AlbertoGarcia2

November 25, 2025

[ ]:
```
#Instalar librerias
import importlib.util
import sys

def check_and_install_library(library_name):
    spec = importlib.util.find_spec(library_name)
    if spec is None:
        print(f"Library '{library_name}' not found. Installing...")
        try:
            # Use pip to install the library
            # The ! prefix runs shell commands from within Jupyter
            !{sys.executable} -m pip install {library_name}
            print(f"Library '{library_name}' installed successfully.")
        except Exception as e:
            print(f"Error installing '{library_name}': {e}")
    else:
        print(f"Library '{library_name}' is already installed.")

# Replace 'your_library_name' with the actual library you want to check/install
check_and_install_library('pandas')
check_and_install_library('numpy')
check_and_install_library('jupyter')
check_and_install_library('notebook')
check_and_install_library('yfinance')
check_and_install_library('matplotlib.pyplot')
check_and_install_library('json')
#check_and_install_library('ipynb')
check_and_install_library('import_ipynb')
check_and_install_library('datetime')
check_and_install_library('ipywidgets')
check_and_install_library('IPython.display')
check_and_install_library('clear_output')
 #To export to HTML and PDF
check_and_install_library('nbconvert')
check_and_install_library('pandoc')
check_and_install_library('TeX')
```

```
Library 'pandas' is already installed.
```

```
Library 'numpy' is already installed.
Library 'jupyter' is already installed.
Library 'notebook' is already installed.
Library 'yfinance' is already installed.
Library 'matplotlib.pyplot' is already installed.
Library 'json' is already installed.
Library 'import_ipynb' is already installed.
Library 'datetime' is already installed.
Library 'ipywidgets' is already installed.
Library 'IPython.display' is already installed.
Library 'clear_output' not found. Installing…
Library 'clear_output' installed successfully.
Library 'nbconvert' is already installed.
Library 'pandoc' is already installed.
Library 'TeX' not found. Installing…
```

```
ERROR: Could not find a version that satisfies the requirement clear_output
(from versions: none)
ERROR: No matching distribution found for clear_output
```

```
Collecting TeX
  Downloading tex-1.8.tar.gz (4.2 kB)
  Installing build dependencies: started
  Installing build dependencies: finished with status 'done'
  Getting requirements to build wheel: started
  Getting requirements to build wheel: finished with status 'done'
  Preparing metadata (pyproject.toml): started
  Preparing metadata (pyproject.toml): finished with status 'done'
Building wheels for collected packages: TeX
  Building wheel for TeX (pyproject.toml): started
  Building wheel for TeX (pyproject.toml): finished with status 'done'
  Created wheel for TeX: filename=tex-1.8-py3-none-any.whl size=5165
sha256=17bf1924f8fc55398e33031fe1caabda1730ce43872b37ee6a7ce6417411b9f3
  Stored in directory: c:\users\egarcia\appdata\local\pip\cache\wheels\4a\ae\32\
7603f192a1b4bc6ca8ff6386b13df9c949a12cafb76fa63e1c
Successfully built TeX
Installing collected packages: TeX
Successfully installed TeX-1.8
Library 'TeX' installed successfully.
```

1 Libraries

[96]:
```
import yfinance as yf
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
```

```
import ipywidgets as widgets
from IPython.display import display, clear_output

#from datetime import date
from datetime import datetime

import json
#import ipynb


from dataclasses import dataclass

import nbconvert

import import_ipynb
import functions # => .../functions.ipynb     file attached
importlib.reload(functions) # Reloads the module

%reload_ext autoreload
%autoreload 2
```

2 Dates

[3]:
```
#Dates
global start_date
start_date = datetime(2021, 4, 25) # <- date in which brokerage account started
today = datetime.today()
today = today.replace(hour=0, minute=0, second=0, microsecond=0)
no_days = today - start_date
print(f"Start Date: {start_date}, End Date: {today}, number of days {no_days}")
```

```
Start Date: 2021-04-25 00:00:00, End Date: 2025-11-25 00:00:00, number of days
1675 days, 0:00:00
```

3 Tickers

Background: From an initial list of 45 assets (tickers) choose a number greather than 5 shuch that they are diversified among industries, liquidity, heterogeneousity, etc.

[4]:
```
risk_free = 0.04152 #10-year treasury

# Open list of tickers from original Robinhood's file
# file = "Robinhood.xlsx"
file = "App_GBM_Detalle_Portafolio_USA_1763936603841.xlsx"
robinHood_xls = pd.read_excel(file)

tickers = robinHood_xls[["Ticker"]]
```

```
tickers = list(tickers["Ticker"])

no_assets = len(tickers)

print(f"Tickers: {tickers}")
print(f"Number of Assets: {no_assets}")
```

```
Tickers: ['OUST', 'IREN', 'HIMS', 'ASTS', 'BMNR', 'BE', 'AMD']
Number of Assets: 7
```

4 Fundamental Analysis ALL Tickers

TS302_Stock Full Analysis.ipynb

Criteria to choose stocks:

Ratio	Formula	Criteria (great if)	Attribute
P/E Ratio	Current Stock Price / Earnings per Share	between 15 and 25	info.trailingPE
P/B Ratio	Price per Share / Book Value	< 1	info.priceToBook
ROIC (%)	NOPAT / Total Inv. Capital (=Debt+Equity-Assets)	$> 15\%$	functions.get_roic('AAPL')
D/E (%)	Debt / Equity	$< 100\%$ (0%-200%)	info.debtToEquity
EPS (USD)	Net Income / Shares Outstanding	$> 10\%$ CAGR	info.epsForward
ROE Ratio	Net Income / Equity	> 0.15	info.returnOnEquity
EBIT Margin (%)	EBIT / Sales	$> 10\%$	functions.get_ebit_margin("AAPL")
Gross Margin Ratio	Sales - COGS / Sales	> 0.40 (0.35-0.65)	info.grossMargins
Net (%)	Net Income / Revenue	(15%-25%)	functions.get_net_margin("AAPL")
Current Ratio	Current Assets / Current Liabilities	(1.5-2.0)	info.currentRatio

ROIC: Return on Invested Capital

COGS: Cost of Goods Sold

CAGR: Compound Annual Growth Rate

NOPAT: Net Operating Profit After Tax (NOPAT) = Operating Income(or Operating Profit) * (1 - Tax Rate)

Book Value: = (Total Assets - Total Liabilities - Preferred Stock) / Number of Outstanding Common Shares

Overall Risk: Overall assessment including Audit Risk, Board Risk, Compensation Risk, Share Holder Rights Risk. 10 is max, 0 is min. Can be seen individually in .info

yfinance provides:

- hist()
- info
- Income Statement
- Financial Statement

[6]:
```
## Fundamental Analysis for all Tickers based on Financial Ratios

# Call the get_fundamental_analysis() function (...be patient takes ~71sec)
fa_df = functions.get_fundamental_analysis(tickers, start_date, today, showLogs
 ↪= "no")
display(fa_df)  #Result filtered by: ['Sector','P/E Ratio','P/B Ratio']
# sys.stdout = original_stdout
```

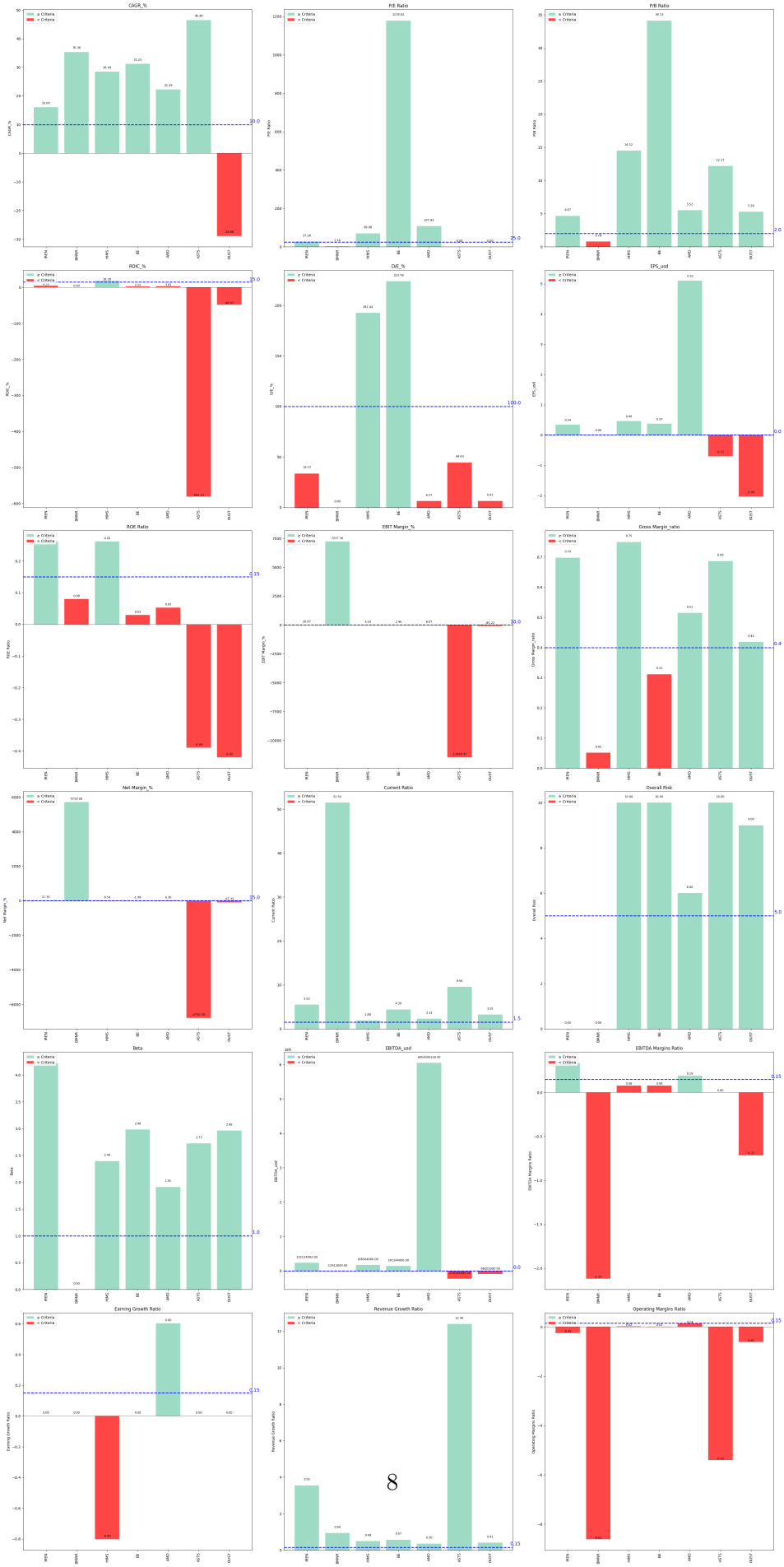
```
                                        Name              Sector  \
Ticker
IREN                             IREN Limited  Financial Services
BMNR    Bitmine Immersion Technologies, Inc.  Financial Services
HIMS                 Hims & Hers Health, Inc.          Healthcare
BE                   Bloom Energy Corporation         Industrials
AMD               Advanced Micro Devices, Inc.          Technology
ASTS                    AST SpaceMobile, Inc.          Technology
OUST                            Ouster, Inc.          Technology

                                        Industry  CAGR_%    P/E Ratio  \
Ticker
IREN                              Capital Markets   16.09    27.281610
BMNR                              Capital Markets   35.36     2.159074
HIMS    Drug Manufacturers - Specialty & Generic   28.49    69.981130
BE                  Electrical Equipment & Parts   31.23  1178.625000
AMD                               Semiconductors   22.29   107.921470
ASTS                     Communication Equipment   46.48     0.000000
OUST                       Electronic Components  -28.88     0.000000

        P/B Ratio  ROIC_%    D/E_%  EPS_usd  ROE Ratio  …  \
Ticker                                                  …
IREN     4.668568    4.39   33.574     0.34    0.26125  …
BMNR     0.780592    0.00    0.000     0.00    0.08016  …
HIMS    14.522319   18.28  192.437     0.46    0.26207  …
BE      34.125950    2.22  223.775     0.37    0.02930  …
AMD      5.520354    3.01    6.366     5.10    0.05317  …
ASTS    12.170577 -581.13   44.433    -0.71   -0.39031  …
OUST     5.277442  -48.07    6.425    -2.04   -0.42033  …

        Gross Margin_ratio  Net Margin_%  Current Ratio  Overall Risk   Beta  \
Ticker
IREN               0.69820         17.35          5.516           0.0  4.220
```

```
BMNR              0.05086       5719.06         51.502           0.0  0.000
HIMS              0.75012          8.54          1.875          10.0  2.397
BE                0.31166         -1.98          4.395          10.0  2.985
AMD               0.51457          6.36          2.308           6.0  1.913
ASTS              0.68685      -6792.28          9.560          10.0  2.725
OUST              0.41856        -87.35          3.255           9.0  2.964

        EBITDA_usd  EBITDA Margins Ratio  Earning Growth Ratio  \
Ticker
IREN     232124992              0.33712                 0.000
BMNR     -12913000             -2.11862                 0.000
HIMS     168504000              0.07621                -0.803
BE       141244000              0.07766                 0.000
AMD     6054000128              0.18903                 0.603
ASTS    -230306000              0.00000                 0.000
OUST     -98201000             -0.71524                 0.000

        Revenue Growth Ratio  Operating Margins Ratio
Ticker
IREN                   3.554                -0.25025
BMNR                   0.942                -8.61545
HIMS                   0.492                 0.02060
BE                     0.571                 0.01512
AMD                    0.356                 0.13736
ASTS                  12.399                -5.40579
OUST                   0.408                -0.61356

[7 rows x 21 columns]
```

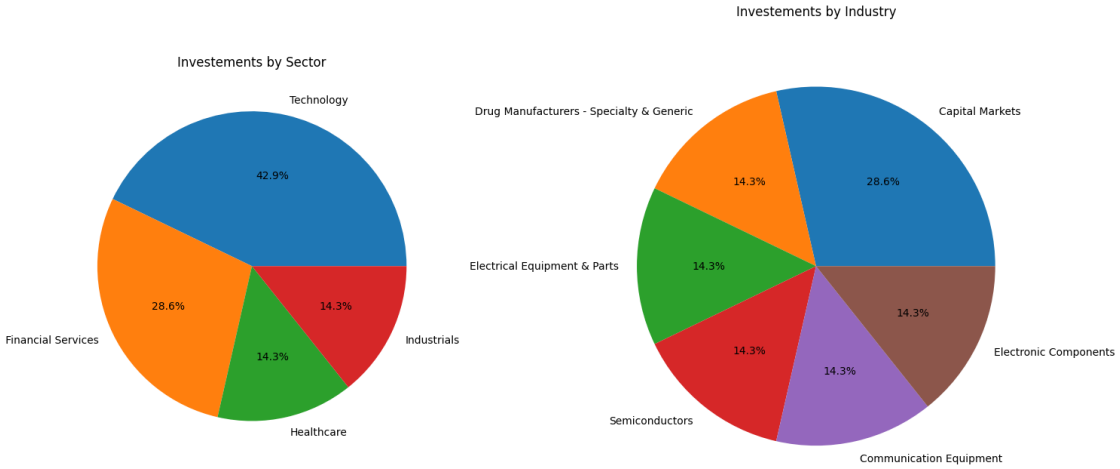
[7]:
```python
ratio_criteria= {
    'CAGR_%': 10.0, # %
    'P/E Ratio': 25.0, # ratio
    'P/B Ratio': 2.0, #ratio
    'ROIC_%': 15.0, # %
    'D/E_%': 100.0, # %
    'EPS_usd': 0.0, # USD. Must be ~>10% of CAGR. To-DO: Make this automatic
    'ROE Ratio': 0.15, # ratio
    'EBIT Margin_%': 10.0, # %
    'Gross Margin_ratio': 0.40, # ratio
    'Net Margin_%': 15.0, # %
    'Current Ratio': 1.5, # ratio
    'Overall Risk': 5.0, # 10 is max, 0 is min.
    'Beta': 1.0,
    'EBITDA_usd': 0.00, # USD
    'EBITDA Margins Ratio': 0.15, # ratio - confirm
    'Earning Growth Ratio': 0.15, # ratio - confirm
    'Revenue Growth Ratio': 0.15, # ratio
```

```
    'Operating Margins Ratio': 0.15 # ratio
}
```

[8]:
```
# Plot financial Ratios
functions.plot_ratios(fa_df, ratio_criteria, num_cols = 3)
```


CAGR_%
P/E Ratio
P/B Ratio
ROIC_%
D/E_%
EPS_usd
ROE Ratio
EBIT Margin_%
Gross Margin_ratio
Net Margin_%
Current Ratio
Overall Risk
Beta
EBITDA_usd
EBITDA Margins Ratio
Earning Growth Ratio
Revenue Growth Ratio
Operating Margins Ratio
≥ Criteria
< Criteria
IFEN
BMWK
HMIS
BE
AMSC
ASTS
OUST

[9]:
```
# Sectors and Industries
functions.print_sector_industry(fa_df)
```



5 **TO-DO

- ☒ ~~1. Show pie charts for categorical variables~~
- ☒ ~~2. show plots for other variables~~
- ☐ 3. Automate the Stock Selection (top 5-10), for now pick manually the best in each cateogry

6 Financial Analysis (ONE) Ticker

TS302_Stock Full Analysis.ipynb

This section is to see in more detail any particular ticker

[10]:
```
# 1. Create the widget
ticker_dropdown = widgets.Dropdown(
    options=tickers,
    #options=["KOF", "AAPL", "MSFT", "MA","NVDA", "GOOGL", "AMZN", "META",␣
 ↪"TSM","BRK-B", "V", "JPM", "XOM", "LLY", "MRK", "UNH", "PG", "MA","CVX",␣
 ↪"KO", "PEP", "COST", "TMO", "ORCL", "CSCO", "NKE", "VZ", "ASML", "TXN",␣
 ↪"ABT", "TM", "SAP", "AMD", "NFLX", "NOW", "ADBE", "LVMUY", "BABA", "SHEL",␣
 ↪"TMUS", "QCOM", "PFE", "SNY", "AZN", "TOT", "GSK", "RIO", "BHP", "MCD"],
    #options=["BTC-USD", "ETH-USD", "USDT-USD", "XRP-USD", "LTC-USD",␣
 ↪"ADA-USD", "DOT-USD", "BCH-USD", "XLM-USD", "LINK-USD"]
```

```
    value=tickers[0],  # Default selected value (must be from the options)
    description='Select Ticker:',
    disabled=False,
)

def on_change(selected_ticker):
    clear_output(wait=False)
    display(fa_df.loc[selected_ticker])

    global ticker_widget
    ticker_widget = selected_ticker

    return ticker_widget

# # 4. Link the function to the widget and capture the output
interactive_plot = widgets.interactive_output(on_change, {'selected_ticker':␣
 ↪ticker_dropdown})

# # 5. Display the widget and the output area in your notebook cell
display(ticker_dropdown, interactive_plot)
```

```
Dropdown(description='Select Ticker:', options=('OUST', 'IREN', 'HIMS', 'ASTS',␣
 ↪'BMNR', 'BE', 'AMD'), value='O…
```

```
Output()
```

[11]:
```
# Choose any ticket from the list above
ticker = yf.Ticker(ticker_widget)
ticker_name = ticker.info.get('symbol')
print(ticker_name)
```

```
OUST
```

6.1 History

[12]:
```
# Example of history() of any ONE ticker for "1y"
hist = ticker.history(period="1y", auto_adjust=True)
print(ticker_name)
display(hist)
```

```
OUST
```

Date	Open	High	Low	Close	Volume
2024-11-26 00:00:00-05:00	9.780	9.870000	9.280000	9.360000	1007700
2024-11-27 00:00:00-05:00	9.430	10.150000	9.430000	9.550000	950900
2024-11-29 00:00:00-05:00	9.700	10.250000	9.690000	9.880000	683000
2024-12-02 00:00:00-05:00	10.090	10.359000	9.570000	9.620000	1329000
2024-12-03 00:00:00-05:00	9.470	9.620000	8.510000	9.100000	1443600

```
…                                 …         …         …         …         …
2025-11-19 00:00:00-05:00  21.020  22.240000  20.290001  20.950001  1812700
2025-11-20 00:00:00-05:00  22.375  22.490000  19.670000  19.719999  2753400
2025-11-21 00:00:00-05:00  19.850  20.410000  18.520000  19.930000  3086500
2025-11-24 00:00:00-05:00  20.410  21.430000  20.139999  21.370001  1958900
2025-11-25 00:00:00-05:00  21.250  21.790001  20.290001  21.780001  1398771

                           Dividends  Stock Splits
Date
2024-11-26 00:00:00-05:00        0.0           0.0
2024-11-27 00:00:00-05:00        0.0           0.0
2024-11-29 00:00:00-05:00        0.0           0.0
2024-12-02 00:00:00-05:00        0.0           0.0
2024-12-03 00:00:00-05:00        0.0           0.0
…                                  …             …
2025-11-19 00:00:00-05:00        0.0           0.0
2025-11-20 00:00:00-05:00        0.0           0.0
2025-11-21 00:00:00-05:00        0.0           0.0
2025-11-24 00:00:00-05:00        0.0           0.0
2025-11-25 00:00:00-05:00        0.0           0.0

[250 rows x 7 columns]
```

- Info
- Income Statement
- Balance Sheet

[13]:
```python
# info
ticker_info = ticker.info
# Optional: Print in JSON format all info
#print(json.dumps(ticker_info, indent=4))

#Income Statement
ticker_income_stmt = ticker.income_stmt
# Optional: Print Income Statement
#print(ticker_income_stmt)

#Balance Sheet
ticker_balance_sheet = ticker.balance_sheet
# Optional: Print Balance Sheet
#print(ticker_balance_sheet)


# To-Do: Support the Stock selection based on Ratios using the Financial␣
 ↪Statements
```

