	0.012725	0.003017	-0.003121	0.001357	0.003
2024-04-29	0.001346	0.024808	0.011903	-0.002418	-0.015746	-0.009402	0.016235	0.017789	0.023705	0.019221	...	0.010198	-0.012759	-0.007579	0.006505	0.004

1840 rows × 110 columns

MVP

IMPORTAÇÃO E PREPARO DE DADOS

Para um posterior teste da eficácia de nossos portfólios, separamos os dados dos últimos 7 anos em dados de treino e dados de teste, tanto para os tickers quanto para o índice, onde 90% foram destinados ao treino do algoritmo e o restante será usado como benchmark..

```python
# Checando numero de linhas do dataframe de retornos dos ativos
len(retornos)

# Separando dados entre teste e treino (ten-fold)
retornos_treino = retornos.head(round(len(retornos)*0.9))
retornos_teste = retornos.tail(len(retornos) - len(retornos_treino))
```

✓ 0.0s Python

```python
# Teste e treino SP100
retornos_SP100_treino = retornos_SP100.head(len(retornos_treino))
retornos_SP100_teste = retornos_SP100.tail(len(retornos_teste))
retornos_SP100_treino
```

✓ 0.0s Python

```
Date
2017-01-04    0.003268
2017-01-05    0.000269
2017-01-06    0.003993
2017-01-09   -0.002480
2017-01-10   -0.001154
                ...
2023-07-28    0.012390
2023-07-31    0.001014
2023-08-01   -0.002922
2023-08-02   -0.015540
2023-08-03   -0.001855
Name: Close, Length: 1656, dtype: float64
```

MVP

CRIAÇÃO DE POSSÍVEIS CESTAS DE ATIVOS

Como o número de combinações a se fazer cresce de forma computacionalmente proibitiva com o aumento de ativos ou diminuição da carteira, decidimos criar listas reduzidas do total de tickers, com 40 itens cada, onde subconjuntos de até 20 ativos seus serão selecionados pelo algoritmo de Index Tracking (pode ser revisado). A primeira lista possui uma mescla entre os 20 ativos com maior e menor rendimentos acumulados.

```
tickers_ordenados = ordem_acumulados.columns.tolist()
lista_acumulados = tickers_ordenados[:20] + tickers_ordenados[-20:]
lista_acumulados
```

[10] ✓ 0.0s

```
['BTC-USD',
 'NVDA',
 'TSLA',
 'AMD',
 'AAPL',
 'LLY',
 'MSFT',
 'ADBE',
 'AVGO',
 'INTU',
 'DE',
 'TMO',
 'MA',
 'AMZN',
 'COST',
 'LIN',
 'GOOG',
 'NFLX',
 'DHR',
 'GOOGL',
 'GILD',
 'BMY',
 'INTC',
 'BK',
 'AIG',
...
 'VZ',
 'MMM',
 'GE',
 'T',
 'KHC']
```

MVP

CRIAÇÃO DE POSSÍVEIS CESTAS DE ATIVOS

A segunda lista possui os 40 ativos com maior desvio-padrão em seus rendimentos, representando uma carteira considerada mais volátil.

```
# Ordenando pelos maiores desvios padrao
lista_std_maiores = ordem_std[:40]
lista_std_maiores
```

✓ 0.0s

```
['CL=F',
 'BTC-USD',
 'NG=F',
 'TSLA',
 'AMD',
 'SWBI',
 'NVDA',
 'BA',
 'NFLX',
 'SPG',
 'COP',
 'META',
 'PYPL',
 'GE',
 'COF',
 'QCOM',
 'F',
 'GM',
 'AIG',
 'SCHW',
 'CRM',
 'INTC',
 'C',
 'ADBE',
 'AVGO',
...
 'MET',
 'CVX',
 'USB',
 'DE',
 'LOW']
```

MVP

CRIAÇÃO DE POSSÍVEIS CESTAS DE ATIVOS

A terceira lista possui os 40 ativos com menor desvio-padrão em seus rendimentos - representando uma carteira mais bem-comportada, menos volátil.

```
# Ordenando pelos menores desvios padrao
lista_std_menores = ordem_std[-40:]
lista_std_menores
```

✓ 0.0s

```
['AMT',
 'CSCO',
 'ABBV',
 'V',
 'HD',
 'TMO',
 'ACN',
 'MMM',
 'GILD',
 'PM',
 'DHR',
 'IBM',
 'LIN',
 'T',
 'NEE',
 'MO',
 'HON',
 'ABT',
 'MDT',
 'LMT',
 'AMGN',
 'PFE',
 'SO',
 'GD',
 'BMY',
...
 'PG',
 'VZ',
 'KO',
 'JNJ',
 'GC=F']
```

MVP

CRIAÇÃO DE POSSÍVEIS CESTAS DE ATIVOS

A quarta lista possui uma mescla dos 20 ativos com maior e 20 ativos com menor desvio-padrão em seus rendimentos - representando uma carteira de volatilidade moderada.

```
# Misto entre maior e menor std
lista_std_misto = ordem_std[:20] + ordem_std[-20:]
lista_std_misto
```

✓ 0.0s

```
['CL=F',
 'BTC-USD',
 'NG=F',
 'TSLA',
 'AMD',
 'SWBI',
 'NVDA',
 'BA',
 'NFLX',
 'SPG',
 'COP',
 'META',
 'PYPL',
 'GE',
 'COF',
 'QCOM',
 'F',
 'GM',
 'AIG',
 'SCHW',
 'AMGN',
 'PFE',
 'SO',
 'GD',
 'BMY',
...
 'PG',
 'VZ',
 'KO',
 'JNJ',
 'GC=F']
```

MVP

CRIAÇÃO DE POSSÍVEIS CESTAS DE ATIVOS

Por fim, criamos mais três listas com 40 ativos aleatórios em cada. Utilizamos seeds diferentes na geração de cada uma, e que estão disponíveis no notebook para garantir reprodutibilidade dos resultados obtidos.

```
# Realizando amostragem aleatoria de forma reprodutivel
lista_todos = pd.Series(lista_todos)
lista_aleatoria1 = lista_todos.sample(n = 40, replace = False, random_state = 1).tolist()
lista_aleatoria1
```

✓ 0.0s

```
# Realizando amostragem aleatoria de forma reprodutivel
lista_todos = pd.Series(lista_todos)
lista_aleatoria2 = lista_todos.sample(n = 40, replace = False, random_state = 12).tolist()
lista_aleatoria2
```

✓ 0.0s

```
# Realizando amostragem aleatoria de forma reprodutivel
lista_todos = pd.Series(lista_todos)
lista_aleatoria3 = lista_todos.sample(n = 40, replace = False, random_state = 12345).tolist()
lista_aleatoria3
```

✓ 0.0s

MVP

CRIAÇÃO DE POSSÍVEIS CESTAS DE ATIVOS

Para definirmos e resolvermos o problema de minimização proposto, utilizamos o solver Gurobi, onde os DataFrames de treino foram passados como arrays e a minimização de suas diferenças quadráticas pro índice foram calculadas.

Para reprodutibilidade dos resultados, também usamos uma seed fixa, disponível no notebook.

Definindo a função

```python
# Função de otimização do portfólio
def otimizacao(x_in, y_in, K, lista, time_limit=300):
    import gurobipy as gp
    from gurobipy import GRB
    import pandas as pd
    import numpy as np

    T, N = x_in.shape

    # Criação do modelo
    model = gp.Model("portfolio_optimization")
    model.setParam(gp.GRB.Param.OutputFlag, 0) # suprimir a output usual do gurobi
    model.setParam('Seed', 0) # seed que garante reprodutibilidade de resultados
    model.Params.TimeLimit = time_limit # limite de tempo
    model.Params.MIPGap = 0.001 # precisão do gurobi

    # Variáveis de decisão
    w = model.addVars(N, lb=0.0, ub=1.0, vtype=GRB.CONTINUOUS, name="w")  # Pesos
    z = model.addVars(N, vtype=GRB.BINARY, name="z")  # Seleção

    # Função objetivo
    obj_expr = gp.quicksum((gp.quicksum(x_in[t, i] * w[i] for i in range(N)) - y_in[t]) ** 2 for t in range(T))
    model.setObjective(obj_expr, GRB.MINIMIZE)

    # Restrições
    model.addConstr(gp.quicksum(w[i] for i in range(N)) == 1, name="soma dos pesos")
    model.addConstr(gp.quicksum(z[i] for i in range(N)) == K, name="binario de selecao")

    # Vinculando variáveis w e z
    M = 1  # Valor grande o suficiente para garantir a restrição correta
    for i in range(N):
        model.addConstr(w[i] <= M * z[i], name=f"vinculo1_{i}")
        model.addConstr(w[i] >= z[i] * 1e-6, name=f"vinculo2_{i}")
```

MVP

RESOLVENDO O PROBLEMA

Começamos então a parte mais interessante. Foram utilizadas como variáveis primeiro as 3 cestas aleatórias, com seus respectivos ativos escolhidos e pesos ao lado. As outputs mais detalhadas se encontram no notebook.

```
# aplicando o modelo para o primeiro portfolio aleatorio
lista = lista_aleatoria1 # lista a ser usada na otimizacao
x_in = np.array(retornos_treino[lista])
y_in = np.array(retornos_SP100_treino)
tickers = otimizacao(x_in=x_in, y_in=y_in, K=20, lista=lista)
tickers
```

✓ 7.5s

```
# Listando os indices escolhidos e seus pesos
portfolio = pd.Series(tickers)
filtro = portfolio != 0
portfolio = portfolio[filtro]
portfolio
```

✓ 0.0s

```
SB=F    0.122047
JNJ     0.089113
KC=F    0.055505
ABBV    0.067032
USB     0.005321
UPS     0.032457
GC=F    0.503162
MRK     0.001417
CVS     0.039324
KHC     0.001562
CT=F    0.083059
dtype: float64
```

MVP

RESOLVENDO O PROBLEMA

Começamos então a parte mais interessante. Foram utilizadas como variáveis primeiro as 3 cestas aleatórias, com seus respectivos ativos escolhidos e pesos ao lado. As outputs mais detalhadas se encontram no notebook.

```
# aplicando o modelo para o segundo portfolio aleatorio
lista = lista_aleatoria2 # lista a ser usada na otimizacao
x_in = np.array(retornos_treino[lista])
y_in = np.array(retornos_SP100_treino)
tickers = otimizacao(x_in=x_in, y_in=y_in, K=20, lista=lista)
tickers
```

✓ 6.5s

```
# Listando os indices escolhidos e seus pesos
portfolio = pd.Series(tickers)
filtro = portfolio != 0
portfolio = portfolio[filtro]
portfolio
```

✓ 0.0s

```
DHR     0.023381
KO      0.228571
PFE     0.099673
HG=F    0.360787
KC=F    0.133450
DUK     0.031519
CL      0.037219
MMM     0.017042
CHTR    0.027007
T       0.041351
dtype: float64
```

MVP

RESOLVENDO O PROBLEMA

Começamos então a parte mais interessante. Foram utilizadas como variáveis primeiro as 3 cestas aleatórias, com seus respectivos ativos escolhidos e pesos ao lado. As outputs mais detalhadas se encontram no notebook.

```
# aplicando o modelo para o terceiro portfolio aleatorio
lista = lista_aleatoria3 # lista a ser usada na otimizacao
x_in = np.array(retornos_treino[lista])
y_in = np.array(retornos_SP100_treino)
tickers = otimizacao(x_in=x_in, y_in=y_in, K=20, lista=lista)
tickers
```

✓ 7.0s

```
# Listando os indices escolhidos e seus pesos
portfolio = pd.Series(tickers)
filtro = portfolio != 0
portfolio = portfolio[filtro]
portfolio
```