[14]:
```python
# Print info by category (some selected data only)
```

```python
def print_info_by_category(info_list, name):
    print(f"\n{name}")
    for key in info_list:
        try:
            value = ticker_info.get(key, 'N/A')
            print(f"{key}: {value}")
        except Exception as e:
            print(f"Error retrieving '{key} for {ticker_name}: {e}")

# One can choose which info parameters to print in each category:
basic_info = ['symbol', 'longName', 'sector', 'industry', 'country']
market_info = ['currentPrice', 'marketCap', 'volume', '52WeekChange',␣
 ↪'fiftyTwoWeekHigh', 'fiftyTwoWeekLow']
financial_info = ['priceToBook', 'forwardPE', 'trailingPE', 'profitMargins',␣
 ↪'totalRevenue', 'debtToEquity',
                  ␣
 ↪'epsForward','ebitda','floatShares','forwardEps','grossMargins','grossProfits','operatingCa
                  ␣
 ↪'operatingMargins','returnOnAssets','returnOnEquity','revenueGrowth','revenuePerShare','imp
                   'totalCash','totalDebt']
dividends_info = ['dividendYield','payoutRatio','dividendRate']
shares_info = ['heldPercentInsiders','heldPercentInstitutions']
technical_info = ['sharesOutstanding','beta','currency']

print_info_by_category(basic_info,      "1. Basic info:")
print_info_by_category(market_info,     "2. Market info:")
print_info_by_category(financial_info,  "3. Financial info:")
print_info_by_category(dividends_info,  "4. Dividends info:")
print_info_by_category(shares_info,     "5. Shares management info:")
print_info_by_category(technical_info,  "6. Technical info:")

# Market Cap = Current Share Price × Shares Outstanding
```

```
1. Basic info:
symbol: OUST
longName: Ouster, Inc.
sector: Technology
industry: Electronic Components
country: United States

2. Market info:
currentPrice: 21.78
marketCap: 1306913792
volume: 1398771
52WeekChange: 1.2831197
fiftyTwoWeekHigh: 41.65
```

```
fiftyTwoWeekLow: 6.34

3. Financial info:
priceToBook: 5.2774415
forwardPE: -10.676471
trailingPE: N/A
profitMargins: -0.64166
totalRevenue: 137298000
debtToEquity: 6.425
epsForward: -2.04
ebitda: -98201000
floatShares: 57272581
forwardEps: -2.04
grossMargins: 0.41856
grossProfits: 57467000
operatingCashflow: -27091000
operatingMargins: -0.61355996
returnOnAssets: -0.21736999
returnOnEquity: -0.42033002
revenueGrowth: 0.408
revenuePerShare: 2.549
impliedSharesOutstanding: 60005219
totalCash: 244518000
totalDebt: 15899000

4. Dividends info:
dividendYield: N/A
payoutRatio: 0.0
dividendRate: N/A

5. Shares management info:
heldPercentInsiders: 0.04538
heldPercentInstitutions: 0.56553

6. Technical info:
sharesOutstanding: 60005219
beta: 2.964
currency: USD
```

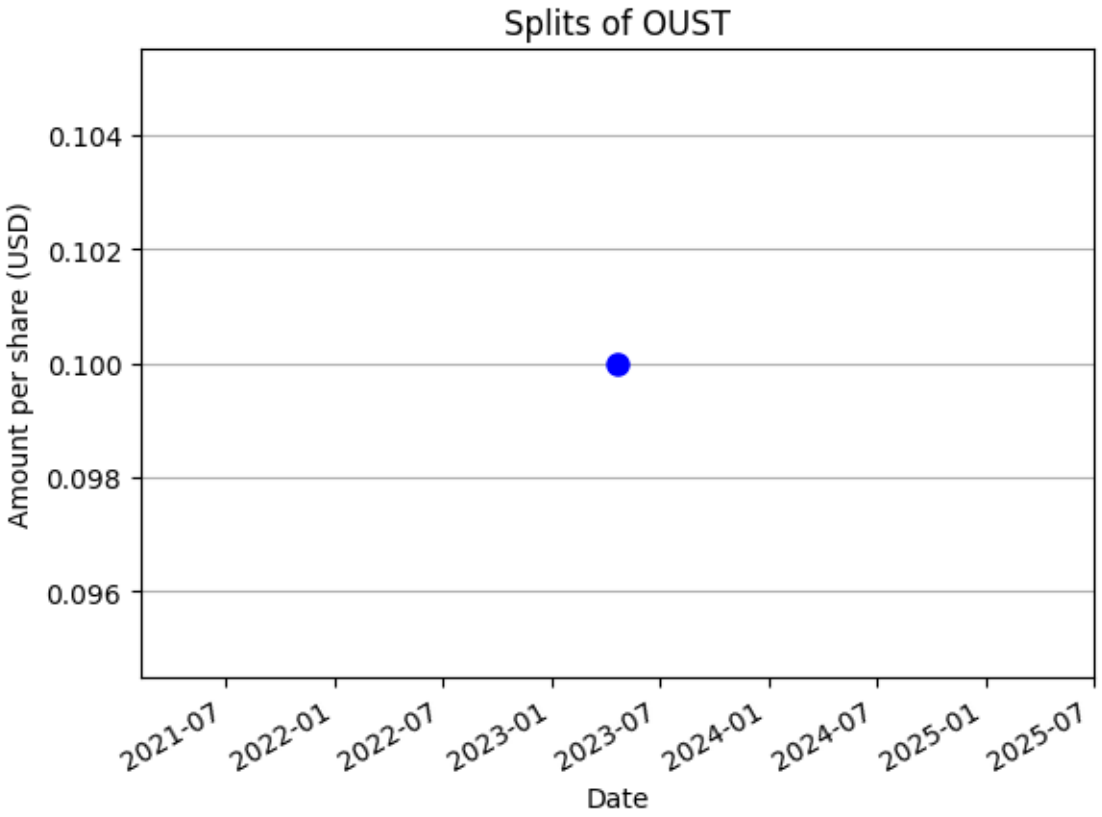
TO-DO: Monitoreao Recompra o dilucion de acciones. Con base en el numero de acciones disponibles por anio

6.1.1 Dividends, Splits and Recommendations

[15]:
```
# 7. Other info:
print("\n7. Other info:")
other_info = {'Dividends' : ticker.dividends,
              'Splits' : ticker.splits,
```

```
              'Recommendations' : ticker.recommendations
            #   'Recommendations Summary' : ticker.recommendations_summary
              }

functions.print_dividends_splits_recommendations(other_info, ticker_name)
```

7. Other info:
There are not Dividends in this period for 'OUST'
Splits:



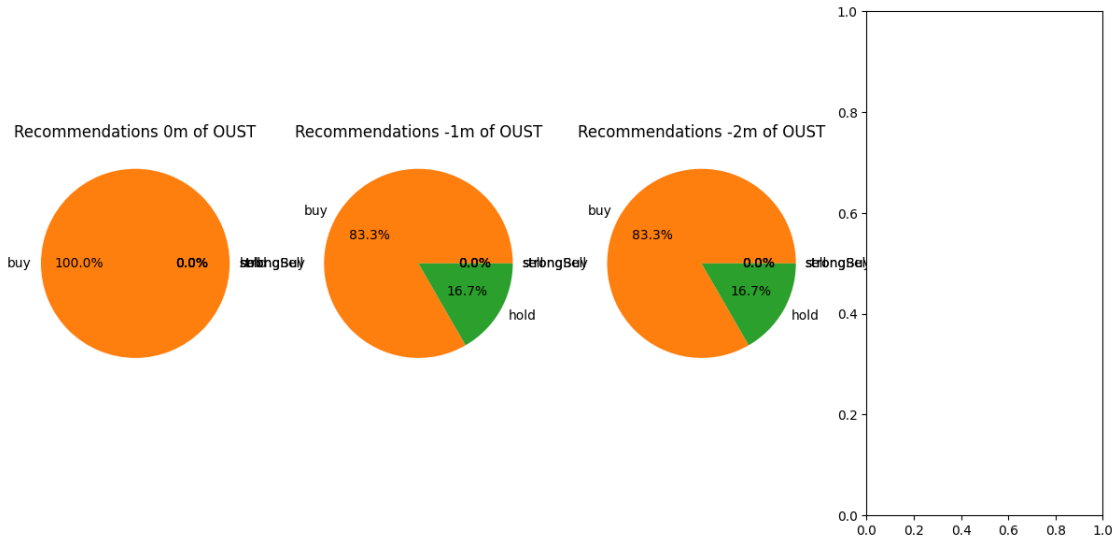
Splits:

```
Date
2023-04-21 00:00:00-04:00    0.1
Name: Stock Splits, dtype: float64
```

Recommendations:

	period	strongBuy	buy	hold	sell	strongSell
0	0m	0	6	0	0	0
1	-1m	0	5	1	0	0
2	-2m	0	5	1	0	0

```
Error retrieving 'Recommendations for OUST: single positional indexer is out-of-
bounds
```



[16]:
```python
import operator
from dataclasses import dataclass
from typing import Callable, Any, Optional

RetrievalFunc = Callable[[dict], Any]   #ticker.info : is a dictionary: ticker.
 ↪info.get('trailingPE', 'N/A'). Look for ticker_info =  ticker.info above
CompareFunc = Callable[[Any, Any], bool]

@dataclass
class FinancialRule:
    retrieval_func: RetrievalFunc   # function used to extract the specific␣
 ↪metric from the data source
    metric_name: str
    target_value: Optional[Any] = None
    comparison_func: Optional[Callable[[Any, Any], bool]] = None
          # for printing pruposes
    # target_value: float             # The target value to compare against
    # comparison_func: CompareFunc    # comparison operator function (e.g.,␣
 ↪operator.lt for <)


# financial_rules = {
#     'D/E_%': FinancialRule(retrieval_func=lambda ticker_info: ticker_info.
 ↪get('debtToEquity', 'N/A'), target_value=100.0, comparison_func=operator.gt,␣
 ↪metric_name='D/E_%')
# }
```

```
financial_rules = {
    'Name':                     FinancialRule(retrieval_func=lambda ticker_info:
 ↪ ticker_info.get('longName', 'N/A'),           metric_name='Name'),
    'Sector':                   FinancialRule(retrieval_func=lambda ticker_info:
 ↪ ticker_info.get('sector', 'ETF, others'),     metric_name='Sector'),
    'Industry':                 FinancialRule(retrieval_func=lambda ticker_info:
 ↪ ticker_info.get('industry', 'ETF, others'),   metric_name='Industry'),
    'CAGR_%':                   FinancialRule(retrieval_func=lambda ticker_info:
 ↪ functions.get_cagr(ticker_info.get('symbol'), start_date, today),  ␣
 ↪target_value=10.0, comparison_func=operator.ge, metric_name='CAGR_%'),
    'P/E Ratio':                FinancialRule(retrieval_func=lambda ticker_info:
 ↪ ticker_info.get('trailingPE', 'N/A'),         target_value=25.0,␣
 ↪comparison_func=operator.lt, metric_name='P/E Ratio'),
    'P/B Ratio':                FinancialRule(retrieval_func=lambda ticker_info:
 ↪ ticker_info.get('priceToBook', 'N/A'),        target_value=2.0,␣
 ↪comparison_func=operator.lt, metric_name='P/B Ratio'),
    'ROIC_%':                   FinancialRule(retrieval_func=lambda ticker_info:
 ↪ functions.get_roic(ticker_info.get('symbol')), target_value=15.0,␣
 ↪comparison_func=operator.gt, metric_name='ROIC_%'),
    'D/E_%':                    FinancialRule(retrieval_func=lambda ticker_info:
 ↪ ticker_info.get('debtToEquity', 'N/A'),       target_value=100.0,␣
 ↪comparison_func=operator.lt, metric_name='D/E_%'),
    'EPS_usd':                  FinancialRule(retrieval_func=lambda ticker_info:
 ↪ ticker_info.get('epsForward', 'N/A'),         target_value=0.0,␣
 ↪comparison_func=operator.gt, metric_name='EPS_usd'),
    'ROE Ratio':                FinancialRule(retrieval_func=lambda ticker_info:
 ↪ ticker_info.get('returnOnEquity', 'N/A'),     target_value=0.15,␣
 ↪comparison_func=operator.ge, metric_name='ROE Ratio'),
    'EBIT Margin_%':            FinancialRule(retrieval_func=lambda ticker_info:
 ↪ functions.get_ebit_margin(ticker_info.get('symbol')),  target_value=10.0,␣
 ↪comparison_func=operator.ge, metric_name='EBIT Margin_%'),
    'Gross Margin_ratio':       FinancialRule(retrieval_func=lambda ticker_info:
 ↪ ticker_info.get('grossMargins', 'N/A'),       target_value=0.40,␣
 ↪comparison_func=operator.ge, metric_name='Gross Margin_ratio'),
    'Net Margin_%':             FinancialRule(retrieval_func=lambda ticker_info:
 ↪ functions.get_net_margin(ticker_info.get('symbol')),   target_value=15.0,␣
 ↪comparison_func=operator.ge, metric_name='Net Margin_%'),
    'Current Ratio':            FinancialRule(retrieval_func=lambda ticker_info:
 ↪ ticker_info.get('currentRatio', 'N/A'),       target_value=1.5,␣
 ↪comparison_func=operator.ge, metric_name='Current Ratio'),
    'Overall Risk':             FinancialRule(retrieval_func=lambda ticker_info:
 ↪ ticker_info.get('overallRisk', 'N/A'),        target_value=5.0,␣
 ↪comparison_func=operator.lt, metric_name='Overall Risk'),
```

```
    'Beta':                    FinancialRule(retrieval_func=lambda ticker_info:
 ↪ ticker_info.get('beta', 'N/A'),              target_value=1.0,␣
 ↪comparison_func=operator.eq, metric_name='Beta'),
    'EBITDA_usd':              FinancialRule(retrieval_func=lambda ticker_info:
 ↪ ticker_info.get('ebitda', 'N/A'),            target_value=0.0,␣
 ↪comparison_func=operator.gt, metric_name='EBITDA_usd'),
    'EBITDA Margins Ratio':    FinancialRule(retrieval_func=lambda ticker_info:
 ↪ ticker_info.get('ebitdaMargins', 'N/A'),     target_value=0.15,␣
 ↪comparison_func=operator.gt, metric_name='EBITDA Margins Ratio'),
    'Earning Growth Ratio':    FinancialRule(retrieval_func=lambda ticker_info:
 ↪ ticker_info.get('earningsGrowth', 'N/A'),    target_value=0.15,␣
 ↪comparison_func=operator.gt, metric_name='Earning Growth Ratio'),
    'Revenue Growth Ratio':    FinancialRule(retrieval_func=lambda ticker_info:
 ↪ ticker_info.get('revenueGrowth', 'N/A'),     target_value=0.15,␣
 ↪comparison_func=operator.gt, metric_name='Revenue Growth Ratio'),
    'Operating Margins Ratio': FinancialRule(retrieval_func=lambda ticker_info:
 ↪ ticker_info.get('operatingMargins', 'N/A'),  target_value=0.15,␣
 ↪comparison_func=operator.gt, metric_name='Operating Margins Ratio'),
}

def evaluate_ticker_rules(financial_rules: dict, ticker_info: dict):
    print(f"---Evaluating Rules for {ticker_info.get('symbol', 'Unknown␣
 ↪Ticker')}---")

    if not ticker_info.get('symbol'):
        print("Status:  SKIP (No ticker data available)")
        return

    for rule_name, rule in financial_rules.items():
        # 1. Call the stored retrieval function to get the actual metric value
        actual_value = rule.retrieval_func(ticker_info)

        print(f"\nRule: {rule_name}")
        print(f"Metric: {rule.metric_name}, Value found: {actual_value}")


        if rule.comparison_func is not None:
            if actual_value is None or actual_value == 'N/A':
                print("Status:  SKIP (Data for comparison not available)")
                continue

            # 2. Call the stored comparison function
            is_pass = rule.comparison_func(actual_value, rule.target_value)

            if is_pass:
                print(f"Status:  PASS ({actual_value} is acceptable)")
```

```
            else:
                print(f"Status:  FAIL ({actual_value} fails rule to be {rule.
 ↪comparison_func.__name__} {rule.target_value})")

        else:
            print("Status:  INFO (No comparison needed)")

evaluate_ticker_rules(financial_rules, ticker_info)
```

```
---Evaluating Rules for OUST---

Rule: Name
Metric: Name, Value found: Ouster, Inc.
Status:  INFO (No comparison needed)

Rule: Sector
Metric: Sector, Value found: Technology
Status:  INFO (No comparison needed)

Rule: Industry
Metric: Industry, Value found: Electronic Components
Status:  INFO (No comparison needed)

Rule: CAGR_%
Metric: CAGR_%, Value found: -28.88
Status:  FAIL (-28.88 fails rule to be ge 10.0)

Rule: P/E Ratio
Metric: P/E Ratio, Value found: N/A
Status:  SKIP (Data for comparison not available)

Rule: P/B Ratio
Metric: P/B Ratio, Value found: 5.2774415
Status:  FAIL (5.2774415 fails rule to be lt 2.0)

Rule: ROIC_%
Metric: ROIC_%, Value found: -48.07
Status:  FAIL (-48.07 fails rule to be gt 15.0)

Rule: D/E_%
Metric: D/E_%, Value found: 6.425
Status:  PASS (6.425 is acceptable)

Rule: EPS_usd
Metric: EPS_usd, Value found: -2.04
Status:  FAIL (-2.04 fails rule to be gt 0.0)

Rule: ROE Ratio
```

```
Metric: ROE Ratio, Value found: -0.42033002
Status:   FAIL (-0.42033002 fails rule to be ge 0.15)

Rule: EBIT Margin_%
Metric: EBIT Margin_%, Value found: -85.22
Status:   FAIL (-85.22 fails rule to be ge 10.0)

Rule: Gross Margin_ratio
Metric: Gross Margin_ratio, Value found: 0.41856
Status:   PASS (0.41856 is acceptable)

Rule: Net Margin_%
Metric: Net Margin_%, Value found: -87.35
Status:   FAIL (-87.35 fails rule to be ge 15.0)

Rule: Current Ratio
Metric: Current Ratio, Value found: 3.255
Status:   PASS (3.255 is acceptable)

Rule: Overall Risk
Metric: Overall Risk, Value found: 9
Status:   FAIL (9 fails rule to be lt 5.0)

Rule: Beta
Metric: Beta, Value found: 2.964
Status:   FAIL (2.964 fails rule to be eq 1.0)

Rule: EBITDA_usd
Metric: EBITDA_usd, Value found: -98201000
Status:   FAIL (-98201000 fails rule to be gt 0.0)

Rule: EBITDA Margins Ratio
Metric: EBITDA Margins Ratio, Value found: -0.71524
Status:   FAIL (-0.71524 fails rule to be gt 0.15)

Rule: Earning Growth Ratio
Metric: Earning Growth Ratio, Value found: N/A
Status:   SKIP (Data for comparison not available)

Rule: Revenue Growth Ratio
Metric: Revenue Growth Ratio, Value found: 0.408
Status:   PASS (0.408 is acceptable)

Rule: Operating Margins Ratio
Metric: Operating Margins Ratio, Value found: -0.61355996
Status:   FAIL (-0.61355996 fails rule to be gt 0.15)
```

7 Import Prices from yfinance (All Tickers)

TS302_Stock Full Analysis.ipynb

[17]:
```
# import data from yahoo Finance
print(f"Number of days since Brokerage account was opened: {no_days}")
df = yf.download(tickers, start=start_date, end=today, auto_adjust=True)
df.info()
```

```
[                       0%                       ]
Number of days since Brokerage account was opened: 1675 days, 0:00:00
[**********************100%**********************]  7 of 7 completed
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 1153 entries, 2021-04-26 to 2025-11-24
Data columns (total 35 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   (Close, AMD)     1153 non-null   float64
 1   (Close, ASTS)    1153 non-null   float64
 2   (Close, BE)      1153 non-null   float64
 3   (Close, BMNR)    120 non-null    float64
 4   (Close, HIMS)    1153 non-null   float64
 5   (Close, IREN)    1009 non-null   float64
 6   (Close, OUST)    1153 non-null   float64
 7   (High, AMD)      1153 non-null   float64
 8   (High, ASTS)     1153 non-null   float64
 9   (High, BE)       1153 non-null   float64
 10  (High, BMNR)     120 non-null    float64
 11  (High, HIMS)     1153 non-null   float64
 12  (High, IREN)     1009 non-null   float64
 13  (High, OUST)     1153 non-null   float64
 14  (Low, AMD)       1153 non-null   float64
 15  (Low, ASTS)      1153 non-null   float64
 16  (Low, BE)        1153 non-null   float64
 17  (Low, BMNR)      120 non-null    float64
 18  (Low, HIMS)      1153 non-null   float64
 19  (Low, IREN)      1009 non-null   float64
 20  (Low, OUST)      1153 non-null   float64
 21  (Open, AMD)      1153 non-null   float64
 22  (Open, ASTS)     1153 non-null   float64
 23  (Open, BE)       1153 non-null   float64
 24  (Open, BMNR)     120 non-null    float64
 25  (Open, HIMS)     1153 non-null   float64
 26  (Open, IREN)     1009 non-null   float64
 27  (Open, OUST)     1153 non-null   float64
 28  (Volume, AMD)    1153 non-null   int64
 29  (Volume, ASTS)   1153 non-null   int64
```

```
 30  (Volume, BE)    1153 non-null   int64
 31  (Volume, BMNR)  120 non-null    float64
 32  (Volume, HIMS)  1153 non-null   int64
 33  (Volume, IREN)  1009 non-null   float64
 34  (Volume, OUST)  1153 non-null   int64
dtypes: float64(30), int64(5)
memory usage: 324.3 KB
```

7.1 Prices (Close)

[18]:
```
prices_raw = df["Close"]
display(prices_raw.tail(5))
```

```
Ticker             AMD       ASTS          BE       BMNR       HIMS  \
Date
2025-11-18  230.289993  58.220001  104.970001  32.279999  36.259998
2025-11-19  223.550003  58.009998  108.930000  29.180000  35.830002
2025-11-20  206.020004  50.700001   93.379997  26.020000  33.619999
2025-11-21  203.779999  51.369999   89.989998  26.000000  34.709999
2025-11-24  215.050003  55.000000   95.559998  31.100000  37.779999

Ticker           IREN       OUST
Date
2025-11-18  48.849998  20.980000
2025-11-19  45.830002  20.950001
2025-11-20  43.470001  19.719999
2025-11-21  42.259998  19.930000
2025-11-24  48.490002  21.370001
```

[19]:
```
# drop NaN
prices = prices_raw.dropna()
display(prices.tail(5))
```

```
Ticker             AMD       ASTS          BE       BMNR       HIMS  \
Date
2025-11-18  230.289993  58.220001  104.970001  32.279999  36.259998
2025-11-19  223.550003  58.009998  108.930000  29.180000  35.830002
2025-11-20  206.020004  50.700001   93.379997  26.020000  33.619999
2025-11-21  203.779999  51.369999   89.989998  26.000000  34.709999
2025-11-24  215.050003  55.000000   95.559998  31.100000  37.779999

Ticker           IREN       OUST
Date
2025-11-18  48.849998  20.980000
2025-11-19  45.830002  20.950001
2025-11-20  43.470001  19.719999
2025-11-21  42.259998  19.930000
```

```
2025-11-24  48.490002  21.370001
```

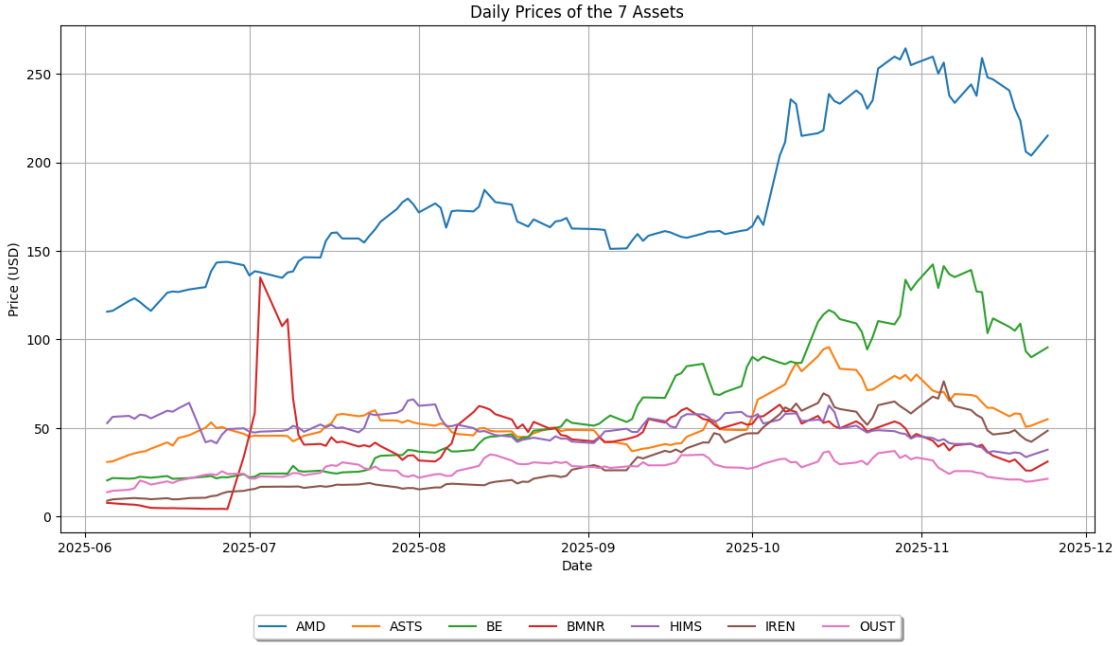
[20]:
```
print(f"prices_raw shape: {prices_raw.shape}")
print(f"prices shape: {prices.shape}")

if len(prices) < len(prices_raw):
    print(f"Reduction of {(len(prices_raw) - len(prices))}")
    print(f"It can be noticed that the amount of rows in the dataframe dropped␣
 ↪from {len(prices_raw)} to {len(prices)} after the drop.na() operation. This␣
 ↪is because there are a couple of assets that are relatively new in the␣
 ↪public market (post IPO).")
```

```
prices_raw shape: (1153, 7)
prices shape: (120, 7)
Reduction of 1033
It can be noticed that the amount of rows in the dataframe dropped from 1153 to
120 after the drop.na() operation. This is because there are a couple of assets
that are relatively new in the public market (post IPO).
```

[21]:
```
print(f"Reminder: the start_date was: {start_date.date()}, when the original␣
 ↪portfolio was created.")

print("\nOldest available dates for each asset:")

df_ = df['Close']
first_valid_dates = df_.apply(pd.Series.first_valid_index)
first_valid_dates.name = "Oldest Date"
first_valid_dates = first_valid_dates.sort_values(ascending=False)

display(first_valid_dates)

print(f"The asset with the newest available data is: '{first_valid_dates.
 ↪index[0]}' from {first_valid_dates.iloc[0].date()} only.")
```

```
Reminder: the start_date was: 2021-04-25, when the original portfolio was
created.

Oldest available dates for each asset:

Ticker
BMNR   2025-06-05
IREN   2021-11-17
AMD    2021-04-26
ASTS   2021-04-26
BE     2021-04-26
HIMS   2021-04-26
OUST   2021-04-26
Name: Oldest Date, dtype: datetime64[ns]

The asset with the newest available data is: 'BMNR' from 2025-06-05 only.
```

[22]:
```
# Plot all daily prices series in a single plot
plt.figure(figsize=(12, 7))
for col in prices.columns:
    plt.plot(prices.index, prices[col], label=col)

plt.title(f"Daily Prices of the {no_assets} Assets")
plt.ylabel("Price (USD)")
plt.xlabel("Date")
plt.legend(loc='upper center',
           bbox_to_anchor=(0.5, -0.15),
           ncol=10,
           fancybox=True,
           shadow=True)
plt.grid(True)
plt.tight_layout()
plt.show()
```



7.1.1 stats

[23]:
```
# statistics about prices
prices.describe()
```

[23]:

Ticker	AMD	ASTS	BE	BMNR	HIMS	\
count	120.000000	120.000000	120.000000	120.000000	120.000000	
mean	179.409917	55.130500	63.711833	42.974167	50.118917	
std	41.420354	14.508423	37.404773	20.468127	7.203238	

```
min     115.690002   30.850000   20.450001    4.265000   33.619999
25%     155.654999   45.550001   25.952499   34.992500   44.967499
50%     164.339996   50.525000   52.695000   44.830000   49.914999
75%     215.392502   62.202498   91.062500   53.137501   56.050000
max     264.329987   95.690002  142.369995  135.000000   66.180000

Ticker        IREN        OUST
count   120.000000  120.000000
mean     33.239708   27.023500
std      19.500038    4.973819
min       8.950000   13.760000
25%      16.935000   23.369999
50%      26.139999   27.845000
75%      48.700001   30.620000
max      76.410004   37.080002
```

8 Returns

TS303_yfinance Indices Bursatiles.ipynb

8.1 Daily Returns

8.1.1 Normal (simple or arithmetic) returns

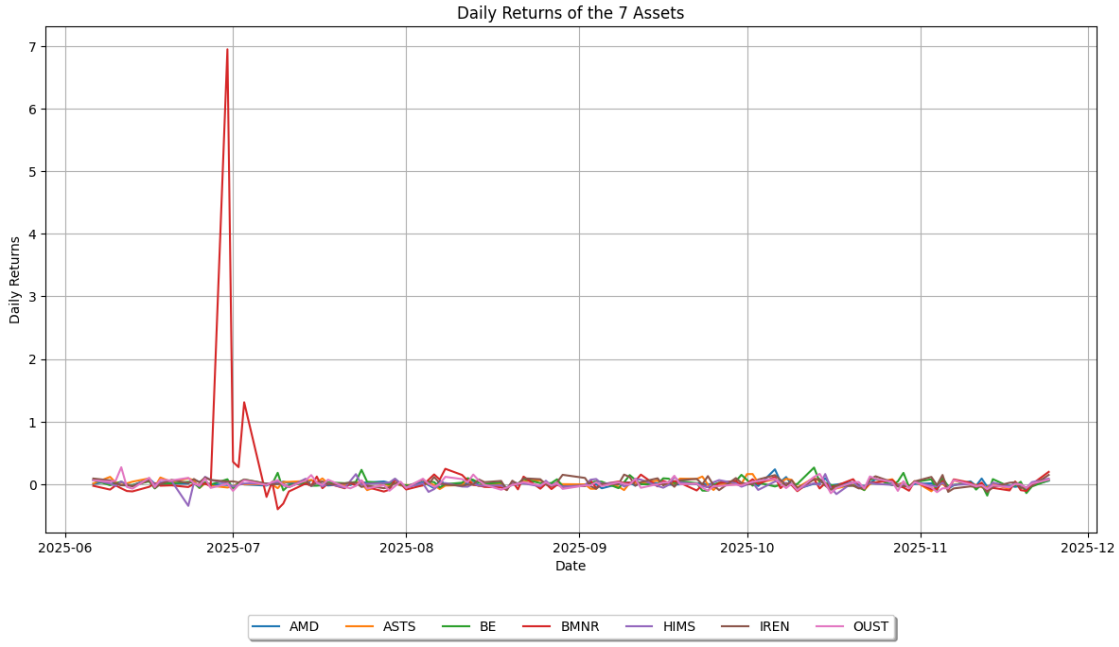
$$Return(R_t) = \frac{P_t - P_{t-1}}{P_{t-1}}$$

[24]:
```
# Daily Returns. Using pct_change()
# Normal (simple or arithmetic) returns
# Expresed in FRACTION.
# If Percentage is needed then multiply by 100.
daily_returns = prices.pct_change(fill_method=None)
daily_returns = daily_returns.dropna()
daily_returns.tail(5)
```

[24]:
```
Ticker           AMD      ASTS        BE      BMNR      HIMS      IREN  \
Date
2025-11-18 -0.042533  0.028622 -0.019979  0.042972  0.019112  0.030373
2025-11-19 -0.029267 -0.003607  0.037725 -0.096035 -0.011859 -0.061822
2025-11-20 -0.078416 -0.126013 -0.142752 -0.108293 -0.061680 -0.051495
2025-11-21 -0.010873  0.013215 -0.036303 -0.000769  0.032421 -0.027835
2025-11-24  0.055305  0.070664  0.061896  0.196154  0.088447  0.147421

Ticker          OUST
Date
2025-11-18  0.002868
2025-11-19 -0.001430
2025-11-20 -0.058711
```

```
2025-11-21  0.010649
2025-11-24  0.072253
```

[25]:
```python
# Plot all daily return series in a single plot
plt.figure(figsize=(12, 7))
for col in daily_returns.columns:
    plt.plot(daily_returns.index, daily_returns[col], label=col)

plt.title(f"Daily Returns of the {no_assets} Assets")
plt.ylabel("Daily Returns")
plt.xlabel("Date")
plt.legend(loc='upper center',
           bbox_to_anchor=(0.5, -0.15),
           ncol=10,
           fancybox=True,
           shadow=True)
plt.grid(True)
plt.tight_layout()
plt.show()
```



8.1.2 Summary

[26]:
```python
# stats and summary of daily retutns
print(f"Number of days of evaluation: {(today - daily_returns.index[0]).days}")
print(f"since {daily_returns.index[0].date()} until {today.date()}")
```

```
max_std = daily_returns.describe().loc['std'].sort_values(ascending=False)
print(f"\nThe asset with the biggest StdDev in Daily Returns is '{max_std.
 ↪index[0]}' with {100*max_std.iloc[0]:,.5}%")
print(f"The asset with the smallest StdDev in Daily Returns is '{max_std.
 ↪index[-1]}' with {100*max_std.iloc[-1]:,.5}%")

max_daily_return = daily_returns.describe().loc['max'].
 ↪sort_values(ascending=False)
print(f"The asset with the biggest Return in a single day is '{max_daily_return.
 ↪index[0]}' with {100*max_daily_return.iloc[0]:,.5}%")

min_daily_return = daily_returns.describe().loc['min'].
 ↪sort_values(ascending=True)
print(f"The asset with the lowest Return in a single day is '{min_daily_return.
 ↪index[0]}' with {100*min_daily_return.iloc[0]:,.5}%")

average_daily_returns = daily_returns.mean()
print(f"The average of all Daily Returns is {average_daily_returns.mean()*100:,.
 ↪4}%")
```

```
Number of days of evaluation: 172
since 2025-06-06 until 2025-11-25

The asset with the biggest StdDev in Daily Returns is 'BMNR' with 65.488%
The asset with the smallest StdDev in Daily Returns is 'AMD' with 3.9922%
The asset with the biggest Return in a single day is 'BMNR' with 694.84%
The asset with the lowest Return in a single day is 'BMNR' with -40.161%
The average of all Daily Returns is 1.57%
```

8.2 Cumulative Returns

TS303_yfinance Indices Bursatiles.ipynb

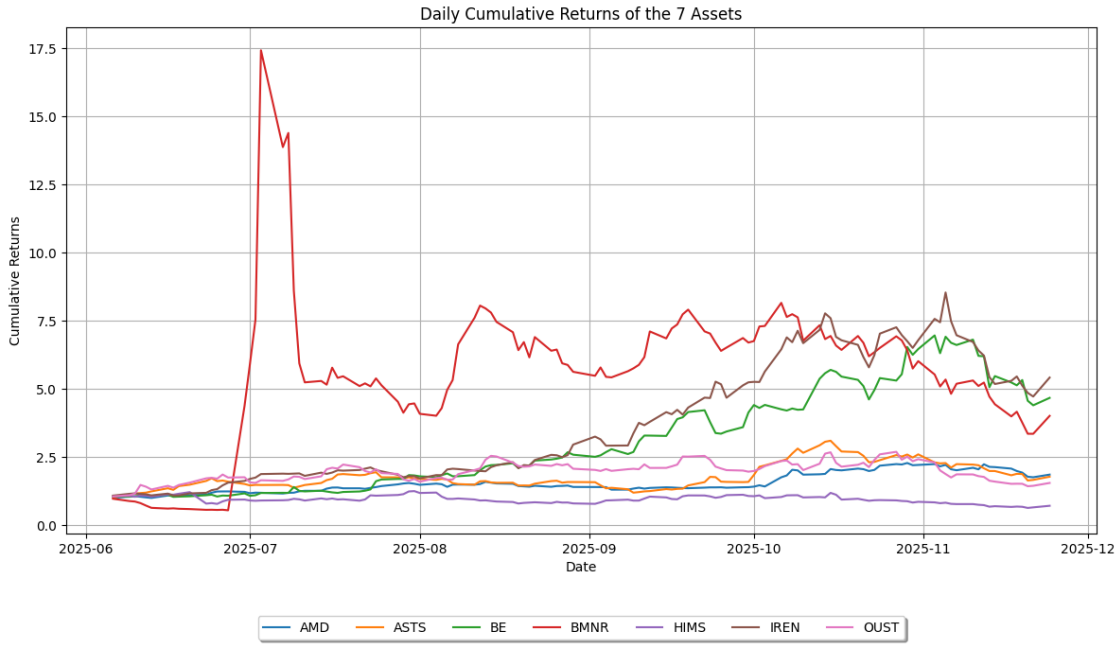
[27]:
```
# cumprod(): calculates the total return over a period by compounding the daily␣
 ↪returns. (a)(ab)(abc)
# It reflects how an initial investment would grow if it were continuously␣
 ↪reinvested and
# earned the daily returns.
cumulative_returns = (1 + daily_returns).cumprod()
cumulative_returns.tail(5)
```

[27]:

Ticker	AMD	ASTS	BE	BMNR	HIMS	IREN	\
Date							
2025-11-18	1.990578	1.887196	5.133007	4.165161	0.687133	5.458101	
2025-11-19	1.932319	1.880389	5.326650	3.765161	0.678984	5.120671	
2025-11-20	1.780793	1.643436	4.566259	3.357419	0.637104	4.856983	
2025-11-21	1.761431	1.665154	4.400489	3.354839	0.657760	4.721788	

```
2025-11-24  1.858847  1.782820  4.672860  4.012903  0.715937  5.417877

Ticker          OUST
Date
2025-11-18  1.524709
2025-11-19  1.522529
2025-11-20  1.433139
2025-11-21  1.448401
2025-11-24  1.553052
```

[28]:
```python
# Plot all cumulative return series in a single plot
plt.figure(figsize=(12, 7))
for col in cumulative_returns.columns:
    plt.plot(cumulative_returns.index, cumulative_returns[col], label=col)

plt.title(f"Daily Cumulative Returns of the {no_assets} Assets")
plt.ylabel("Cumulative Returns")
plt.xlabel("Date")
plt.legend(loc='upper center',
           bbox_to_anchor=(0.5, -0.15),
           ncol=10,
           fancybox=True,
           shadow=True)
plt.grid(True)
plt.tight_layout()
plt.show()
```



8.3 Final Cumulative Returns

Option 1 to obtain the final cumulative returns:

[29]:
```
# Rendimientos Acumulados al Final del Tiempo en (%) use .prod() y -1
print(f"Final Cumulative Returns in (%) in {(today - daily_returns.index[0]).
 ↪days} days of evaluation: ")
print(f"since {daily_returns.index[0].date()} until {today.date()}")

cumulative_returns_final = (1 + daily_returns).prod() - 1
cumulative_returns_final.name = "Final Cumulative Returns (%)"
cumulative_returns_final = round(cumulative_returns_final.
 ↪sort_values(ascending=False)*100 , 2)
display(cumulative_returns_final)
```

```
Final Cumulative Returns in (%) in 172 days of evaluation:
since 2025-06-06 until 2025-11-25
```

```
Ticker
IREN    441.79
BE      367.29
BMNR    301.29
AMD      85.88
ASTS     78.28
OUST     55.31
HIMS    -28.41
Name: Final Cumulative Returns (%), dtype: float64
```

[30]:
```
# Uncomment for learning purposes another option to compute the final␣
 ↪cumulative return values
"""
cumulative_returns_final_opt2 = (cumulative_returns.iloc[-1] - 1) * 100
cumulative_returns_final_opt2.name = "Final Cumulative Returns (%)"
cumulative_returns_final_opt2 = cumulative_returns_final_opt2.
 ↪sort_values(ascending=False)
cumulative_returns_final_opt2
"""
```

[30]:
```
'\ncumulative_returns_final_opt2 = (cumulative_returns.iloc[-1] - 1) *
100\ncumulative_returns_final_opt2.name = "Final Cumulative Returns
(%)"\ncumulative_returns_final_opt2 = cumulative_returns_final_opt2.sort_values(
ascending=False)\ncumulative_returns_final_opt2\n'
```

8.3.1 Summary

[31]:
```
print(f"Number of days of evaluation: {(today - cumulative_returns.index[0]).
 ↪days}")
print(f"From {cumulative_returns.index[0].date()} until {today.date()}")
```

```
print(f"The asset with the best cumulative return is '{cumulative_returns_final.
 ↪index[0]}' with a {cumulative_returns_final.iloc[0]:,.5}% in the period")
print(f"The asset with the worst cumulative return is␣
 ↪'{cumulative_returns_final.index[-1]}' with a {cumulative_returns_final.
 ↪iloc[-1]:,.5}% in the period")
```

```
Number of days of evaluation: 172
From 2025-06-06 until 2025-11-25
The asset with the best cumulative return is 'IREN' with a 441.79% in the period
The asset with the worst cumulative return is 'HIMS' with a -28.41% in the
period
```

9 Risk, Annualized Volatility

TS303_yfinance Indices Bursatiles.ipynb

[32]:
```
# Yearly volatility (risk)

# StdDev of daily returns in a 252-days year
annualized_volatility = daily_returns.std() * np.sqrt(252)

# Percentage (%)
annualized_volatility_percent = annualized_volatility * 100

# DataFrame of Annualized Volatility
volatility_df = pd.DataFrame(annualized_volatility_percent,␣
 ↪columns=["Volatility (%)"])

print("Annualized Assets Volatility:")

volatility_df = round(volatility_df.sort_values(by="Volatility (%)",␣
 ↪ascending=False), 2)

display(volatility_df)
```

```
Annualized Assets Volatility:
```

Ticker	Volatility (%)
BMNR	1039.59
OUST	106.72
BE	104.44
IREN	98.02
HIMS	95.13
ASTS	87.52
AMD	63.37

Stats of Volatility

[33]:
```
display(volatility_df.describe()) # all stocks
```

	Volatility (%)
count	7.000000
mean	227.827143
std	358.245943
min	63.370000
25%	91.325000
50%	98.020000
75%	105.580000
max	1039.590000