✓ 0.0s

```
LMT     0.025105
PFE     0.013940
CVS     0.005797
SWBI    0.017354
GC=F    0.488634
MO      0.125304
KO      0.052991
ABBV    0.063486
HG=F    0.182202
NG=F    0.025186
dtype: float64
```

MVP

RESOLVENDO O PROBLEMA

Seguem tambem os ativos escolhidos na lista de retornos acumulados...

```
# aplicando o modelo para o portfolio de maiores e menores retornos acumulados
lista = lista_acumulados # lista a ser usada na otimizacao
x_in = np.array(retornos_treino[lista])
y_in = np.array(retornos_SP100_treino)
tickers = otimizacao(x_in=x_in, y_in=y_in, K=20, lista=lista)
tickers
```

✓ 6.6s

```
# Listando os indices escolhidos e seus pesos
portfolio = pd.Series(tickers)
filtro = portfolio != 0
portfolio = portfolio[filtro]
portfolio
```

✓ 0.0s

```
BTC-USD    0.006922
TSLA       0.001637
TMO        0.049171
AMZN       0.016332
COST       0.069340
GILD       0.046738
BMY        0.200234
SWBI       0.043556
NG=F       0.045942
MO         0.184201
VZ         0.284731
MMM        0.051197
dtype: float64
```

MVP

RESOLVENDO O PROBLEMA

...maiores desvios-padrão...

```
# aplicando o modelo para o portfolio de maiores desvios-padrao
lista = lista_std_maiores # lista a ser usada na otimizacao
x_in = np.array(retornos_treino[lista])
y_in = np.array(retornos_SP100_treino)
tickers = otimizacao(x_in=x_in, y_in=y_in, K=20, lista=lista)
tickers
```

✓ 7.9s

```
# Listando os indices escolhidos e seus pesos
portfolio = pd.Series(tickers)
filtro = portfolio != 0
portfolio = portfolio[filtro]
portfolio
```

✓ 0.0s

```
BTC-USD    0.014940
NG=F       0.067977
SWBI       0.073116
NFLX       0.023483
GE         0.021615
F          0.030949
FDX        0.021713
BKNG       0.038937
AMZN       0.089687
TGT        0.095798
KC=F       0.321104
CVX        0.016625
USB        0.107263
DE         0.046288
LOW        0.030504
dtype: float64
```

MVP

RESOLVENDO O PROBLEMA

...menores desvios-padrão....

```
# aplicando o modelo para o portfolio de menores std
lista = lista_std_menores # lista a ser usada na otimizacao
x_in = np.array(retornos_treino[lista])
y_in = np.array(retornos_SP100_treino)
tickers = otimizacao(x_in=x_in, y_in=y_in, K=20, lista=lista)
tickers
```

✓ 7.1s

```
# Listando os indices escolhidos e seus pesos
portfolio = pd.Series(tickers)
filtro = portfolio != 0
portfolio = portfolio[filtro]
portfolio
```

✓ 0.0s

```
ABBV    0.035535
MO      0.109369
LMT     0.000378
GD      0.010910
BMY     0.074821
HG=F    0.180533
WMT     0.009304
VZ      0.099714
KO      0.007811
GC=F    0.471625
dtype: float64
```

MVP

RESOLVENDO O PROBLEMA

...e desvios-padrão. mistos


```
# aplicando o modelo para o portfolio de std mistos
lista = lista_std_misto # lista a ser usada na otimizacao
x_in = np.array(retornos_treino[lista])
y_in = np.array(retornos_SP100_treino)
tickers = otimizacao(x_in=x_in, y_in=y_in, K=20, lista=lista)
tickers
```

✓ 8.7s

```
# Listando os indices escolhidos e seus pesos
portfolio = pd.Series(tickers)
filtro = portfolio != 0
portfolio = portfolio[filtro]
portfolio
```

✓ 0.0s

```
NG=F    0.023271
SWBI    0.017350
GD      0.028113
BMY     0.092360
HG=F    0.188271
WMT     0.014880
VZ      0.121394
KO      0.059184
JNJ     0.000283
GC=F    0.454894
dtype: float64
```

MVP

RESOLVENDO O PROBLEMA

Após realizados os cálculos para o período de testes com todas as cestas de ativos, organizamos suas séries de retornos em um dataframe único.

Date	Portfolio1	Portfolio2	Portfolio3	Acumulados	Maiores std	Menores std	Misto std	SP100
2023-08-07	1.004775	1.003317	1.000994	1.005115	1.015958	0.997985	0.999200	1.009055
2023-08-08	0.999241	0.995156	0.995864	1.005110	1.010106	0.992338	0.993415	1.005643
2023-08-09	0.998513	0.997857	0.996868	1.009811	1.009448	0.992166	0.995567	0.995903
2023-08-10	0.999974	0.995046	0.993291	1.010500	1.001492	0.992554	0.994588	0.996817
2023-08-11	1.003723	0.992726	0.991737	1.013034	0.998070	0.991245	0.994241	0.995243
...	...	...	...	...	...	...	...	...
2024-04-23	1.089116	1.093781	1.133372	1.053129	1.253745	1.138583	1.141799	1.149446
2024-04-24	1.086933	1.098431	1.129138	1.046365	1.248578	1.135453	1.136521	1.148957
2024-04-25	1.084256	1.104797	1.136409	1.025799	1.249496	1.135347	1.133060	1.141185
2024-04-26	1.080517	1.104866	1.133468	1.030983	1.251204	1.135702	1.135930	1.157424
2024-04-29	1.091633	1.118433	1.150399	1.047796	1.279618	1.144895	1.150805	1.160114