9.1 Summary

[34]:
```
# summary and stats

print(f"Number of days of evaluation: {(today - cumulative_returns.index[0]).
  ↪days}")
print(f"From {cumulative_returns.index[0].date()} until {today.date()}")

print(f"The asset with the biggest Annualized Volatility is '{volatility_df.
  ↪index[0]}' with a {volatility_df.iloc[0].values[0]:,.5} %")
print(f"The asset with the lowest Annualized Volatility is '{volatility_df.
  ↪index[-1]}' with a {volatility_df.iloc[-1].values[0]:,.5} %")
print(f"The Average Annualized Volatility of all assets is: {volatility_df.
  ↪mean().values[0]:,.5} %")
```

```
Number of days of evaluation: 172
From 2025-06-06 until 2025-11-25
The asset with the biggest Annualized Volatility is 'BMNR' with a 1,039.6 %
The asset with the lowest Annualized Volatility is 'AMD' with a 63.37 %
The Average Annualized Volatility of all assets is: 227.83 %
```

10 Initial Portfolio

10.1 Assets Weights

Compute Assets weights from the original portfolio

[35]:
```
#call the original portfolio [Ticker, Current QTY]
robinHood_xls

#Close prices df
close_prices = prices.iloc[-1]
close_prices.name = "Close Price"

#Merge original df with Close Prices
weights_df = pd.merge(robinHood_xls, close_prices, on='Ticker', how='inner')
```

```
#Add column of Amount Invested for each asset
weights_df['Investment'] = weights_df["Current QTY"] * weights_df["Close Price"]

#Calculate the Total Invested
Total_invested = weights_df['Investment'].sum()
print(f"Total Invested: ${Total_invested:,.2f}")

#Add column of weights of each asset
weights_df['Weights'] = weights_df['Investment'] / Total_invested
print(f"the sum of the weights is: {weights_df['Weights'].sum()}")

weights_df.set_index('Ticker', inplace=True)
weights_df.sort_values(by='Investment', ascending=False, inplace=True)
weights_df = weights_df.round(2)
display(weights_df)
```

```
Total Invested: $98,190.65
the sum of the weights is: 0.9999999999999999
```

Ticker	Current QTY	Close Price	Investment	Weights
IREN	638.06	48.49	30939.77	0.32
OUST	1100.79	21.37	23523.80	0.24
ASTS	201.13	55.00	11062.36	0.11
AMD	50.28	215.05	10811.87	0.11
HIMS	282.47	37.78	10671.68	0.11
BE	69.28	95.56	6620.74	0.07
BMNR	146.64	31.10	4560.43	0.05

TO-DO: Grafica de Pastel con los Porcentajes

Grafica por Sectores, Industrias

10.2 Portfolio Daily Returns

[36]:
```
#checking the shapes of the objects to multiply
# Daily Returns Expresed in FRACTION.
print("Portfolio daily returns = [Daily Returns] x [weights]")
print(f"daily_returns shape: {daily_returns.shape}")
print(f"weights_df shape: {weights_df['Weights'].shape}")
print(f"Matrix multiplication [{daily_returns.shape[0]} x {daily_returns.
  ↪shape[1]}] x [{weights_df['Weights'].shape[0]} x 1] = [{daily_returns.
  ↪shape[0]} x 1]")
```

```
Portfolio daily returns = [Daily Returns] x [weights]
daily_returns shape: (119, 7)
weights_df shape: (7,)
Matrix multiplication [119 x 7] x [7 x 1] = [119 x 1]
```

[37]:
```
# option 1: Portfolio's Daily Returns: Matrix multiplication (see above)
portfolio_daily_returns = (daily_returns @ weights_df['Weights'])

# option 2: Portfolio's Daily Returns: weighted sum of the daily returns of␣
 ↪each asset
# portfolio_daily_returns = (daily_returns * weights_df['Weights']).sum(axis=1)


portfolio_daily_returns.name = 'portfolio daily returns'
print("portfolio_daily_returns:")
display(portfolio_daily_returns) # Expresed in FRACTION (not percentage)
```

```
portfolio_daily_returns:
```

```
Date
2025-06-06    0.055419
2025-06-09    0.042768
2025-06-10    0.018696
2025-06-11    0.063678
2025-06-12   -0.022582
                ...
2025-11-18    0.011730
2025-11-19   -0.027208
2025-11-20   -0.075248
2025-11-21   -0.005107
2025-11-24    0.102241
Name: portfolio daily returns, Length: 119, dtype: float64
```

10.3 Portfolio Cumulative Returns

[38]:
```
# Portfolio's Cumulative daily Returns
portfolio_cumulative_returns = (1 + portfolio_daily_returns).cumprod()
portfolio_cumulative_returns = portfolio_cumulative_returns.rename('portfolio␣
 ↪cumulative returns')
print("Portfolio Daily Cumulative Returns:")
display(portfolio_cumulative_returns)
```

```
Portfolio Daily Cumulative Returns:
```

```
Date
2025-06-06    1.055419
2025-06-09    1.100557
2025-06-10    1.121133
2025-06-11    1.192524
2025-06-12    1.165595
                ...
2025-11-18    3.621898
2025-11-19    3.523354
2025-11-20    3.258228
```

```
2025-11-21    3.241587
2025-11-24    3.573012
Name: portfolio cumulative returns, Length: 119, dtype: float64
```

Portfolio Cumulative Final

[39]:
```
# Portfolio's final comulated return
portafolio_cumulative_final = (1 + portfolio_daily_returns).prod() - 1
print(f"portafolio_cumulative Returns_final: {portafolio_cumulative_final*100:,.
↪5}%")
```

```
portafolio_cumulative Returns_final: 257.3%
```

10.4 Portfolio Volatility

[40]:
```
# Compute annualized Volatility of the Portfolio
portfolio_annualized_volatility = portfolio_daily_returns.std() * np.sqrt(252)

# Convert to percentage
portfolio_annualized_volatility_perc = portfolio_annualized_volatility * 100

print(f"Portfolio Annualized Volatility: {portfolio_annualized_volatility_perc:
↪,.4}%")
```

```
Portfolio Annualized Volatility: 81.5%
```

10.5 Portfolio Sharpe Ratio

$$Sharpe = \frac{R_p - R_f}{\sigma_p} = \frac{\mu_p - r_f}{\sigma_p}$$

Where: * R_p: Expected Portfolio Return, μ (average rate of return) * R_f: Risk Free Rate (can be 0 if ignored) * σ_p: Portfolio Risk (StdDev) Standard Deviation of the portfolio's excess return.

[41]:
```
portfolio_annualized_return = portfolio_daily_returns.mean()*252 #Annualized
portfolio_sr = round((portfolio_annualized_return - (risk_free))/
↪portfolio_annualized_volatility, 2)
print(f"Portfolio Sharpe Ratio: {portfolio_sr}")
```

```
Portfolio Sharpe Ratio: 3.65
```

10.6 Summary

[42]:
```
print(f"Number of days of evaluation: {(today - cumulative_returns.index[0]).
↪days}")
print(f"From {cumulative_returns.index[0].date()} until {today.date()}")

print(f"Total Invested: ${Total_invested:,.2f}")
print(f"portafolio_cumulative returns_final (all period):␣
↪{portafolio_cumulative_final*100:,.4}%")
```

```
print(f"Portfolio Annualized Average Returns: {portfolio_annualized_return:,.
↪4%}")
print(f"Portfolio Annualized Volatility: {portfolio_annualized_volatility_perc:
↪,.4}%")
print(f"Portfolio Sharpe Ratio: {portfolio_sr}")
```

```
Number of days of evaluation: 172
From 2025-06-06 until 2025-11-25
Total Invested: $98,190.65
portafolio_cumulative returns_final (all period): 257.3%
Portfolio Annualized Average Returns: 301.5415%
Portfolio Annualized Volatility: 81.5%
Portfolio Sharpe Ratio: 3.65
```

11 Benchmarks (Indices)

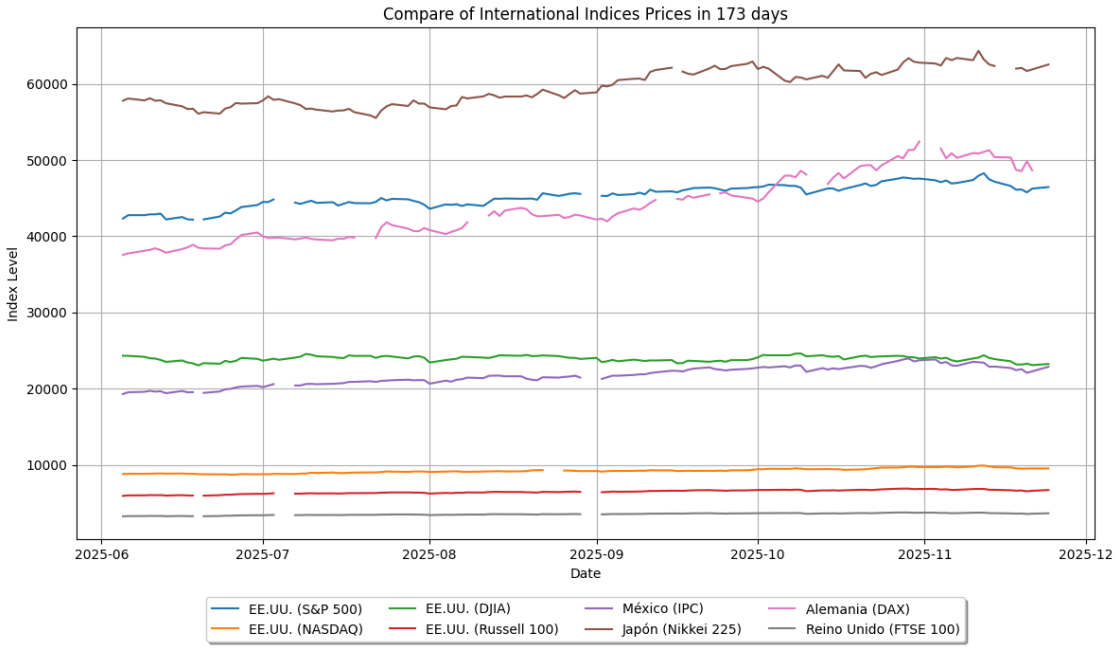
11.1 Daily Index Levels

[43]:
```
# Benchmark Indices
benchmark_indices = {
    "EE.UU. (S&P 500)": "^GSPC",
    "EE.UU. (NASDAQ)": "^IXIC",
    "EE.UU. (DJIA)": "^DJI",
    "EE.UU. (Russell 100)": "^RUI",
    "México (IPC)": "^MXX",
    "Japón (Nikkei 225)": "^N225",
    "Alemania (DAX)": "^GDAXI",
    "Reino Unido (FTSE 100)": "^FTSE"
}

# get data from yahoo finance
benchmarks_df = yf.download(list(benchmark_indices.values()),␣
↪start=first_valid_dates.iloc[0], end=today, auto_adjust=True)["Close"]

# Rename columns
benchmarks_df.columns = list(benchmark_indices.keys())

print("Benchmark Indices levels:")
display(benchmarks_df.tail(5))
```

```
[**********************100%**********************]  8 of 8 completed

Benchmark Indices levels:


            EE.UU. (S&P 500)  EE.UU. (NASDAQ)  EE.UU. (DJIA)  \
Date
2025-11-18      46091.738281      9552.299805   23180.529297
```

```
2025-11-19      46138.769531      9507.400391   23162.919922
2025-11-20      45752.261719      9527.700195   23278.849609
2025-11-21      46245.410156      9539.700195   23091.869141
2025-11-24      46448.269531      9534.900391   23239.179688

            EE.UU. (Russell 100)  México (IPC)  Japón (Nikkei 225)  \
Date
2025-11-18           6617.319824  22432.849609        61984.449219
2025-11-19           6642.160156  22564.230469        62080.578125
2025-11-20           6538.759766  22078.050781        61672.191406
2025-11-21           6602.990234  22273.080078        61877.269531
2025-11-24           6705.120117  22872.009766        62522.609375

            Alemania (DAX)  Reino Unido (FTSE 100)
Date
2025-11-18    48702.980469             3607.639893
2025-11-19    48537.699219             3620.030029
2025-11-20    49823.941406             3562.699951
2025-11-21    48625.878906             3599.479980
2025-11-24             NaN             3654.580078
```

[44]:
```python
# Plot Prices of each Index
plt.figure(figsize=(12, 7))
for col in benchmarks_df.columns:
    plt.plot(benchmarks_df.index, benchmarks_df[col], label=col)

plt.title(f"Compare of International Indices Prices in {(today - benchmarks_df.
  ↪index[0]).days} days")
plt.ylabel("Index Level")
plt.xlabel("Date")
plt.legend(loc='upper center',
           bbox_to_anchor=(0.5, -0.1),
           ncol=4,
           fancybox=True,
           shadow=True)
plt.grid(True)
plt.tight_layout()
plt.show()
```



11.2 Daily Returns

[45]:
```
# Daily Returns
daily_returns_bm = benchmarks_df.pct_change(fill_method=None)

# limpieza básica de daily_returns (elimina la primera fila con NaN)
daily_returns_bm = daily_returns_bm.dropna()
daily_returns_bm.tail(5)
```

[45]:
```
            EE.UU. (S&P 500)  EE.UU. (NASDAQ)  EE.UU. (DJIA)  \
Date
2025-11-13         -0.016529        -0.010463      -0.013939
2025-11-14         -0.006527        -0.011144      -0.006866
2025-11-19          0.001020        -0.004700      -0.000760
2025-11-20         -0.008377         0.002135       0.005005
2025-11-21          0.010779         0.001259      -0.008032

            EE.UU. (Russell 100)  México (IPC)  Japón (Nikkei 225)  \
Date
2025-11-13             -0.016557     -0.022904           -0.010464
2025-11-14             -0.000502      0.001322           -0.003211
2025-11-19              0.003754      0.005857            0.001551
2025-11-20             -0.015567     -0.021546           -0.006578
2025-11-21              0.009823      0.008834            0.003325

            Alemania (DAX)  Reino Unido (FTSE 100)
```

Date		
2025-11-13	0.004279	-0.017017
2025-11-14	-0.017653	-0.000811
2025-11-19	-0.003394	0.003434
2025-11-20	0.026500	-0.015837
2025-11-21	-0.024046	0.010324

11.3 Cumulative Daily Returns

[46]:
```
cumulative_returns_bm = (1 + daily_returns_bm).cumprod()
display(cumulative_returns_bm.tail(5))
```

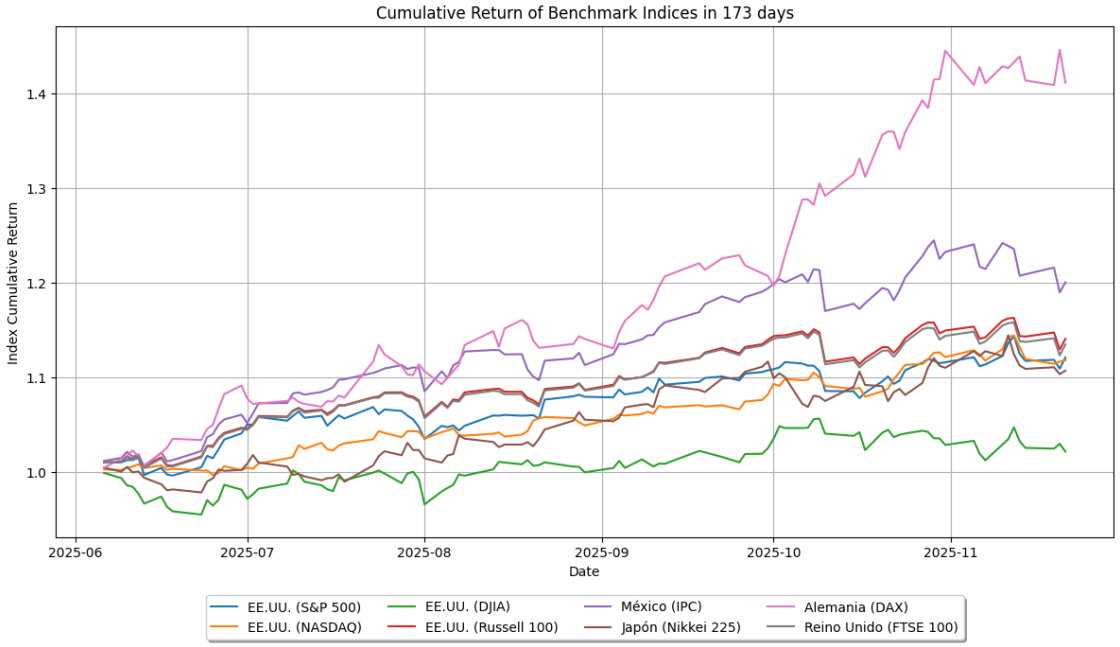
Date	EE.UU. (S&P 500)	EE.UU. (NASDAQ)	EE.UU. (DJIA)	\
2025-11-13	1.125086	1.132503	1.032860	
2025-11-14	1.117742	1.119882	1.025768	
2025-11-19	1.118883	1.114618	1.024989	
2025-11-20	1.109510	1.116998	1.030119	
2025-11-21	1.121469	1.118405	1.021845	

Date	EE.UU. (Russell 100)	México (IPC)	Japón (Nikkei 225)	\
2025-11-13	1.144037	1.207716	1.112895	
2025-11-14	1.143463	1.209313	1.109321	
2025-11-19	1.147756	1.216395	1.111042	
2025-11-20	1.129888	1.190186	1.103733	
2025-11-21	1.140987	1.200700	1.107403	

Date	Alemania (DAX)	Reino Unido (FTSE 100)
2025-11-13	1.439580	1.138676
2025-11-14	1.414167	1.137752
2025-11-19	1.409367	1.141660
2025-11-20	1.446715	1.123579
2025-11-21	1.411928	1.135179

[47]:
```
# Plot all cumulative return of Benchmarks in a single plot
plt.figure(figsize=(12, 7))
for col in cumulative_returns_bm.columns:
    plt.plot(cumulative_returns_bm.index, cumulative_returns_bm[col], label=col)

plt.title(f"Cumulative Return of Benchmark Indices in {(today - benchmarks_df.
  ↪index[0]).days} days")
plt.ylabel("Index Cumulative Return")
plt.xlabel("Date")
plt.legend(loc='upper center',
           bbox_to_anchor=(0.5, -0.1),
```

```
          ncol=4,
          fancybox=True,
          shadow=True)
plt.grid(True)
plt.tight_layout()
plt.show()
```



[48]:
```
# Final Cumulative Return in the period of evaluation for all Indices
print("Final Cumulative Returns (%)")
cumulative_returns_final_bm = (1 + daily_returns_bm).prod() -1
cumulative_returns_final_bm = cumulative_returns_final_bm.
 ↪sort_values(ascending=False)*100
cumulative_returns_final_bm = cumulative_returns_final_bm.rename("final␣
 ↪cumulative returns Benchmarks (%)")
display(cumulative_returns_final_bm)
```

```
Final Cumulative Returns (%)
```

```
Alemania (DAX)            41.192776
México (IPC)              20.069963
EE.UU. (Russell 100)      14.098708
Reino Unido (FTSE 100)    13.517870
EE.UU. (S&P 500)          12.146908
EE.UU. (NASDAQ)           11.840455
Japón (Nikkei 225)        10.740325
EE.UU. (DJIA)              2.184505
Name: final cumulative returns Benchmarks (%), dtype: float64
```

11.4 Volatility

[49]:
```
#Annualized Volatility
annualized_volatility_bm = daily_returns_bm.std() * np.sqrt(252)

#Annualized Volatility Percentage
annualized_volatility_bm_perc = annualized_volatility_bm * 100

annualized_volatility_bm_perc.rename('Volatility Benchmarks (%)', inplace=True)

print("Volatility of Benchmark Indices:")
annualized_volatility_bm_perc.sort_values(ascending=False, inplace=True)
display(annualized_volatility_bm_perc)
```

```
Volatility of Benchmark Indices:

Alemania (DAX)            19.672425
México (IPC)              15.059706
Japón (Nikkei 225)        13.292796
EE.UU. (DJIA)             13.005154
Reino Unido (FTSE 100)    11.128019
EE.UU. (Russell 100)      10.970399
EE.UU. (S&P 500)          10.934319
EE.UU. (NASDAQ)            7.901677
Name: Volatility Benchmarks (%), dtype: float64
```

11.5 Sharpe Ratios

$$Sharpe = \frac{R_p - R_f}{\sigma_p} = \frac{\mu_p - r_f}{\sigma_p}$$

Where: * R_p: Expected Portfolio Return, μ * R_f: Risk Free Rate (can be 0 if ignored) * σ_p: Portfolio Risk (StdDev)

[50]:
```
# Sharpe Ratios
annualized_return_bm = daily_returns_bm.mean()*252 # annualized
benchmark_sr = round((annualized_return_bm - (risk_free))/
 ↪annualized_volatility_bm, 2)
benchmark_sr.rename('Sharpe Ratio', inplace=True)
benchmark_sr.sort_values(ascending=False, inplace=True)
display(benchmark_sr)
```

```
Alemania (DAX)            4.40
EE.UU. (NASDAQ)           3.16
México (IPC)              2.93
EE.UU. (Russell 100)      2.77
Reino Unido (FTSE 100)    2.61
EE.UU. (S&P 500)          2.37
Japón (Nikkei 225)        1.73
```

```
EE.UU. (DJIA)              0.17
Name: Sharpe Ratio, dtype: float64
```