MVP

RESULTADOS

Feito isso, começamos a análise de resultados. Começamos calculando o RMSE (Raíz do erro quadrático médio) para todas as cestas difetentes
O RMSE difere do MAE (Erro absoluto médio) por aumentar o peso dos outliers nos erros, penalizando mais uma previsção muito distante do valor real.
Podemos ver que a cesta com a maior volatilidade surpreendentemente obteve de longe o melhor desempenho.

```python
# Definindo funcao rmse (root-mean-squared-error)
def rmse(x,y):
    root_mean_square_error = np.sqrt((((x-y)**2).mean()))
    return(root_mean_square_error)

# inicializando listas para o loop
RMSE_list = []
pfolio_list = []

for pfolio in acumulados.columns:
    if pfolio != 'SP100': # ignorando os valores de SP100
        RMSE_list.append(rmse(acumulados[pfolio], acumulados['SP100']))
        pfolio_list.append(pfolio)

# transformando resultados em um dicionario e listando-os
pfolio_RMSE = {pfolio_list[i]: RMSE_list[i] for i in range(len(pfolio_list))}
pd.Series(pfolio_RMSE)
```

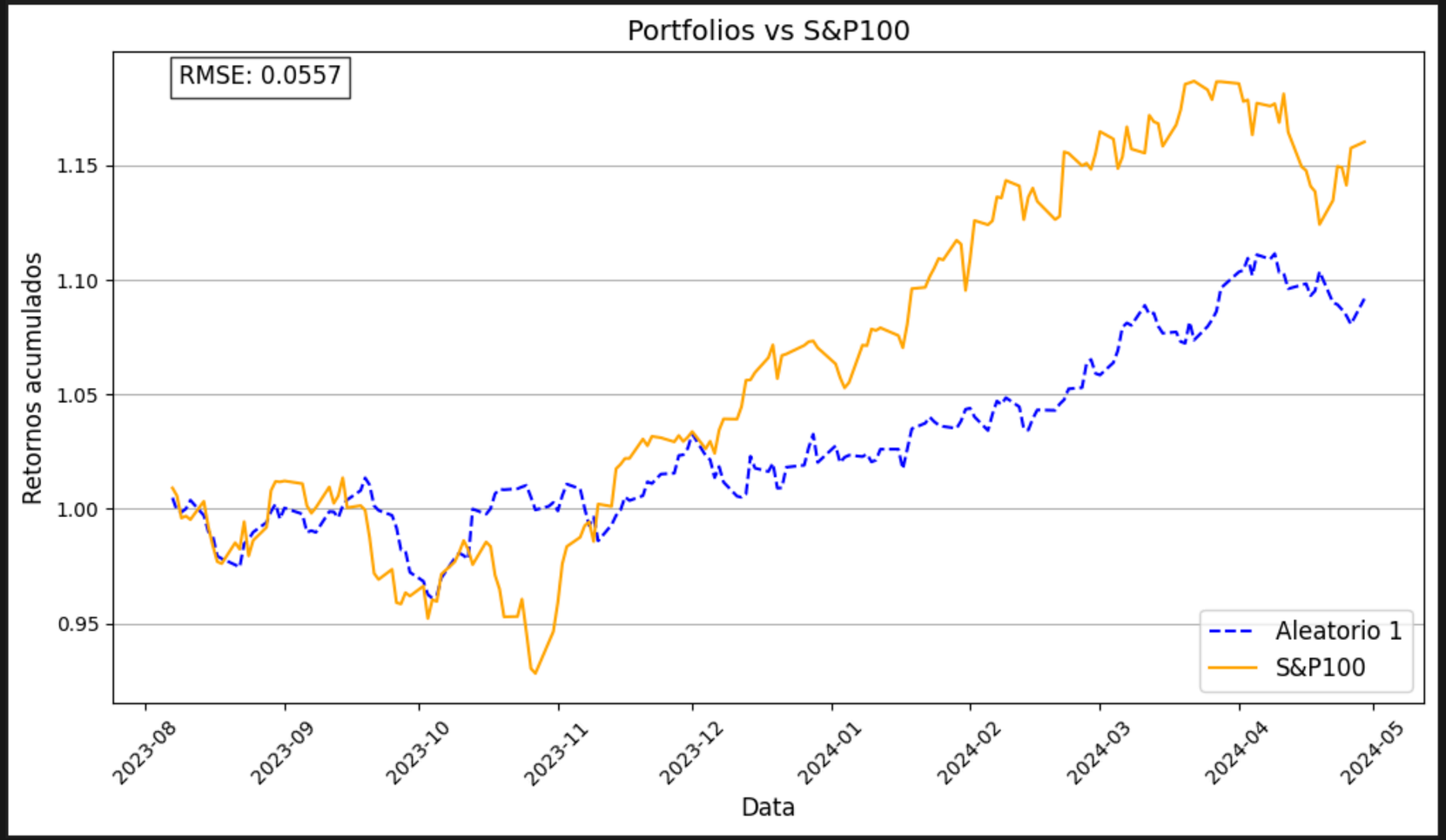
✓ 0.0s

```
Portfolio1     0.055676
Portfolio2     0.080367
Portfolio3     0.062905
Acumulados     0.065878
Maiores std    0.044502
Menores std    0.054565
Misto std      0.052683
dtype: float64
```