11.6 Summary

[51]:
```
# Days of evaluation
no_days_compare = today.date() - first_valid_dates.iloc[0].date()
print(f"\nNumber of days of evaluation: {no_days_compare.days} days.")
print(f"From: {first_valid_dates.iloc[0].date()} to: {today.date()}")

# Summary Return and Volatility
print("\nSummary: Benchmark Indices:")
print(f"'{cumulative_returns_final_bm.index[0]}' has the largest total return␣
 ↪{cumulative_returns_final_bm.iloc[0]:,.5}%")
print(f"and'{cumulative_returns_final_bm.index[-1]}' the lowest total return␣
 ↪{cumulative_returns_final_bm.iloc[-1]:,.3}%")
print(f"\n'{annualized_volatility_bm_perc.index[0]}' has the largest volatility␣
 ↪{annualized_volatility_bm_perc.iloc[0]:,.5}%")
print(f"and '{annualized_volatility_bm_perc.index[-1]}' the lowest volatility␣
 ↪{annualized_volatility_bm_perc.iloc[-1]:,.5}%")
```

```
Number of days of evaluation: 173 days.
From: 2025-06-05 to: 2025-11-25

Summary: Benchmark Indices:
'Alemania (DAX)' has the largest total return 41.193%
and'EE.UU. (DJIA)' the lowest total return 2.18%

'Alemania (DAX)' has the largest volatility 19.672%
and 'EE.UU. (NASDAQ)' the lowest volatility 7.9017%
```

12 Compare Initial Portfolio and Indices

12.1 Daily Returns

Combine dataframes

[52]:
```
# Combine Daily Returns of Initial Porfolio and
# Benchmark Indices in a single dataFrame to plot

# convert portfolio_daily_returns to datafre to merge it with Indices daily␣
 ↪returns
portfolio_daily_returns_df = pd.DataFrame(portfolio_daily_returns)
portfolio_daily_returns_df.rename(columns={'portfolio daily returns':'Initial␣
 ↪Portfolio'}, inplace=True)

# check if the lenghts of the dataframes match
```

```
if len(portfolio_daily_returns_df) != len(daily_returns_bm):
    print(f"Lengths of DataFrames don't match {len(portfolio_daily_returns_df)}␣
 ↪vs {len(daily_returns_bm)} but a left merge will help")


# Merge Daily Returns of Benchmark Indices and Initial Portfolio
# Note: Left-Merge on Benchmark daily returns because they don't operate on␣
 ↪weekends or bank holidays
# thus we compare both benchmark and portfolio using the same labor day dates␣
 ↪only.
merge_daily_returns = pd.merge(daily_returns_bm, portfolio_daily_returns_df,␣
 ↪on='Date', how="left")
print("\nDaily Returns (merged portfolio and indices):")
display(merge_daily_returns.tail(5))
```

```
Lengths of DataFrames don't match 119 vs 98 but a left merge will help

Daily Returns (merged portfolio and indices):
```

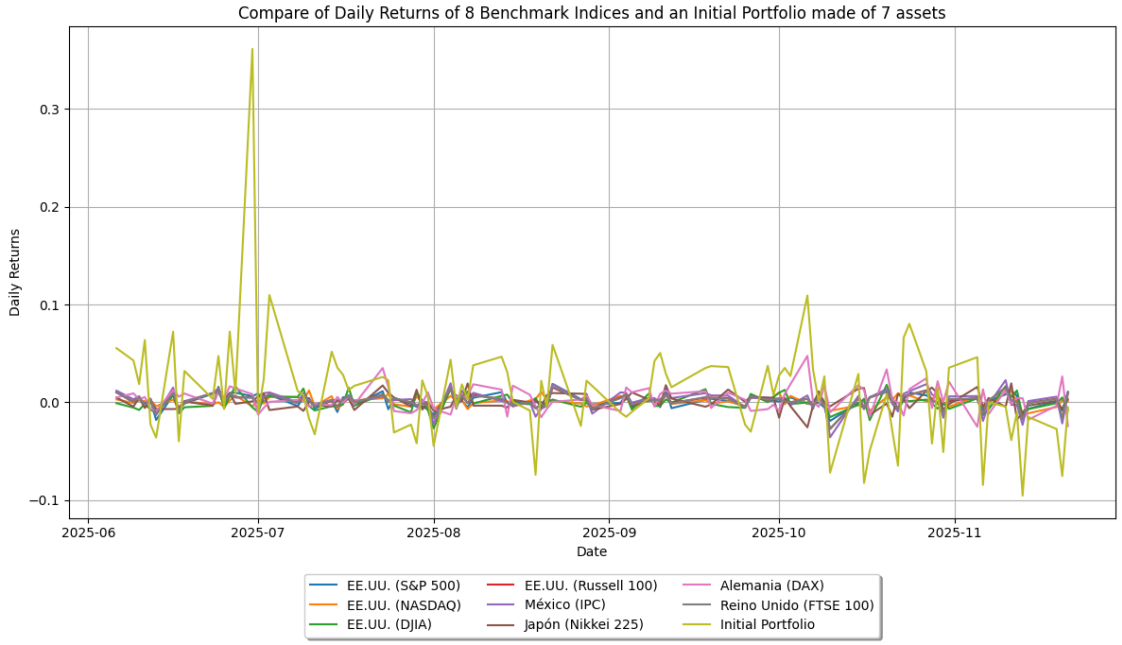
Date	EE.UU. (S&P 500)	EE.UU. (NASDAQ)	EE.UU. (DJIA)	\
2025-11-13	-0.016529	-0.010463	-0.013939	
2025-11-14	-0.006527	-0.011144	-0.006866	
2025-11-19	0.001020	-0.004700	-0.000760	
2025-11-20	-0.008377	0.002135	0.005005	
2025-11-21	0.010779	0.001259	-0.008032	

Date	EE.UU. (Russell 100)	México (IPC)	Japón (Nikkei 225)	\
2025-11-13	-0.016557	-0.022904	-0.010464	
2025-11-14	-0.000502	0.001322	-0.003211	
2025-11-19	0.003754	0.005857	0.001551	
2025-11-20	-0.015567	-0.021546	-0.006578	
2025-11-21	0.009823	0.008834	0.003325	

Date	Alemania (DAX)	Reino Unido (FTSE 100)	Initial Portfolio
2025-11-13	0.004279	-0.017017	-0.095230
2025-11-14	-0.017653	-0.000811	-0.014870
2025-11-19	-0.003394	0.003434	-0.027208
2025-11-20	0.026500	-0.015837	-0.075248
2025-11-21	-0.024046	0.010324	-0.005107

[53]:
```
# Plot all daily Returns
plt.figure(figsize=(12, 7))
for col in merge_daily_returns.columns:
    plt.plot(merge_daily_returns.index, merge_daily_returns[col], label=col)
```

```
plt.title(f"Compare of Daily Returns of {len(benchmark_indices)} Benchmark␣
 ↪Indices and an Initial Portfolio made of {no_assets} assets")
plt.ylabel("Daily Returns")
plt.xlabel("Date")
plt.legend(loc='upper center',
           bbox_to_anchor=(0.5, -0.1),
           ncol=3,
           fancybox=True,
           shadow=True)
plt.grid(True)
plt.tight_layout()
plt.show()
```



12.2 Cumulative Returns

[54]:
```
# Combine Comulative Returns of Initial Porfolio and Benchmark Indices in a␣
 ↪single dataFrame to plot

# convert portfolio_cumulative_returns to datafre to merge it with Indices␣
 ↪returns
portfolio_cumulative_returns_df = pd.DataFrame(portfolio_cumulative_returns)
portfolio_cumulative_returns_df.rename(columns={'portfolio cumulative returns':
 ↪'Initial Portfolio'}, inplace=True)

# check if the lenghts of the dataframes match
```

```
if len(portfolio_cumulative_returns_df) != len(cumulative_returns_bm):
    print(f"Lengths of DataFrames don't match␣
 ↪{len(portfolio_cumulative_returns_df)} vs {len(cumulative_returns_bm)} but a␣
 ↪left merge will help")


# Merge Cumulative Returns of Benchmark Indices and Initial Portfolio
# Note: Left-Merge on Benchmark cumulative returns because they don't operate␣
 ↪on weekends or bank holidays
# thus we compare both benchmark and portfolio using the same labor day dates␣
 ↪only.
merge_cumulative_returns = pd.merge(cumulative_returns_bm,␣
 ↪portfolio_cumulative_returns_df, on='Date', how="left")
print("\nDaily Cumulative Returns (merged portfolio and indices):")
display(merge_cumulative_returns.tail(5))
```

```
Lengths of DataFrames don't match 119 vs 98 but a left merge will help

Daily Cumulative Returns (merged portfolio and indices):
```

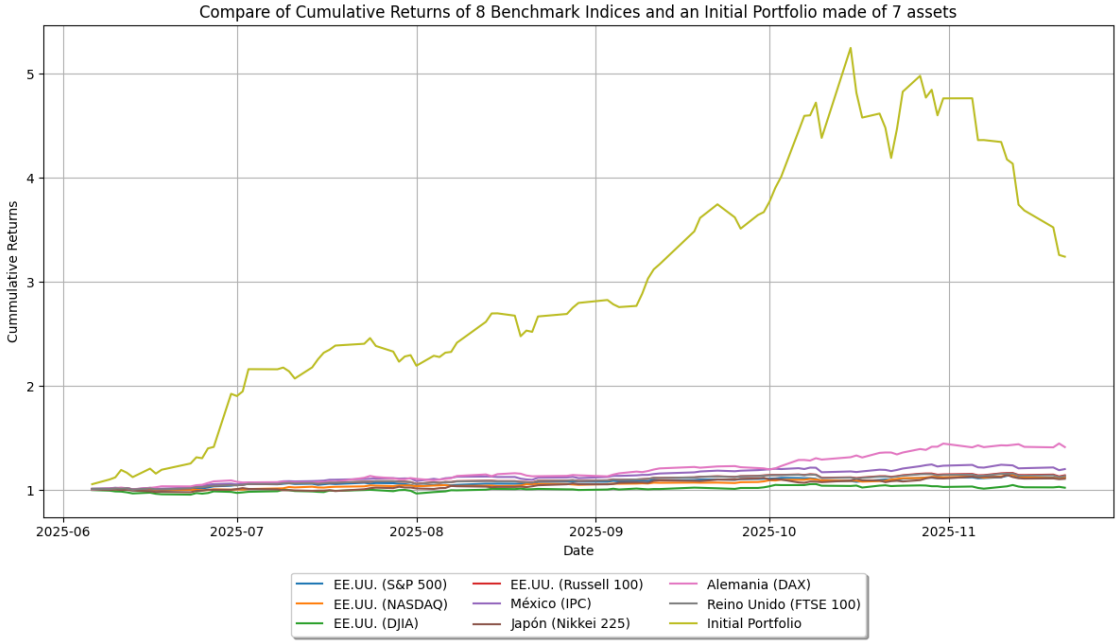
Date	EE.UU. (S&P 500)	EE.UU. (NASDAQ)	EE.UU. (DJIA)	\
2025-11-13	1.125086	1.132503	1.032860	
2025-11-14	1.117742	1.119882	1.025768	
2025-11-19	1.118883	1.114618	1.024989	
2025-11-20	1.109510	1.116998	1.030119	
2025-11-21	1.121469	1.118405	1.021845	

Date	EE.UU. (Russell 100)	México (IPC)	Japón (Nikkei 225)	\
2025-11-13	1.144037	1.207716	1.112895	
2025-11-14	1.143463	1.209313	1.109321	
2025-11-19	1.147756	1.216395	1.111042	
2025-11-20	1.129888	1.190186	1.103733	
2025-11-21	1.140987	1.200700	1.107403	

Date	Alemania (DAX)	Reino Unido (FTSE 100)	Initial Portfolio
2025-11-13	1.439580	1.138676	3.741297
2025-11-14	1.414167	1.137752	3.685664
2025-11-19	1.409367	1.141660	3.523354
2025-11-20	1.446715	1.123579	3.258228
2025-11-21	1.411928	1.135179	3.241587

[55]:
```
# Plot all daily cummulative Returns
plt.figure(figsize=(12, 7))
for col in merge_cumulative_returns.columns:
```

```
    plt.plot(merge_cumulative_returns.index, merge_cumulative_returns[col],␣
 ↪label=col)

plt.title(f"Compare of Cumulative Returns of {len(benchmark_indices)} Benchmark␣
 ↪Indices and an Initial Portfolio made of {no_assets} assets")
plt.ylabel("Cummulative Returns")
plt.xlabel("Date")
plt.legend(loc='upper center',
           bbox_to_anchor=(0.5, -0.1),
           ncol=3,
           fancybox=True,
           shadow=True)
plt.grid(True)
plt.tight_layout()
plt.show()
```



[56]:
```
# Total Cumulative Returns
print("\nTotal (Final) cumulative Returns (%):")
merge_cumulative_returns_finals = (merge_cumulative_returns.iloc[-1] - 1) * 100
merge_cumulative_returns_finals.rename('Final Cumulative Returns (%)',␣
 ↪inplace=True)
merge_cumulative_returns_finals.sort_values(ascending=False, inplace=True)
display(merge_cumulative_returns_finals)
```

```
Total (Final) cumulative Returns (%):
```

```
Initial Portfolio         224.158721
Alemania (DAX)             41.192776
México (IPC)               20.069963
EE.UU. (Russell 100)       14.098708
Reino Unido (FTSE 100)     13.517870
EE.UU. (S&P 500)           12.146908
EE.UU. (NASDAQ)            11.840455
Japón (Nikkei 225)         10.740325
EE.UU. (DJIA)               2.184505
Name: Final Cumulative Returns (%), dtype: float64
```

12.3 Volatility

[57]:
```
# Anualized Volatility of portfolio + Benchmarks
print('\nAnnualized Volatility (%):')
merge_annualized_volatility = merge_daily_returns.std() * np.sqrt(252)
merge_annualized_volatility = merge_annualized_volatility * 100
merge_annualized_volatility.rename('Annualized Volatility (%)', inplace=True)
merge_annualized_volatility.sort_values(ascending=False, inplace=True)
display(merge_annualized_volatility)
```

```
Annualized Volatility (%):
```

```
Initial Portfolio         85.741979
Alemania (DAX)            19.672425
México (IPC)              15.059706
Japón (Nikkei 225)        13.292796
EE.UU. (DJIA)             13.005154
Reino Unido (FTSE 100)    11.128019
EE.UU. (Russell 100)      10.970399
EE.UU. (S&P 500)          10.934319
EE.UU. (NASDAQ)            7.901677
Name: Annualized Volatility (%), dtype: float64
```

12.4 Sharpe Ratio

[58]:
```
sharpe_ratio = pd.DataFrame()
sharpe_ratio['Annualized Returns (%)'] = merge_daily_returns.mean()*252*100␣
 ↪#Annualized returns
sharpe_ratio = pd.concat([sharpe_ratio['Annualized Returns (%)'],␣
 ↪merge_annualized_volatility], axis=1)
sharpe_ratio['Sharpe Ratio'] = (sharpe_ratio['Annualized Returns (%)'] -␣
 ↪(risk_free*100)) / (sharpe_ratio['Annualized Volatility (%)'])
sharpe_ratio.sort_values(by='Sharpe Ratio', ascending=False, inplace=True)
sharpe_ratio
```

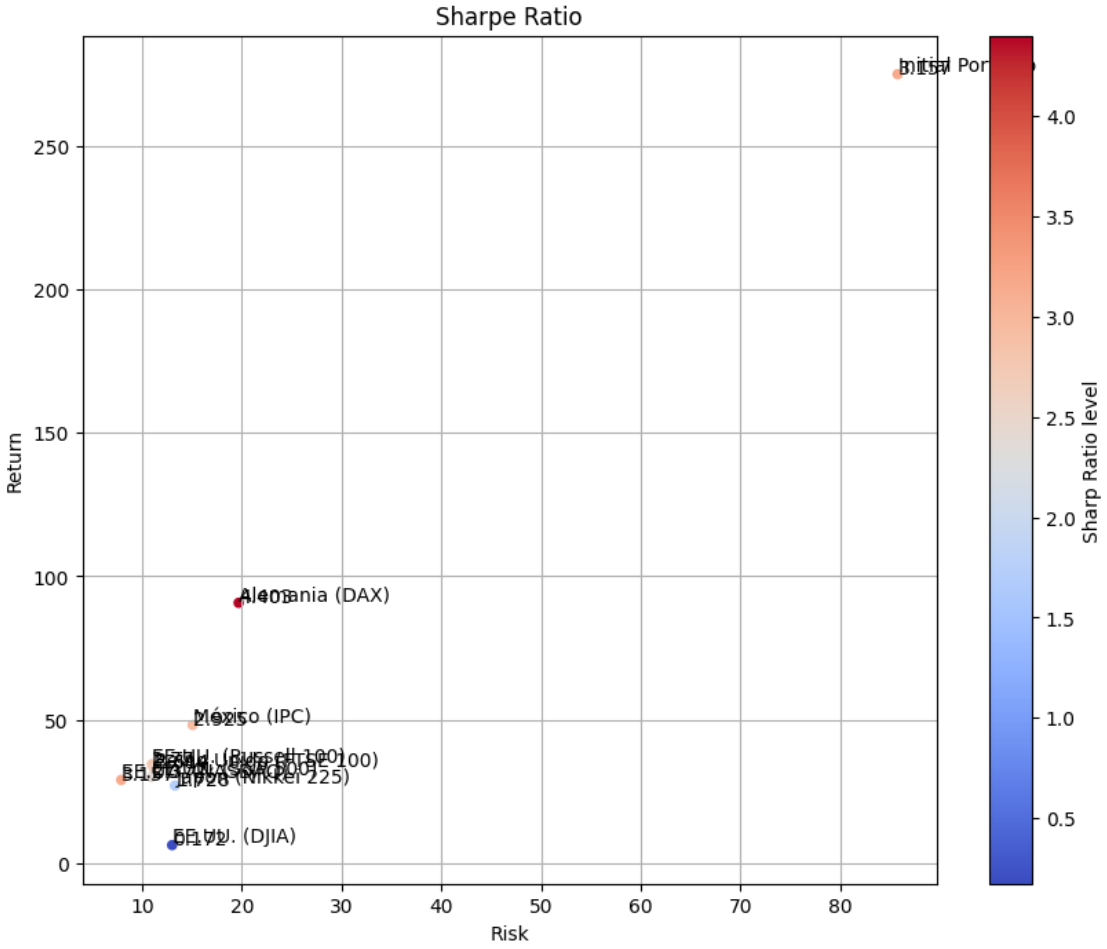
[58]:

	Annualized Returns (%)	Annualized Volatility (%)	\
Alemania (DAX)	90.760442	19.672425	
EE.UU. (NASDAQ)	29.100245	7.901677	
Initial Portfolio	274.849112	85.741979	
México (IPC)	48.204464	15.059706	
EE.UU. (Russell 100)	34.535971	10.970399	
Reino Unido (FTSE 100)	33.239149	11.128019	
EE.UU. (S&P 500)	30.087931	10.934319	
Japón (Nikkei 225)	27.120712	13.292796	
EE.UU. (DJIA)	6.394808	13.005154	

	Sharpe Ratio
Alemania (DAX)	4.402530
EE.UU. (NASDAQ)	3.157335
Initial Portfolio	3.157113
México (IPC)	2.925187
EE.UU. (Russell 100)	2.769632
Reino Unido (FTSE 100)	2.613866
EE.UU. (S&P 500)	2.371975
Japón (Nikkei 225)	1.727907
EE.UU. (DJIA)	0.172455

TO-do: Poner nota explicatoria de por que el SR es 1.58 aqui contra 1.3 arriba

[59]:
```
#fig, ax = plt.subplots()

ax = sharpe_ratio.plot.scatter(
                        x=sharpe_ratio.columns[1],
                        y=sharpe_ratio.columns[0],
                        c=sharpe_ratio.columns[2],
                        colormap='coolwarm',
                        alpha=1.0,
                        figsize=(10,8))
for i, label in enumerate(round(sharpe_ratio['Sharpe Ratio'],3)):
    ax.text(sharpe_ratio['Annualized Volatility (%)'].iloc[i] + 0.05,␣
 ↪sharpe_ratio['Annualized Returns (%)'].iloc[i], label)
    ax.text(sharpe_ratio['Annualized Volatility (%)'].iloc[i] + 0.05,␣
 ↪sharpe_ratio['Annualized Returns (%)'].iloc[i] + 1, sharpe_ratio.index[i])

ax.figure.axes[1].set_ylabel('Sharp Ratio level')

plt.title('Sharpe Ratio')
plt.xlabel('Risk')
plt.ylabel('Return')
plt.grid()
plt.show()
```



12.5 Summary

[60]:
```
# Days of evaluation
no_days_compare = today.date() - first_valid_dates.iloc[0].date()
print(f"\nNumber of days of evaluation: {no_days_compare.days} days. From:␣
 ↪{first_valid_dates.iloc[0].date()} to: {today.date()}")

# Summary Return, Volatility and Sharp Ratio
print("\nSummary: Benchmark Indices + Initial Portfolio")
print("\nReturns:")
print(f"'{merge_cumulative_returns_finals.index[0]}' has the largest total␣
 ↪return {merge_cumulative_returns_finals.iloc[0]:,.5}%")
print(f"and'{merge_cumulative_returns_finals.index[-1]}' the lowest total␣
 ↪return {merge_cumulative_returns_finals.iloc[-1]:,.3}%")
print("\nVolatility:")
print(f"'{merge_annualized_volatility.index[0]}' has the largest volatility␣
 ↪{merge_annualized_volatility.iloc[0]:,.5}%")
```

```
print(f"and '{merge_annualized_volatility.index[-1]}' the lowest volatility␣
  ↪{merge_annualized_volatility.iloc[-1]:,.5}%")
print("\nSharp Ratio:")
print(f"'{sharpe_ratio.index[0]}' has the best Sharpe Ratio␣
  ↪{sharpe_ratio['Sharpe Ratio'].iloc[0]:,.4}")
print(f"and '{sharpe_ratio.index[-1]}' the worst Sharp Ratio␣
  ↪{sharpe_ratio['Sharpe Ratio'].iloc[-1]:,.3}")
```

```
Number of days of evaluation: 173 days. From: 2025-06-05 to: 2025-11-25

Summary: Benchmark Indices + Initial Portfolio

Returns:
'Initial Portfolio' has the largest total return 224.16%
and'EE.UU. (DJIA)' the lowest total return 2.18%

Volatility:
'Initial Portfolio' has the largest volatility 85.742%
and 'EE.UU. (NASDAQ)' the lowest volatility 7.9017%

Sharp Ratio:
'Alemania (DAX)' has the best Sharpe Ratio 4.403
and 'EE.UU. (DJIA)' the worst Sharp Ratio 0.172
```