MVP

RESULTADOS - VISUALIZAÇÃO

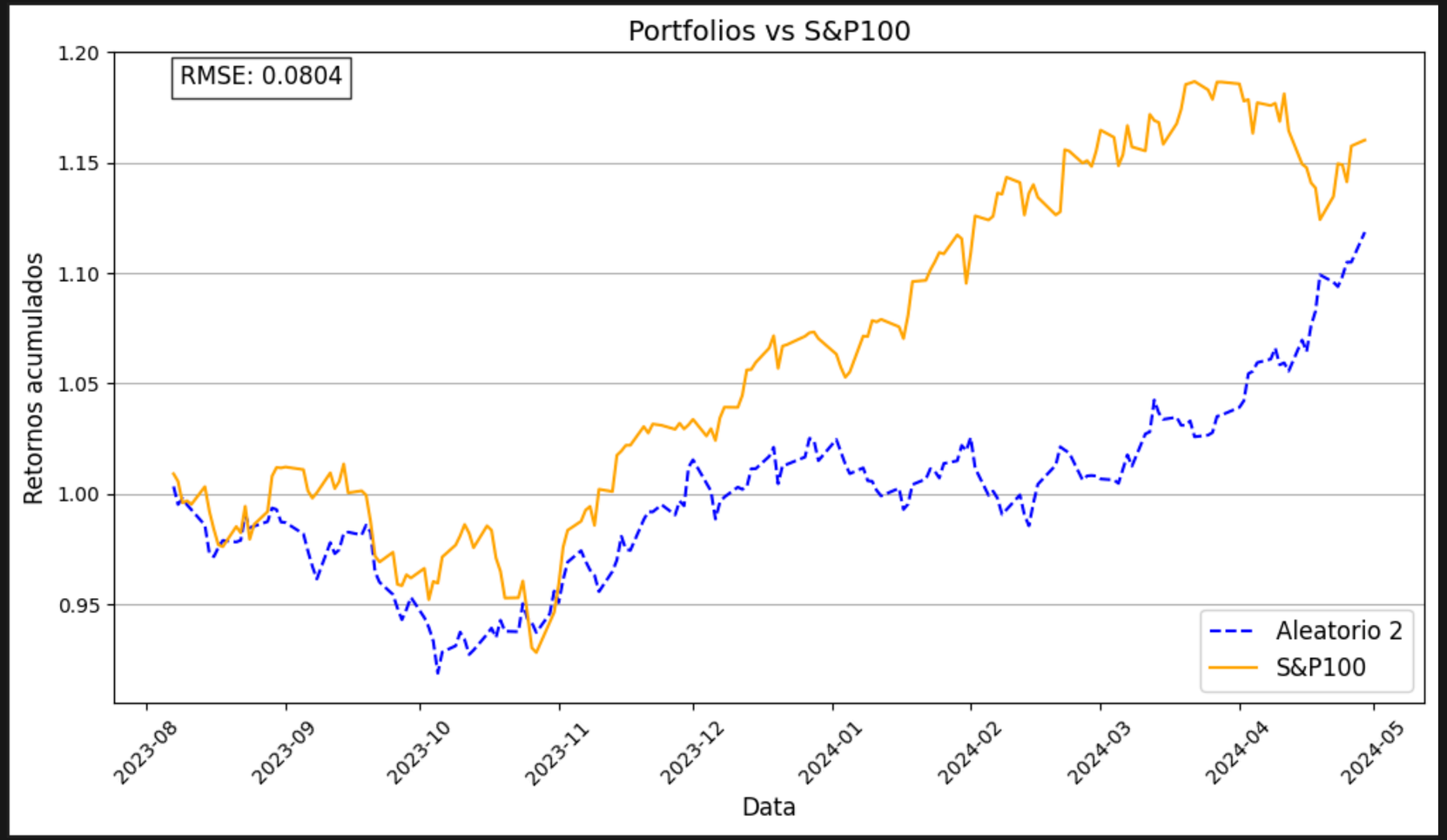
Trazemos aqui então uma breve visualização dos retornos de nossos diversos portfólios comparados aos retornos do S&P100.



MVP

RESULTADOS - VISUALIZAÇÃO

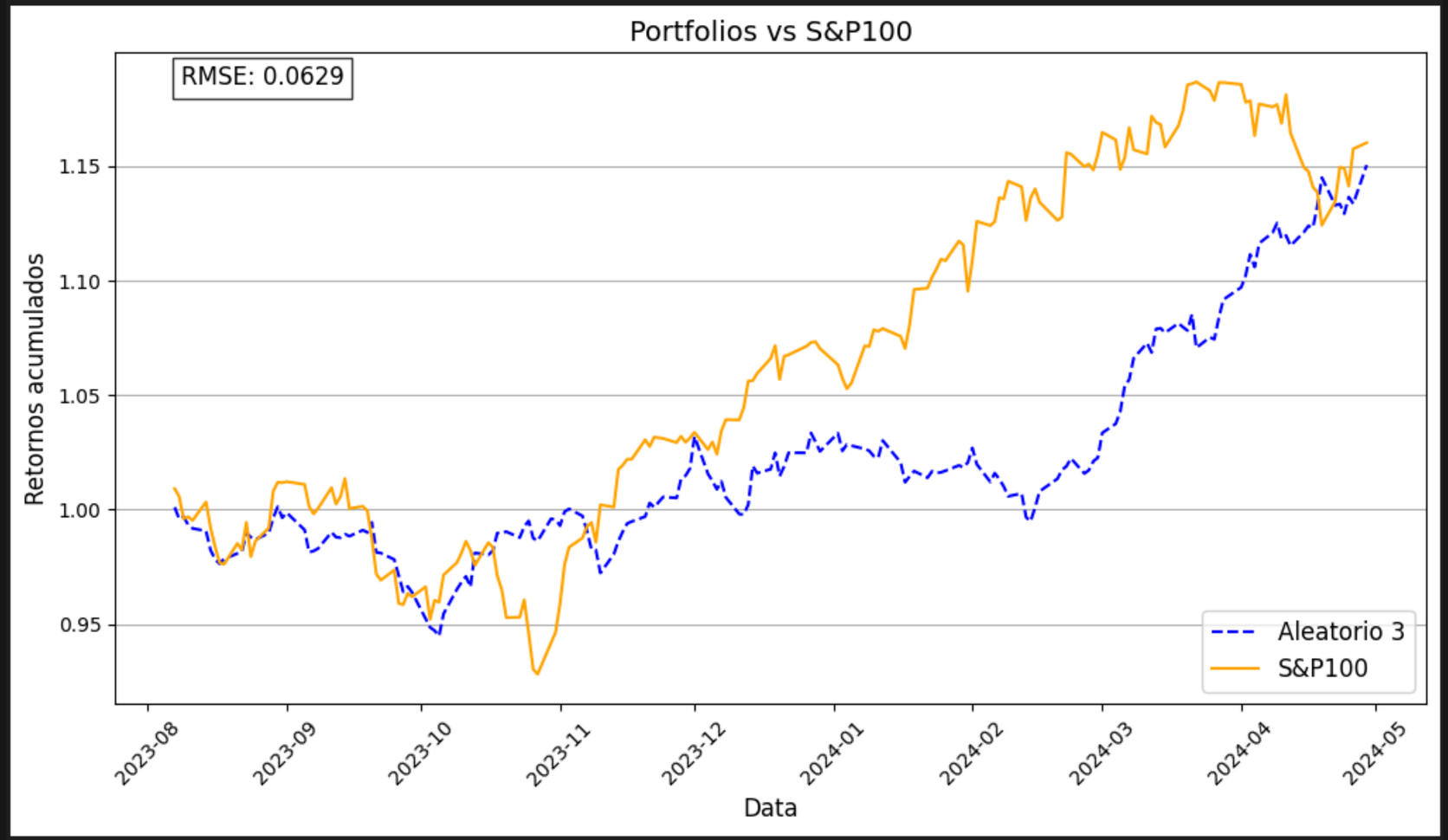
Trazemos aqui então uma breve visualização dos retornos de nossos diversos portfólios comparados aos retornos do S&P100.



MVP

RESULTADOS - VISUALIZAÇÃO

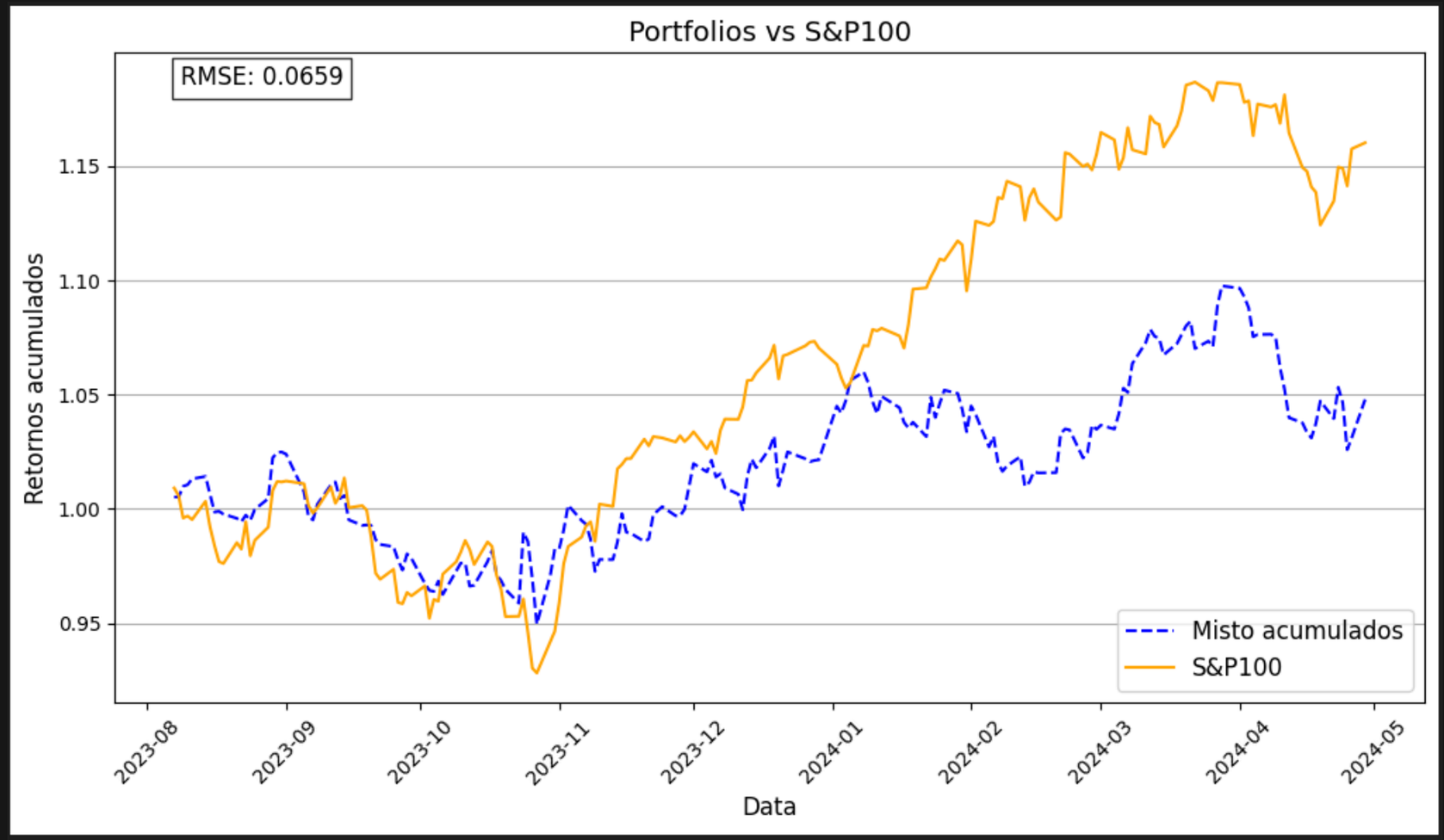
Trazemos aqui então uma breve visualização dos retornos de nossos diversos portfólios comparados aos retornos do S&P100.