13 Buy and Hold

13.1 B&H - Single Stock

Choose the one with **Biggest Final Cumulative Retur**n

If I have bought $100,000 in each Index and Initial Portfolio

Option 1 to compute the B&H of a singl estock:

[61]:
```
# If I have bought at the begining of the period $100,000 usd, how much it␣
  ↪would be today?

# Days of evaluation
no_days_compare = today.date() - first_valid_dates.iloc[0].date()
print(f"\nNumber of days of evaluation: {no_days_compare.days} days.")

# Initial investment
initial_investment = 100000

print(f"The asset with the best cumulative return is '{cumulative_returns_final.
  ↪index[0]}' with a {cumulative_returns_final.iloc[0]:,.5}% in the period")
```

```
stock_final_cumulative_return = cumulative_returns_final.iloc[0] / 100

# From: Cumulative Return = (Final Value / Initial Value) - 1
# then, Final value = Initial Value * (1 + Cumulative Return)
stock_final_value = (stock_final_cumulative_return + 1) * initial_investment

# Margin = Final Value - Initial Value
stock_margin = stock_final_value - initial_investment

# Yields in percentage
yields_perc = ((stock_final_value / initial_investment) - 1) * 100

print(f"From: {first_valid_dates.iloc[0].date()} to: {today.date()}")
print(f"\nInvestment Analysis of ${initial_investment:,.2f} USD")
print(f"\nStock with the best cumulative return: {cumulative_returns_final.
  ↪index[0]}")
print(f"\nInitial Value (USD): ${initial_investment:,.2f}")
print(f"\nFinal Value (USD): ${stock_final_value:,.2f}")
print(f"\nMargin (USD): ${stock_margin:,.2f}")
print(f"\nYields: {yields_perc:,.2f}%")
```

```
Number of days of evaluation: 173 days.
The asset with the best cumulative return is 'IREN' with a 441.79% in the period
From: 2025-06-05 to: 2025-11-25

Investment Analysis of $100,000.00 USD

Stock with the best cumulative return: IREN

Initial Value (USD): $100,000.00

Final Value (USD): $541,790.00

Margin (USD): $441,790.00

Yields: 441.79%
```

Option 2 to compute the B&H of a single stock:

[62]:
```
# Chosen Stock: The one with the largest return in the period
stock = cumulative_returns_final.index[0]

# Initial Investment
initial_investment

# Initial Price of the Stock
initial_price = prices[stock].iloc[0]
```

```
# Final price of the Stock
final_price = prices[stock].iloc[-1]

# Shares bought at the begining
shares_bought = initial_investment / initial_price

# Final value ($usd) of the shares
final_value = shares_bought * final_price

# Margin
margin = final_value - initial_investment

# Return
return_perc = (margin / initial_investment) * 100

# Results
print(f"Stock: {stock}")
print(f"Initial date: {first_valid_dates.iloc[0].date()}")
print(f"Initial Investment: ${initial_investment:,.2f}")
print(f"Initial Price: ${initial_price:,.2f}")
print(f"Final price: ${final_price:,.2f}")
print(f"Shares bought: {shares_bought:,.2f}")
print(f"Final value of the investment: ${final_value:,.2f}")
print(f"Margin: ${margin:,.2f}")
print(f"Yield: {return_perc:,.2f}%")
```

```
Stock: IREN
Initial date: 2025-06-05
Initial Investment: $100,000.00
Initial Price: $8.95
Final price: $48.49
Shares bought: 11,173.18
Final value of the investment: $541,787.74
Margin: $441,787.74
Yield: 441.79%
```

13.2 B&H - Indices and Portfolio

If I have bought $100,000 in each Index and Initial Portfolio

[63]:
```
# If I have bought at the begining of the period $100,000 usd, how much it␣
↪would be today?

# Days of evaluation
no_days_compare = today.date() - first_valid_dates.iloc[0].date()
print(f"\nNumber of days of evaluation: {no_days_compare.days} days.")
```

```python
# Initial investment
initial_investment = 100000

# From: Cumulative Return = (Final Value / Initial Value) - 1
# then, Final value = Initial Value * (1 + Cumulative Return)
merge_final_values = initial_investment * (1 + merge_cumulative_returns.
 ↪iloc[-1])

# Gain = Final Value - Initial Value
merge_gain = merge_final_values - initial_investment

#Cumulative Returns
merge_cumulative_returns_finals

print(f"From: {first_valid_dates.iloc[0].date()} to: {today.date()}")
print(f"\nInvestment Analysis of ${initial_investment:,.2f} USD")
print("\nBenchmarks & Initial Portfolio:")
print(f"\nInitial Value (USD): ${initial_investment:,.2f}")
print(f"\nFinal Values (USD):")
print(merge_final_values.sort_values(ascending=False))
print(f"\nMargins (USD):")
print(merge_gain.sort_values(ascending=False))
print(f"\nYields (%):")
print(merge_cumulative_returns_finals.sort_values(ascending=False))
```

```
Number of days of evaluation: 173 days.
From: 2025-06-05 to: 2025-11-25

Investment Analysis of $100,000.00 USD

Benchmarks & Initial Portfolio:

Initial Value (USD): $100,000.00

Final Values (USD):
Initial Portfolio         424158.721187
Alemania (DAX)            241192.776251
México (IPC)              220069.963232
EE.UU. (Russell 100)      214098.707521
Reino Unido (FTSE 100)    213517.869763
EE.UU. (S&P 500)          212146.908293
EE.UU. (NASDAQ)           211840.454603
Japón (Nikkei 225)        210740.324517
EE.UU. (DJIA)             202184.504918
Name: 2025-11-21 00:00:00, dtype: float64

Margins (USD):
```

```
Initial Portfolio         324158.721187
Alemania (DAX)            141192.776251
México (IPC)              120069.963232
EE.UU. (Russell 100)      114098.707521
Reino Unido (FTSE 100)    113517.869763
EE.UU. (S&P 500)          112146.908293
EE.UU. (NASDAQ)           111840.454603
Japón (Nikkei 225)        110740.324517
EE.UU. (DJIA)             102184.504918
Name: 2025-11-21 00:00:00, dtype: float64

Yields (%):
Initial Portfolio         224.158721
Alemania (DAX)             41.192776
México (IPC)               20.069963
EE.UU. (Russell 100)       14.098708
Reino Unido (FTSE 100)     13.517870
EE.UU. (S&P 500)           12.146908
EE.UU. (NASDAQ)            11.840455
Japón (Nikkei 225)         10.740325
EE.UU. (DJIA)               2.184505
Name: Final Cumulative Returns (%), dtype: float64
```

14 Dollar Cost Averaging (DCA)

14.1 Single Stock

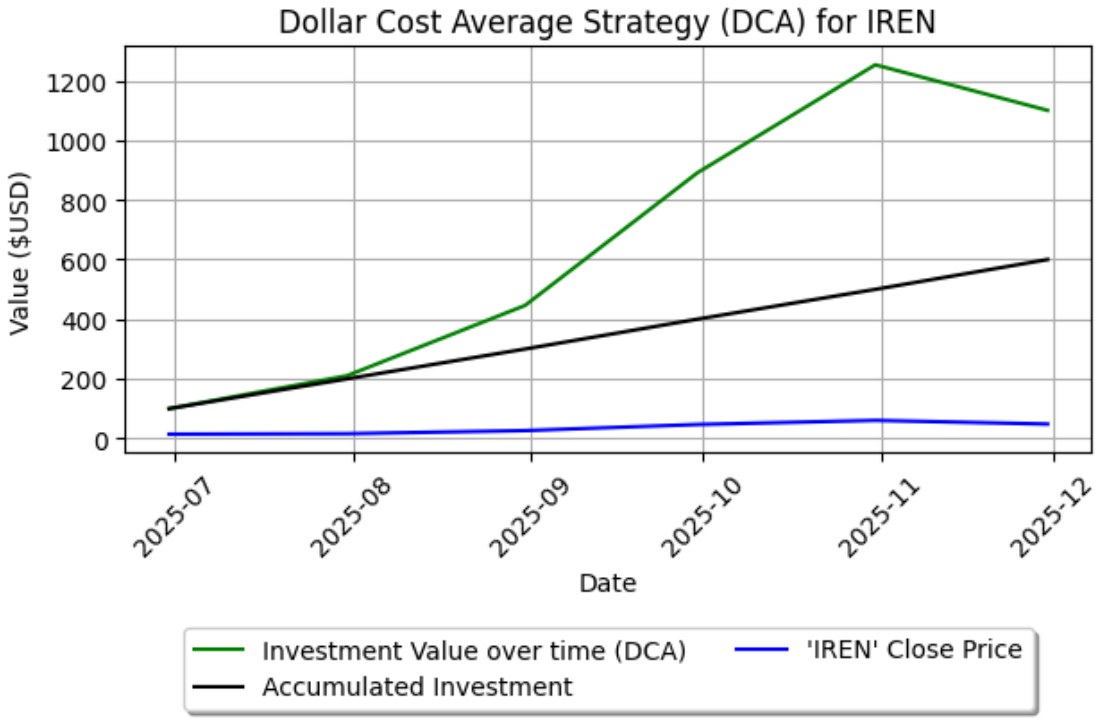
[64]:
```python
prices_dca = pd.DataFrame.copy(prices)

# Making sure the index of the dataframe is in datetime format
if not type(prices_dca.index) == pd.core.indexes.datetimes.DatetimeIndex :
    prices_dca.index = pd.to_datetime(prices_dca.index)
    print(type(prices_dca.index))

# Chosen Stock: The one with the largest return in the period
stock = cumulative_returns_final.index[0]

# "Last_days" is a DataFrame with the Latest Days of each month from the␣
 ↪selected Stock's prices
last_days = prices_dca[stock].resample('ME').last() # Last day of each Month
last_days = pd.DataFrame(last_days)

# Monthly investment
last_days['Monthly_Investment'] = 100    # $100usd every month

# Cumulative Investment
last_days['Accumulated_Investment'] = last_days['Monthly_Investment'].cumsum()
```

```
# Shares bought each month
last_days['Shares_bought'] = last_days['Monthly_Investment'] / last_days[stock]

# Get the cumulative number of shares bought
last_days['Accumulated_shares'] = last_days['Shares_bought'].cumsum()

# Get the value of the investment through the time
last_days['Investment_value'] = last_days['Accumulated_shares'] *␣
 ↪last_days[stock]

display(last_days)

# plots
plt.Figure(figsize=(12, 9))
plt.plot(last_days.index, last_days['Investment_value'], label='Investment␣
 ↪Value over time (DCA)', color='green')
plt.plot(last_days['Accumulated_Investment'], label='Accumulated Investment',␣
 ↪color='black')
plt.plot(last_days[stock], label=f"'{stock}' Close Price", color='blue')
plt.ylabel('Value ($USD)')
plt.xlabel('Date')
plt.xticks(rotation=45)
plt.title(f'Dollar Cost Average Strategy (DCA) for {stock}')
#plt.legend()
plt.legend(loc='upper center',
          bbox_to_anchor=(0.5, -0.4),
          ncol=2,
          fancybox=True,
          shadow=True)
plt.grid(True)
plt.tight_layout()
plt.show()
```

Date	IREN	Monthly_Investment	Accumulated_Investment
2025-06-30	14.570000	100	100
2025-07-31	16.110001	100	200
2025-08-31	26.480000	100	300
2025-09-30	46.930000	100	400
2025-10-31	60.750000	100	500
2025-11-30	48.490002	100	600

Date	Shares_bought	Accumulated_shares	Investment_value
2025-06-30	6.863418	6.863418	100.000000
2025-07-31	6.207324	13.070743	210.569670

2025-08-31	3.776435	16.847178	446.113256
2025-09-30	2.130833	18.978011	890.638052
2025-10-31	1.646091	20.624101	1252.914155
2025-11-30	2.062281	22.686382	1100.062708



14.2 **TO-DO: DCA for each Index and Initial Portfolio

15 Momentum

Applied to all **Stocks** in Initial Portfolio

[65]:
```
# call the DataFrame of the Final Cumulative Returns, which is the total return␣
↪of each asset in the period.
cumulative_returns_final

# MOMENTUM: Order from largest to lowest return
momentum_ranking = cumulative_returns_final.sort_values(ascending=False)
print("Momentum Ranking (Annual yield):")
display(momentum_ranking)
```

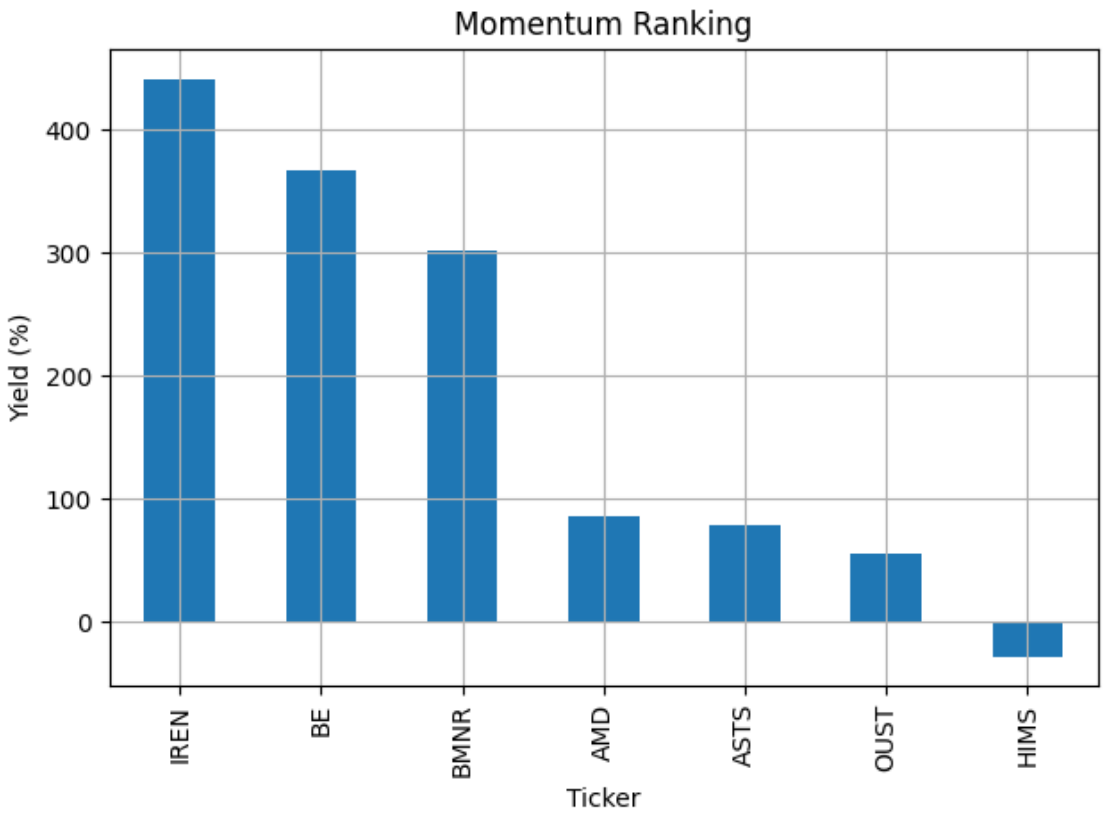
```
Momentum Ranking (Annual yield):

Ticker
IREN    441.79
BE      367.29
```

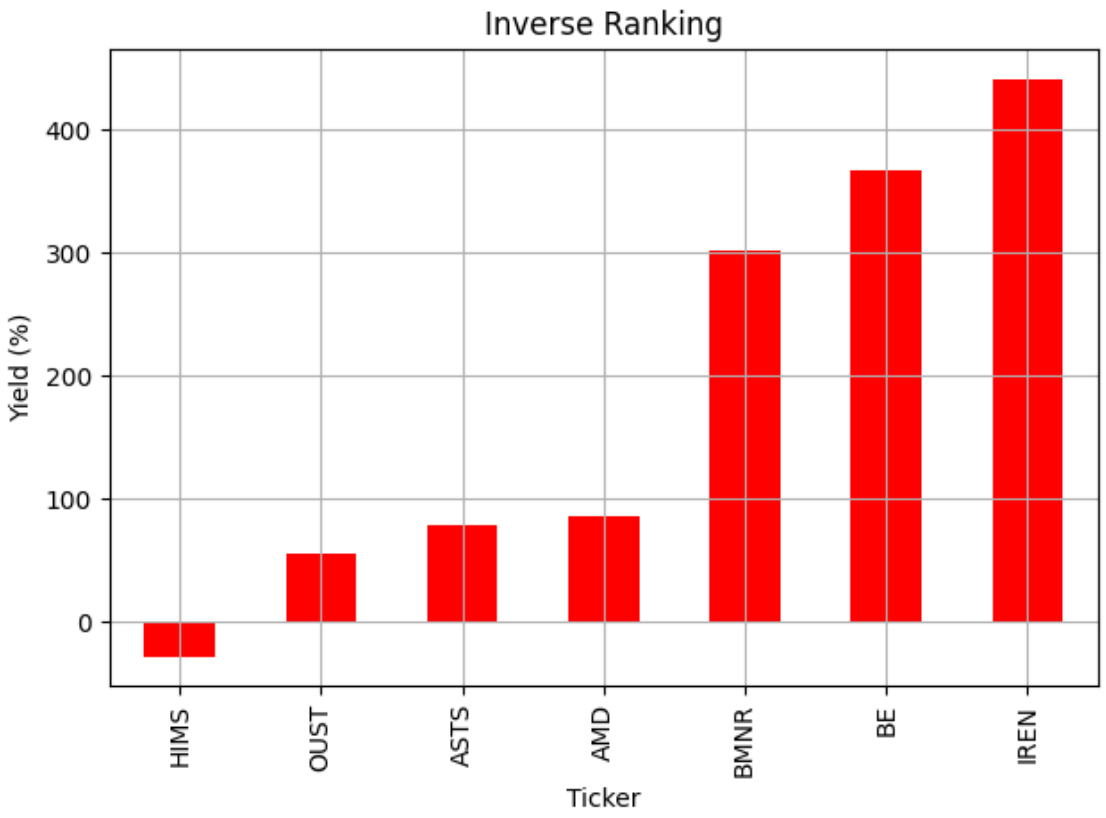
```
BMNR    301.29
AMD      85.88
ASTS     78.28
OUST     55.31
HIMS    -28.41
Name: Final Cumulative Returns (%), dtype: float64
```

[66]:
```
# Inverse Ranking
inverse_ranking = cumulative_returns_final.sort_values(ascending=True)
print("Inverse Momentum Ranking (Annual yield):")
display(inverse_ranking)
```

```
Inverse Momentum Ranking (Annual yield):

Ticker
HIMS    -28.41
OUST     55.31
ASTS     78.28
AMD      85.88
BMNR    301.29
BE      367.29
IREN    441.79
Name: Final Cumulative Returns (%), dtype: float64
```

[67]:
```
# Plot Momentum and Inverse momentum

plt.Figure(figsize=(12, 9))
momentum_ranking.plot(kind='bar', title='Momentum Ranking')
plt.ylabel('Yield (%)')
plt.grid(True)
plt.tight_layout()
plt.show()

plt.Figure(figsize=(12, 9))
inverse_ranking.plot(kind='bar', title='Inverse Ranking', color='red')
plt.ylabel('Yield (%)')
plt.grid(True)
plt.tight_layout()
plt.show()
```


Momentum Ranking
Yield (%)
400
300
200
100
0
IREN
BE
BMNR
AMD
ASTS
OUST
HIMS
Ticker