MVP

RESULTADOS - VISUALIZAÇÃO

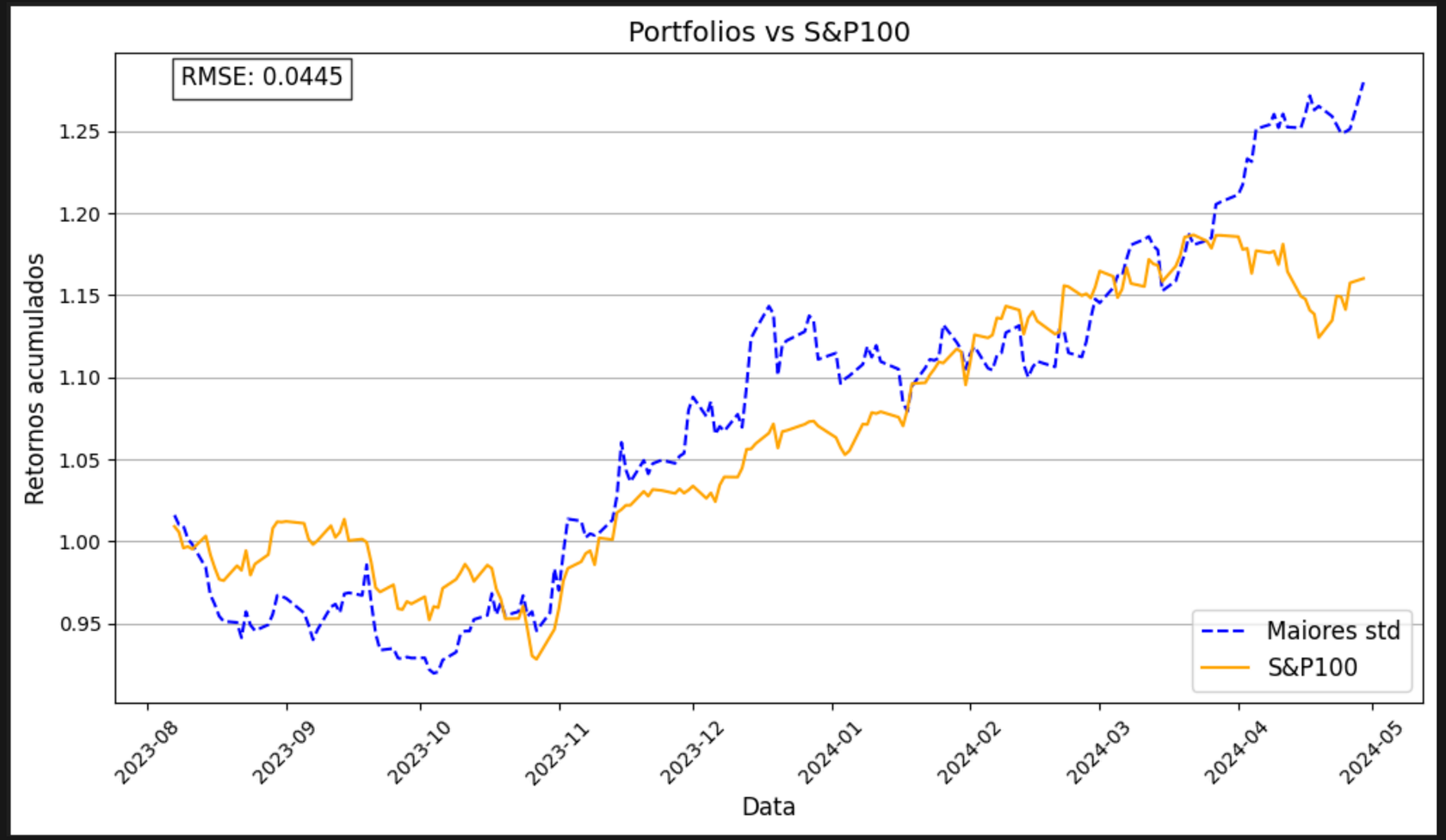
Trazemos aqui então uma breve visualização dos retornos de nossos diversos portfólios comparados aos retornos do S&P100.



MVP

RESULTADOS - VISUALIZAÇÃO

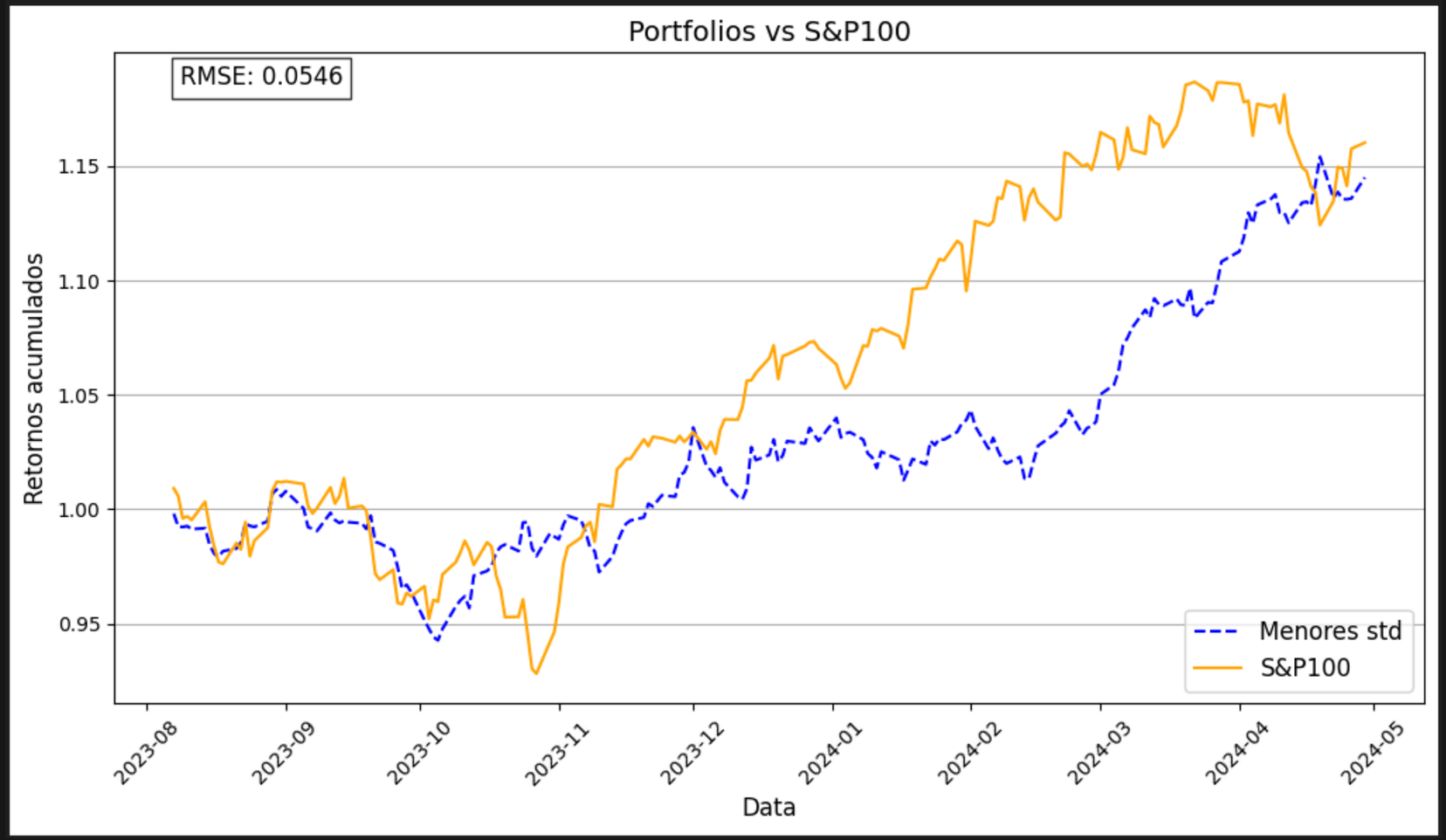
Trazemos aqui então uma breve visualização dos retornos de nossos diversos portfólios comparados aos retornos do S&P100.