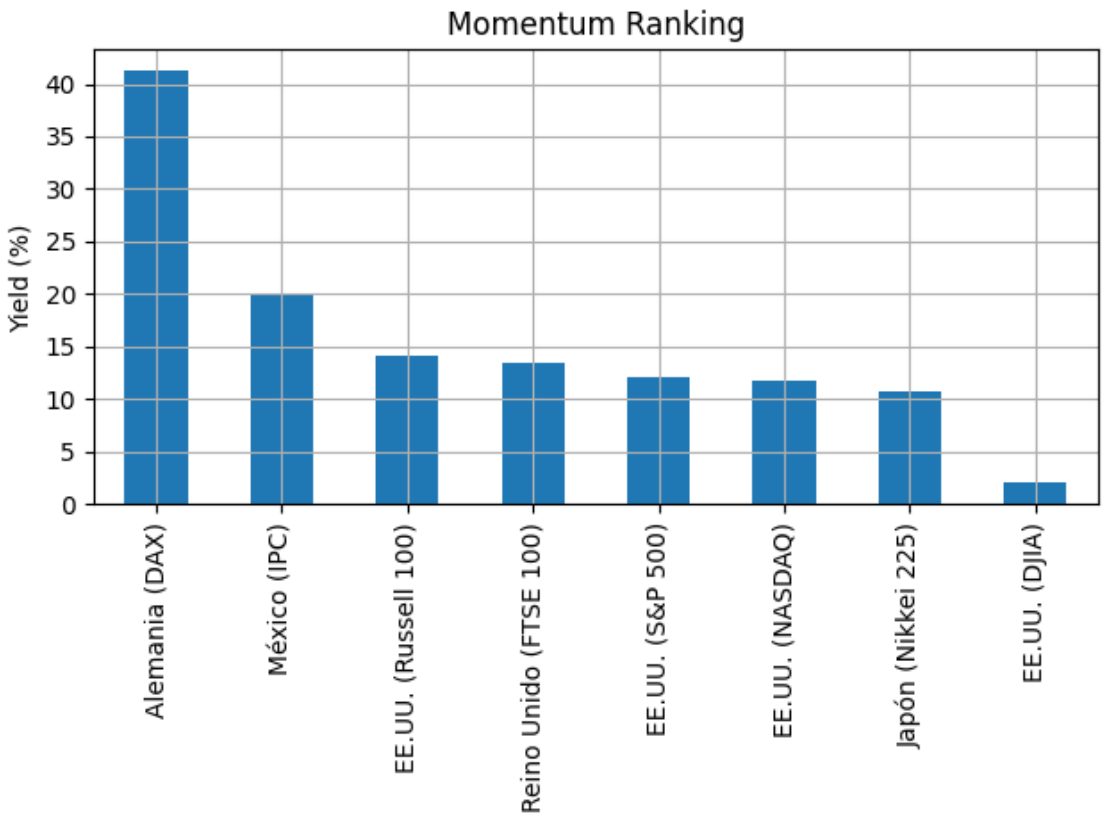


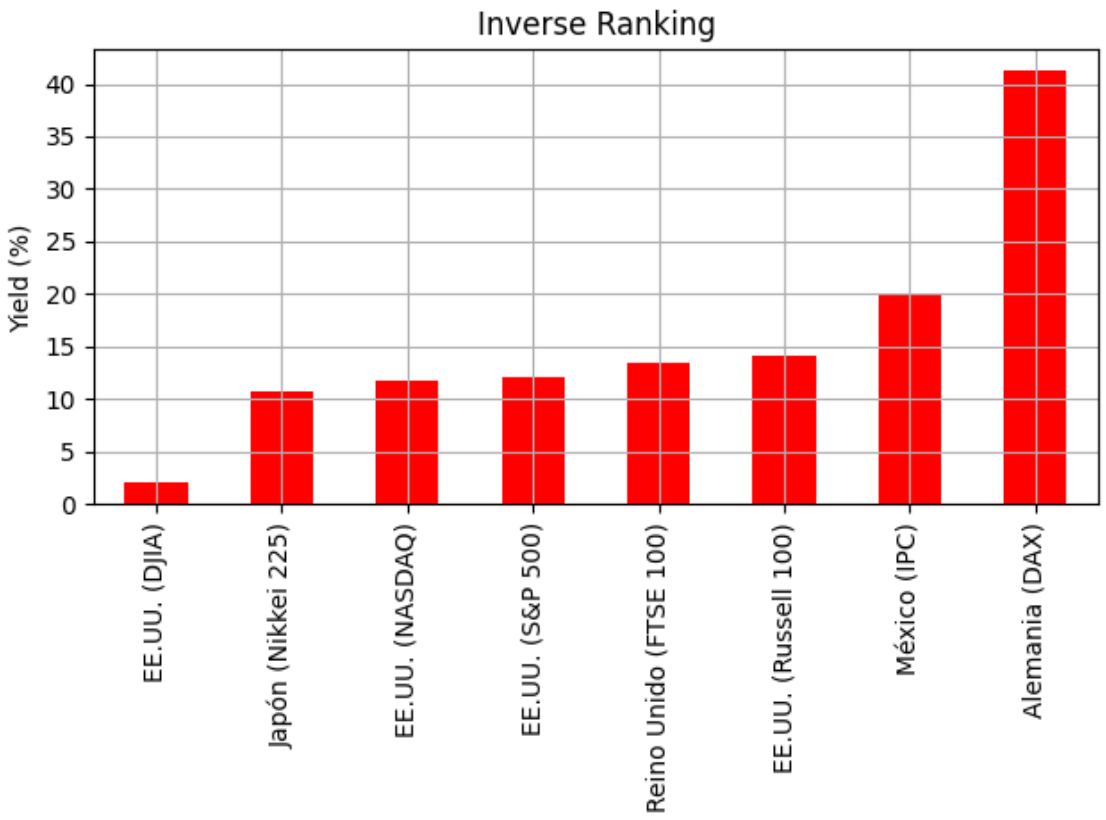
Applied to Indices

[68]:
```
# call the DataFrame of the Final Cumulative Returns, which is the total return␣
↪of each Index in the period.
cumulative_returns_final_bm

# MOMENTUM: Order from largest to lowest return
momentum_ranking_bm = cumulative_returns_final_bm.sort_values(ascending=False)
print("Momentum Ranking (Annual yield) of Benchmark Indices:")
display(momentum_ranking_bm)

# Inverse Ranking
inverse_ranking_bm = cumulative_returns_final_bm.sort_values(ascending=True)
print("Inverse Momentum Ranking (Annual yield) of Benchmark Indices:")
display(inverse_ranking_bm)

# Plot Momentum and Inverse momentum
plt.Figure(figsize=(12, 9))
momentum_ranking_bm.plot(kind='bar', title='Momentum Ranking')
plt.ylabel('Yield (%)')
plt.grid(True)
```

```
plt.tight_layout()
plt.show()

plt.Figure(figsize=(12, 9))
inverse_ranking_bm.plot(kind='bar', title='Inverse Ranking', color='red')
plt.ylabel('Yield (%)')
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
Momentum Ranking (Annual yield) of Benchmark Indices:

Alemania (DAX)            41.192776
México (IPC)              20.069963
EE.UU. (Russell 100)      14.098708
Reino Unido (FTSE 100)    13.517870
EE.UU. (S&P 500)          12.146908
EE.UU. (NASDAQ)           11.840455
Japón (Nikkei 225)        10.740325
EE.UU. (DJIA)              2.184505
Name: final cumulative returns Benchmarks (%), dtype: float64

Inverse Momentum Ranking (Annual yield) of Benchmark Indices:

EE.UU. (DJIA)              2.184505
Japón (Nikkei 225)        10.740325
EE.UU. (NASDAQ)           11.840455
EE.UU. (S&P 500)          12.146908
Reino Unido (FTSE 100)    13.517870
EE.UU. (Russell 100)      14.098708
México (IPC)              20.069963
Alemania (DAX)            41.192776
Name: final cumulative returns Benchmarks (%), dtype: float64
```


Momentum Ranking
Yield (%)
0
5
10
15
20
25
30
35
40
Alemania (DAX)
México (IPC)
EE.UU. (Russell 100)
Reino Unido (FTSE 100)
EE.UU. (S&P 500)
EE.UU. (NASDAQ)
Japón (Nikkei 225)
EE.UU. (DJIA)



16 Correlation

16.1 Correlation matrix

Pearson correlation coefficient:

$$\rho_{X,Y} = \frac{cov(X,Y)}{\sigma_X \sigma_Y} = \frac{E[(X-\mu_X)(Y-\mu_Y)]}{\sigma_X \sigma_Y}$$

$$\sigma_p^2 = w_A^2 \sigma_A^2 \; + \; w_B^2 \sigma_B^2 \; + \; 2\, w_A w_B\, \rho_{AB}\, \sigma_A \sigma_B$$

16.2 Stocks

[69]:
```
# Recall - Daily Returns of the stocks
daily_returns

# Recall - Volatility (Annualized)
volatility_df

# Correlation Matrix
```

```
correlation = daily_returns.corr()

# Covariance (Annualized)
covariance = daily_returns.cov() * 252

# Mean (Annualized)
mean_annualized = daily_returns.mean() * 252

# stats and summary of daily retutns
print(f"Number of days of evaluation: {(today - daily_returns.index[0]).days}")
print(f"since {daily_returns.index[0].date()} until {today.date()}")

print("Correlation Matrix of the period:")
display(correlation.round(2))
print("\nAnnual Volatilities (%):")
display((volatility_df).round(2))
```

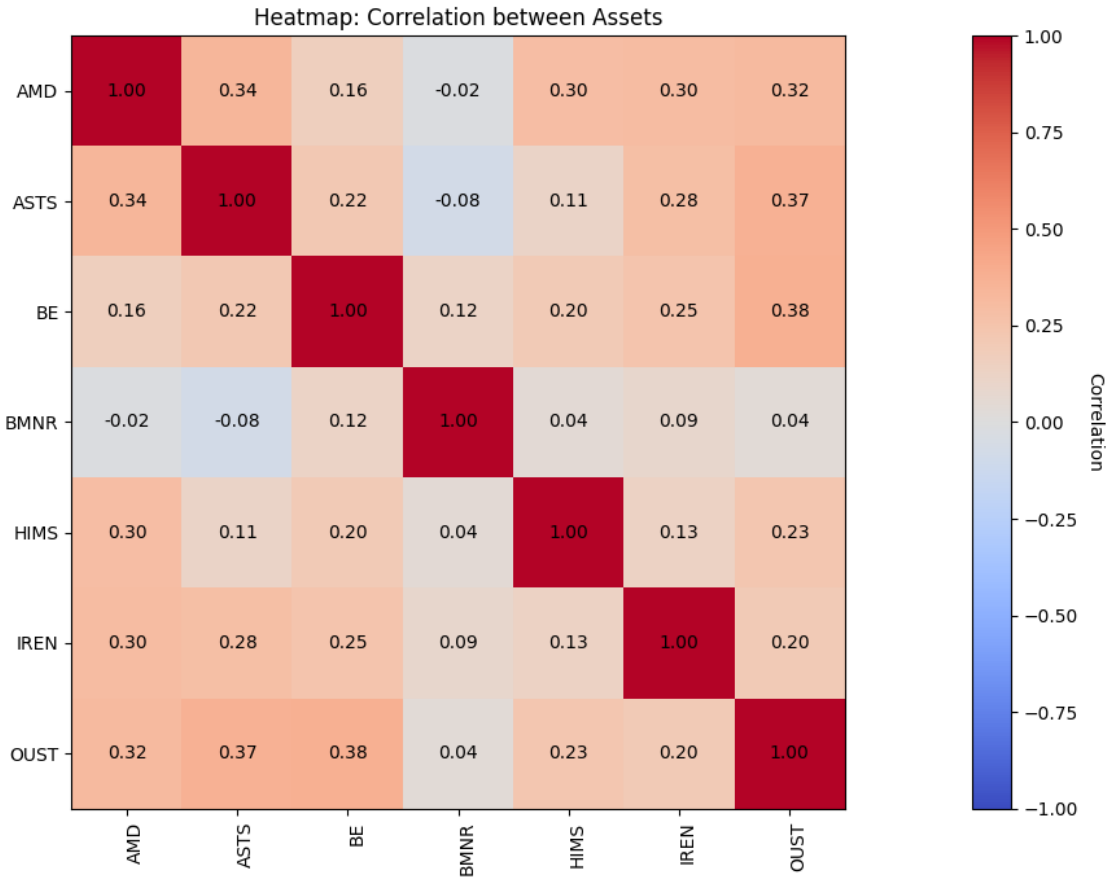
```
Number of days of evaluation: 172
since 2025-06-06 until 2025-11-25
Correlation Matrix of the period:
```

Ticker	AMD	ASTS	BE	BMNR	HIMS	IREN	OUST
Ticker							
AMD	1.00	0.34	0.16	-0.02	0.30	0.30	0.32
ASTS	0.34	1.00	0.22	-0.08	0.11	0.28	0.37
BE	0.16	0.22	1.00	0.12	0.20	0.25	0.38
BMNR	-0.02	-0.08	0.12	1.00	0.04	0.09	0.04
HIMS	0.30	0.11	0.20	0.04	1.00	0.13	0.23
IREN	0.30	0.28	0.25	0.09	0.13	1.00	0.20
OUST	0.32	0.37	0.38	0.04	0.23	0.20	1.00

```
Annual Volatilities (%):
```

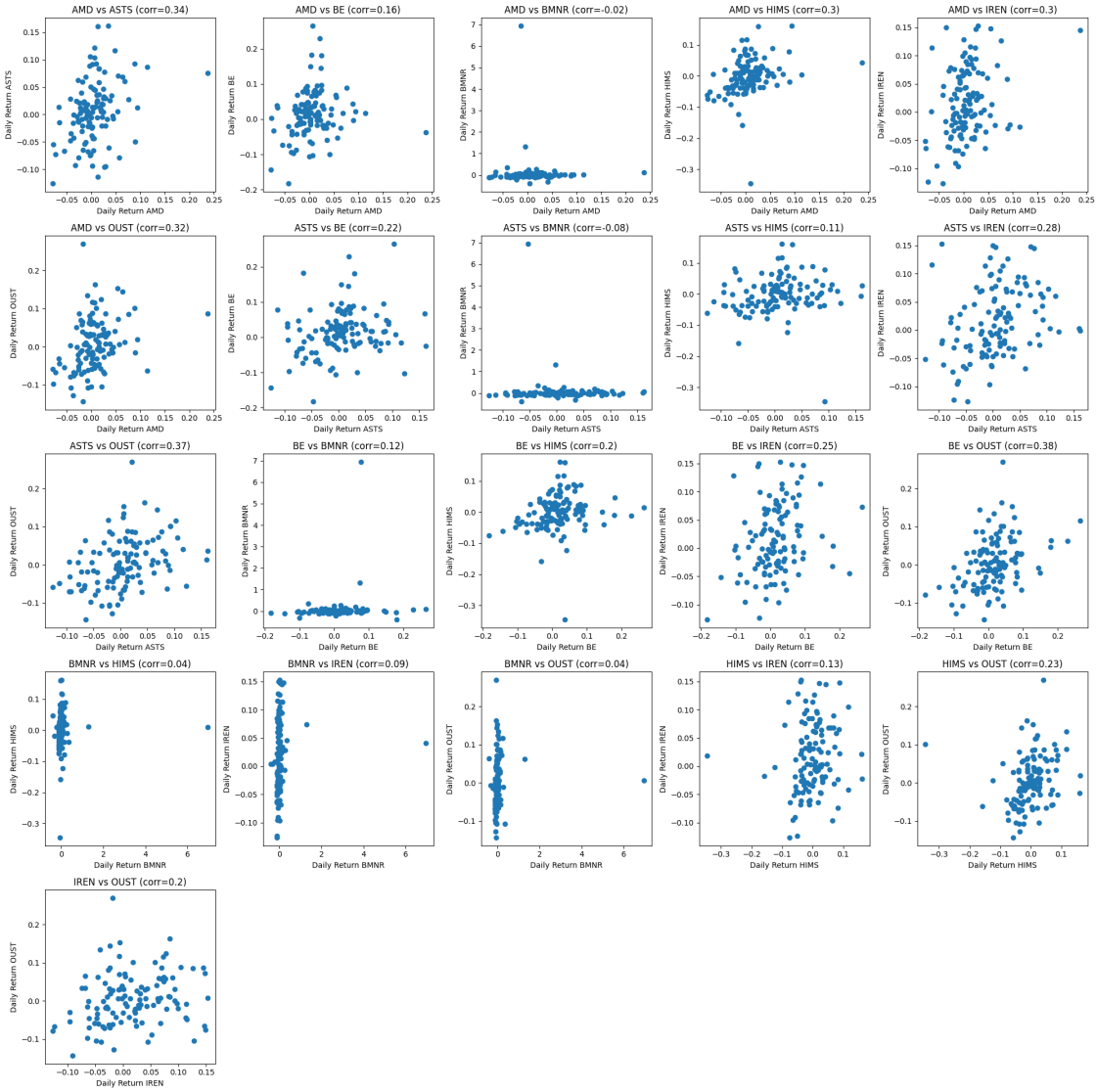
Ticker	Volatility (%)
BMNR	1039.59
OUST	106.72
BE	104.44
IREN	98.02
HIMS	95.13
ASTS	87.52
AMD	63.37

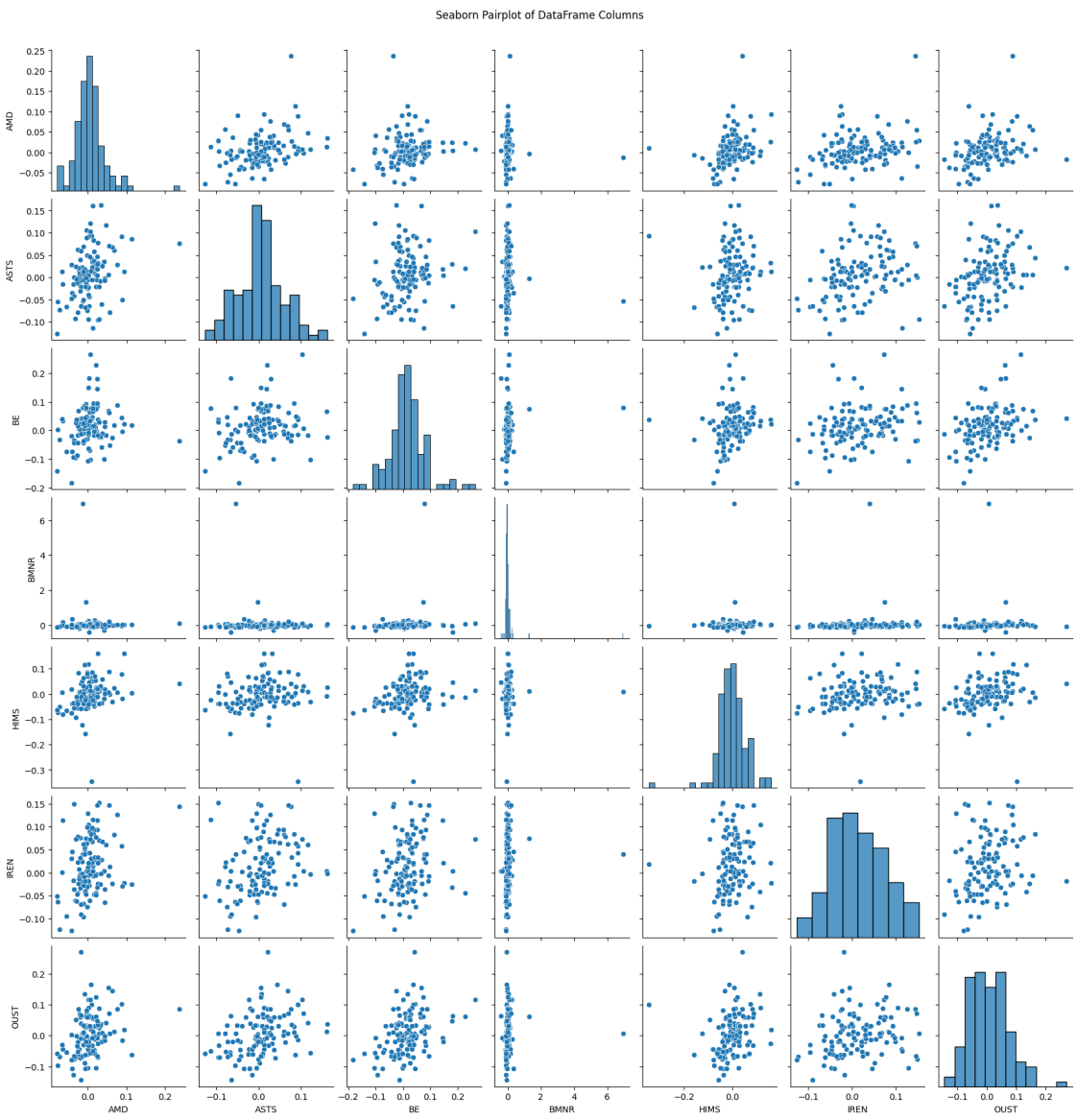
[70]:
```
functions.plot_heatmap_correlation(correlation)
```



[71]:
```
functions.plot_pairs_returns(daily_returns, correlation)
```

```
<string>:42: FutureWarning: Series.__getitem__ treating keys as positions is
deprecated. In a future version, integer keys will always be treated as labels
(consistent with DataFrame behavior). To access a value by position, use
`ser.iloc[pos]`

plot_index: 21
```


AMD vs ASTS (corr=0.34)
Daily Return ASTS
Daily Return AMD
AMD vs BE (corr=0.16)
Daily Return BE
Daily Return AMD
AMD vs BMNR (corr=-0.02)
Daily Return BMNR
Daily Return AMD
AMD vs HIMS (corr=0.3)
Daily Return HIMS
Daily Return AMD
AMD vs IREN (corr=0.3)
Daily Return IREN
Daily Return AMD
AMD vs OUST (corr=0.32)
Daily Return OUST
Daily Return AMD
ASTS vs BE (corr=0.22)
Daily Return BE
Daily Return ASTS
ASTS vs BMNR (corr=-0.08)
Daily Return BMNR
Daily Return ASTS
ASTS vs HIMS (corr=0.11)
Daily Return HIMS
Daily Return ASTS
ASTS vs IREN (corr=0.28)
Daily Return IREN
Daily Return ASTS
ASTS vs OUST (corr=0.37)
Daily Return OUST
Daily Return ASTS
BE vs BMNR (corr=0.12)
Daily Return BMNR
Daily Return BE
BE vs HIMS (corr=0.2)
Daily Return HIMS
Daily Return BE
BE vs IREN (corr=0.25)
Daily Return IREN
Daily Return BE
BE vs OUST (corr=0.38)
Daily Return OUST
Daily Return BE
BMNR vs HIMS (corr=0.04)
Daily Return HIMS
Daily Return BMNR
BMNR vs IREN (corr=0.09)
Daily Return IREN
Daily Return BMNR
BMNR vs OUST (corr=0.04)
Daily Return OUST
Daily Return BMNR
HIMS vs IREN (corr=0.13)
Daily Return IREN
Daily Return HIMS
HIMS vs OUST (corr=0.23)
Daily Return OUST
Daily Return HIMS
IREN vs OUST (corr=0.2)
Daily Return OUST
Daily Return IREN



(above) WE look for low linearity in the relations, plots with vertical shapes could mean low correlation.