MVP

RESULTADOS - VISUALIZAÇÃO

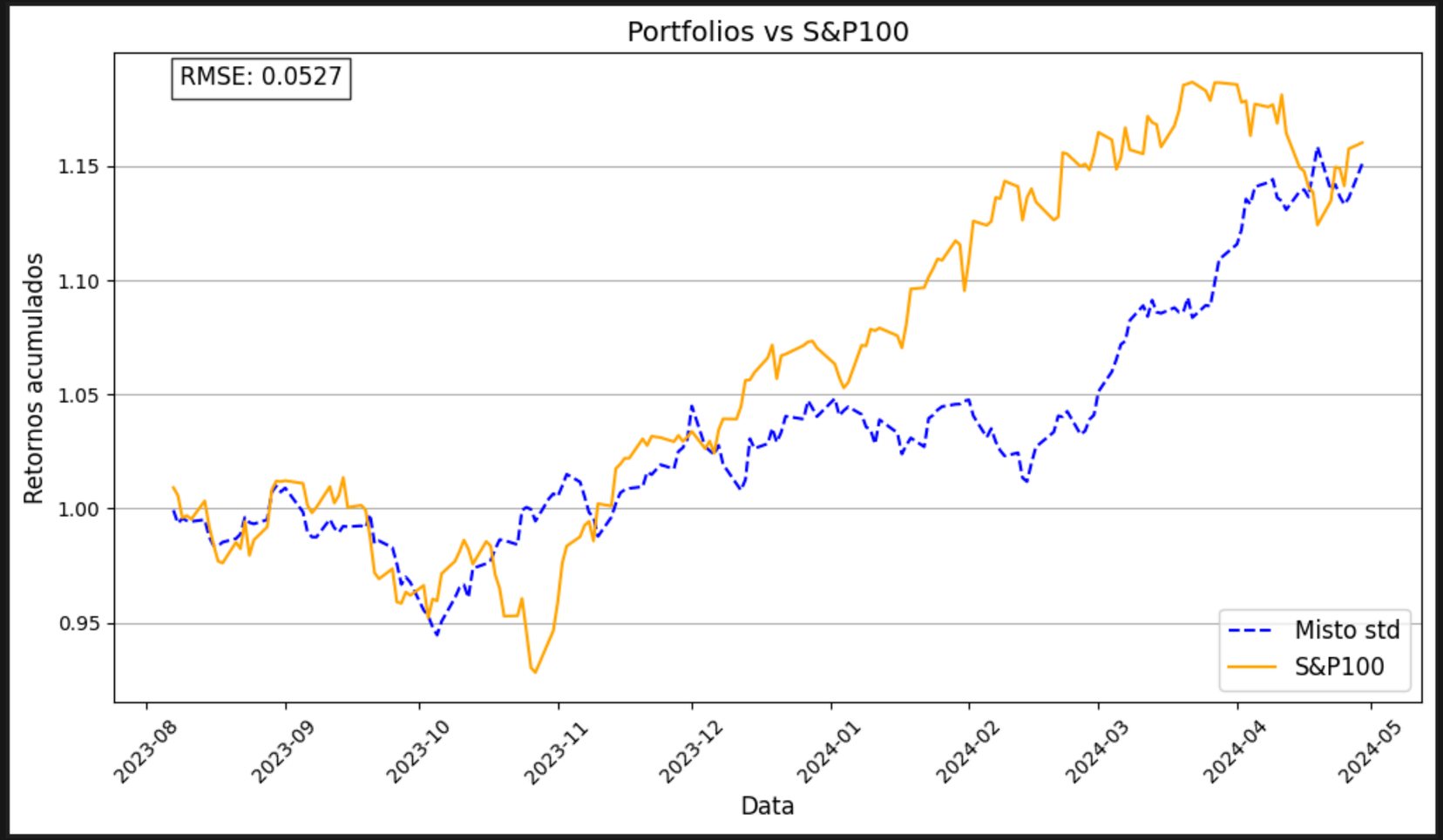
Trazemos aqui então uma breve visualização dos retornos de nossos diversos portfólios comparados aos retornos do S&P100.



MVP

RESULTADOS - VISUALIZAÇÃO

Trazemos aqui então uma breve visualização dos retornos de nossos diversos portfólios comparados aos retornos do S&P100.



MVP

Perspectivas futuras:

- Resolver o problema de minimização para o IBOV;
- Adicionar mais possíveis ativos na lista de candidados ao portfolio;
- Alterar o número máximo de ativos e o tamanho das cestas aleatórias/arbitrárias;
- Implementar um rebalanceamento dos portfolios a cada X períodos, para aumentar sua performance;
- Analisar outra métricas interessantes nos resultados.

MUITO OBRIGADO!