16.3 Indices & Portfolio

[72]:
```
# Recall - Daily Returns of the Benchmark Inidices and Portfolio
merge_daily_returns

# Recall - Volatility (Annualized) Benchmark Inidices and Portfolio
merge_annualized_volatility

# Correlation Matrix
```

```
merge_correlation = merge_daily_returns.corr()

# Covariance (Annualized)
merge_covariance = merge_daily_returns.cov() * 252

# Mean (Annualized)
merge_mean_annualized = merge_daily_returns.mean() * 252

# stats and summary of daily retutns
print(f"Number of days of evaluation: {(today - merge_daily_returns.index[0]).
  ↪days}")
print(f"since {merge_daily_returns.index[0].date()} until {today.date()}")

print("Correlation Matrix of the period:")
display(merge_correlation.round(2))
print("\nAnnual Volatilities (%):")
display((merge_annualized_volatility).round(2))
```

```
Number of days of evaluation: 172
since 2025-06-06 until 2025-11-25
Correlation Matrix of the period:
```

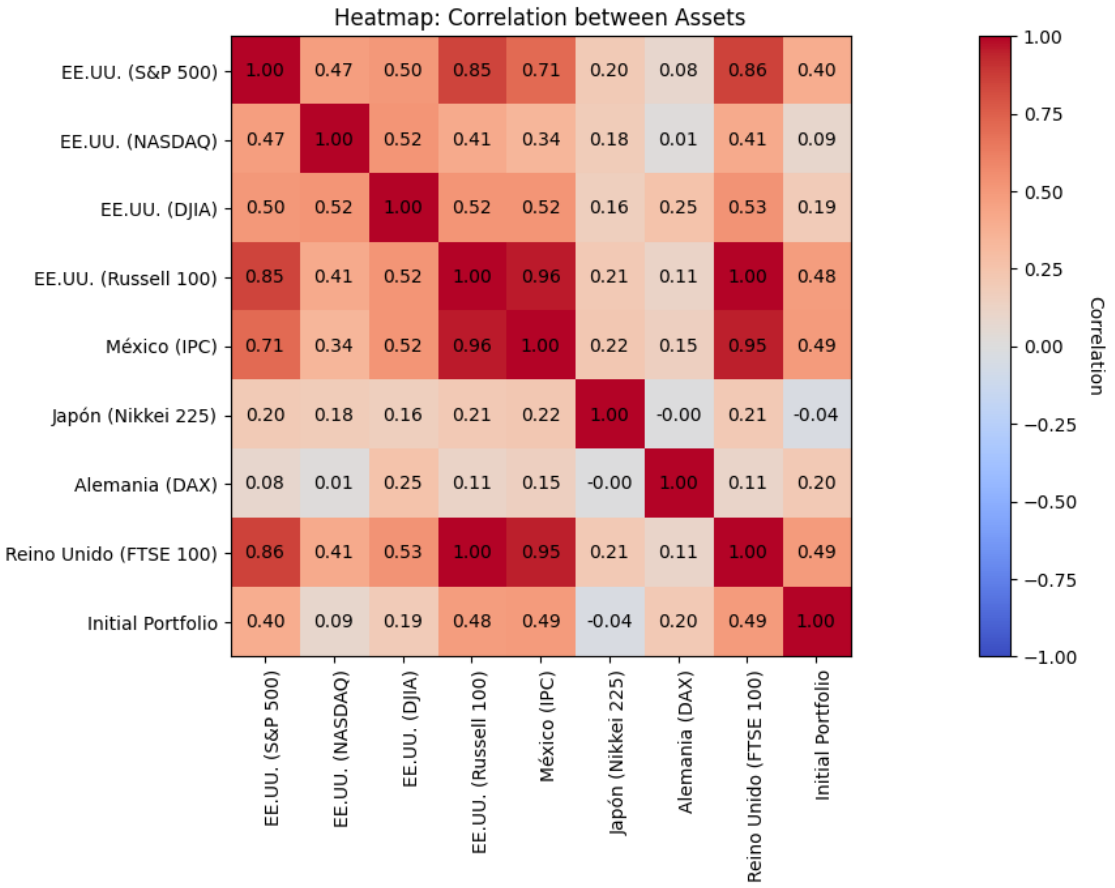
	EE.UU. (S&P 500)	EE.UU. (NASDAQ)	EE.UU. (DJIA)	\
EE.UU. (S&P 500)	1.00	0.47	0.50	
EE.UU. (NASDAQ)	0.47	1.00	0.52	
EE.UU. (DJIA)	0.50	0.52	1.00	
EE.UU. (Russell 100)	0.85	0.41	0.52	
México (IPC)	0.71	0.34	0.52	
Japón (Nikkei 225)	0.20	0.18	0.16	
Alemania (DAX)	0.08	0.01	0.25	
Reino Unido (FTSE 100)	0.86	0.41	0.53	
Initial Portfolio	0.40	0.09	0.19	

	EE.UU. (Russell 100)	México (IPC)	\
EE.UU. (S&P 500)	0.85	0.71	
EE.UU. (NASDAQ)	0.41	0.34	
EE.UU. (DJIA)	0.52	0.52	
EE.UU. (Russell 100)	1.00	0.96	
México (IPC)	0.96	1.00	
Japón (Nikkei 225)	0.21	0.22	
Alemania (DAX)	0.11	0.15	
Reino Unido (FTSE 100)	1.00	0.95	
Initial Portfolio	0.48	0.49	

	Japón (Nikkei 225)	Alemania (DAX)	\
EE.UU. (S&P 500)	0.20	0.08	
EE.UU. (NASDAQ)	0.18	0.01	
EE.UU. (DJIA)	0.16	0.25	

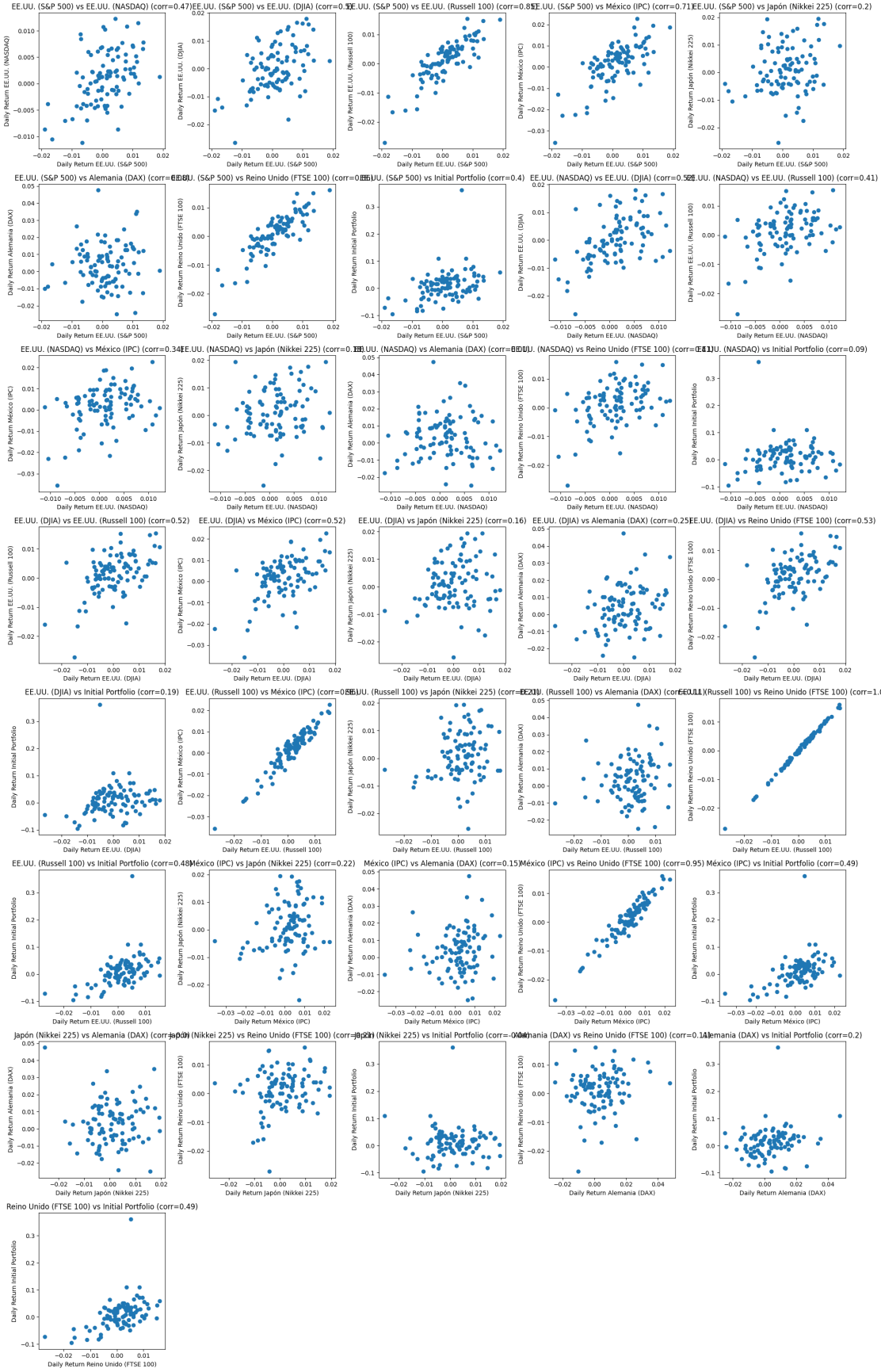
```
EE.UU. (Russell 100)                 0.21           0.11
México (IPC)                         0.22           0.15
Japón (Nikkei 225)                   1.00          -0.00
Alemania (DAX)                      -0.00           1.00
Reino Unido (FTSE 100)               0.21           0.11
Initial Portfolio                   -0.04           0.20

                      Reino Unido (FTSE 100)  Initial Portfolio
EE.UU. (S&P 500)                        0.86               0.40
EE.UU. (NASDAQ)                         0.41               0.09
EE.UU. (DJIA)                           0.53               0.19
EE.UU. (Russell 100)                    1.00               0.48
México (IPC)                            0.95               0.49
Japón (Nikkei 225)                      0.21              -0.04
Alemania (DAX)                          0.11               0.20
Reino Unido (FTSE 100)                  1.00               0.49
Initial Portfolio                       0.49               1.00


Annual Volatilities (%):

Initial Portfolio         85.74
Alemania (DAX)            19.67
México (IPC)              15.06
Japón (Nikkei 225)        13.29
EE.UU. (DJIA)             13.01
Reino Unido (FTSE 100)    11.13
EE.UU. (Russell 100)      10.97
EE.UU. (S&P 500)          10.93
EE.UU. (NASDAQ)            7.90
Name: Annualized Volatility (%), dtype: float64
```

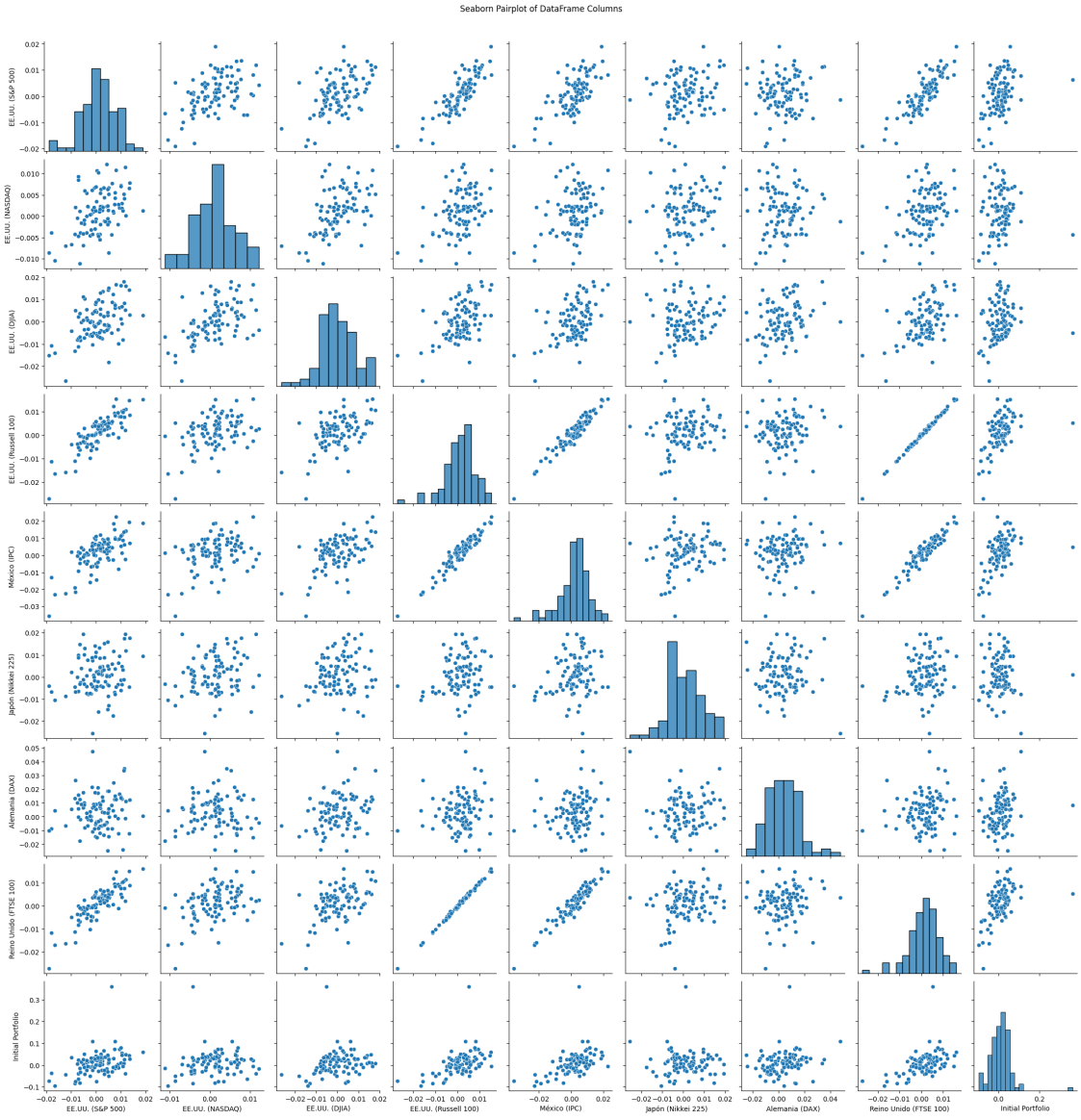
[73]:
```
functions.plot_heatmap_correlation(merge_correlation)
```



[74]:
```
functions.plot_pairs_returns(merge_daily_returns, merge_correlation)
```

```
<string>:42: FutureWarning: Series.__getitem__ treating keys as positions is
deprecated. In a future version, integer keys will always be treated as labels
(consistent with DataFrame behavior). To access a value by position, use
`ser.iloc[pos]`

plot_index: 36
```


EE.UU. (S&P 500) vs EE.UU. (NASDAQ) (corr=0.47)
EE.UU. (S&P 500) vs EE.UU. (DJIA) (corr=0.5
EE.UU. (S&P 500) vs EE.UU. (Russell 100) (corr=0.8
EE.UU. (S&P 500) vs México (IPC) (corr=0.71)
EE.UU. (S&P 500) vs Japón (Nikkei 225) (corr=0.2)
EE.UU. (S&P 500) vs Alemania (DAX) (corr=0.6
EE.UU. (S&P 500) vs Reino Unido (FTSE 100) (corr=0.8
EE.UU. (S&P 500) vs Initial Portfolio (corr=0.4)
EE.UU. (NASDAQ) vs EE.UU. (DJIA) (corr=0.5
EE.UU. (NASDAQ) vs EE.UU. (Russell 100) (corr=0.41)
EE.UU. (NASDAQ) vs México (IPC) (corr=0.34
EE.UU. (NASDAQ) vs Japón (Nikkei 225) (corr=0.1
EE.UU. (NASDAQ) vs Alemania (DAX) (corr=0.1
EE.UU. (NASDAQ) vs Reino Unido (FTSE 100) (corr=0.1
EE.UU. (NASDAQ) vs Initial Portfolio (corr=0.09)
EE.UU. (DJIA) vs EE.UU. (Russell 100) (corr=0.52)
EE.UU. (DJIA) vs México (IPC) (corr=0.52)
EE.UU. (DJIA) vs Japón (Nikkei 225) (corr=0.16)
EE.UU. (DJIA) vs Alemania (DAX) (corr=0.25)
EE.UU. (DJIA) vs Reino Unido (FTSE 100) (corr=0.53)
EE.UU. (DJIA) vs Initial Portfolio (corr=0.19)
EE.UU. (Russell 100) vs México (IPC) (corr=0.85)
EE.UU. (Russell 100) vs Japón (Nikkei 225) (corr=0.2
EE.UU. (Russell 100) vs Alemania (DAX) (corr=0.11
EE.UU. (Russell 100) vs Reino Unido (FTSE 100) (corr=1.0)
EE.UU. (Russell 100) vs Initial Portfolio (corr=0.48)
México (IPC) vs Japón (Nikkei 225) (corr=0.22)
México (IPC) vs Alemania (DAX) (corr=0.15)
México (IPC) vs Reino Unido (FTSE 100) (corr=0.95)
México (IPC) vs Initial Portfolio (corr=0.49)
Japón (Nikkei 225) vs Alemania (DAX) (corr=0.0
Japón (Nikkei 225) vs Reino Unido (FTSE 100) (corr=0.22
Japón (Nikkei 225) vs Initial Portfolio (corr=-0.04
Alemania (DAX) vs Reino Unido (FTSE 100) (corr=0.14
Alemania (DAX) vs Initial Portfolio (corr=0.2)
Reino Unido (FTSE 100) vs Initial Portfolio (corr=0.49)



(above) WE look for low linearity in the relations, plots with vertical shapes could mean low correlation.

17 Covariance

- La matriz de covarianza es la base para calcular la varianza del portafolio:

$$\sigma_p^2 = w^T \Sigma w$$

donde: * w = vector de pesos del portafolio. * Σ = *matriz* de covarianzas. * Un gestor de portafolios busca combinaciones de activos con **baja covarianza** para reducir la volatilidad total manteniendo el rendimiento esperado.

- Covariance:

$$\text{Cov}(X, Y) = E[(X - \mu_X)(Y - \mu_Y)]$$

- Values:
  - Positive → one rises, other rises too.
  - Negative → one rises, other goes down.
  - near 0 → there's no clear relationtip.

17.1 Stocks

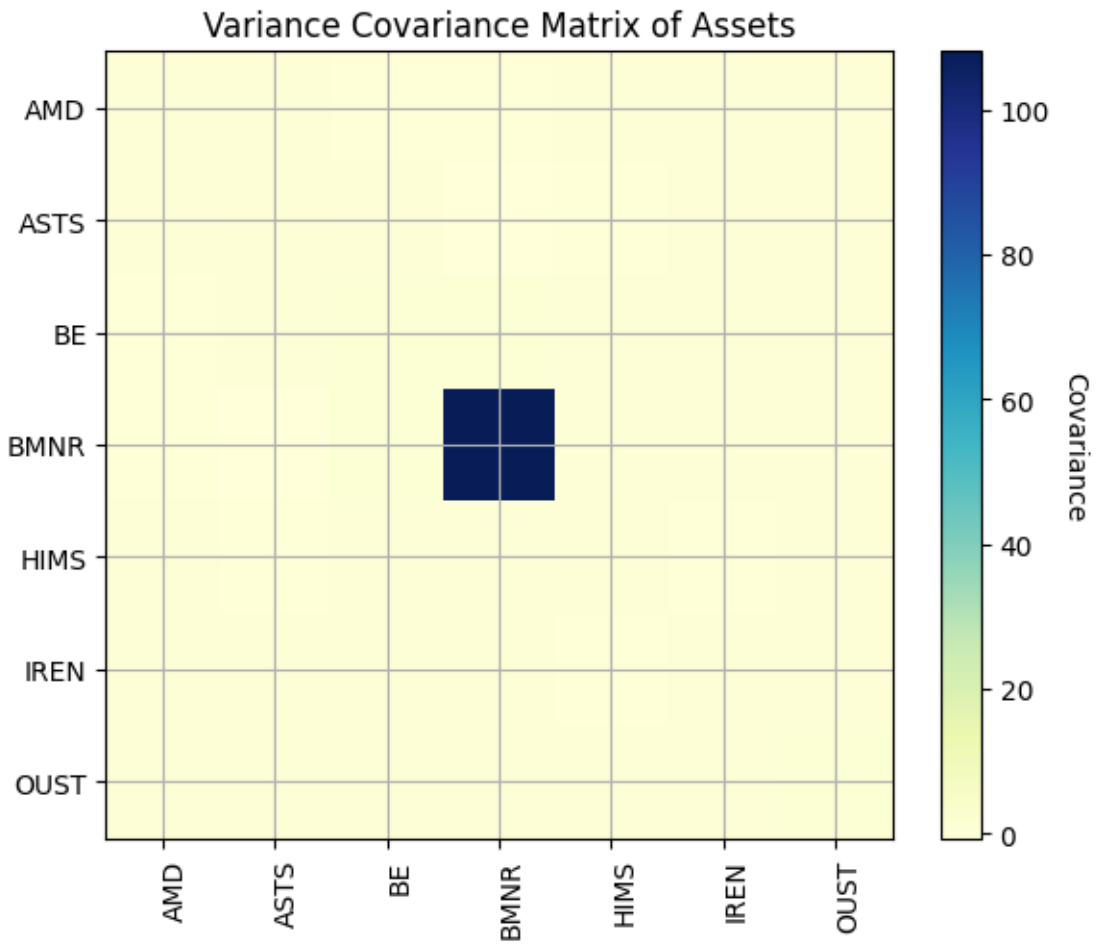
[75]:
```
# Recall Variance Covariance Matrix of Stocks
print("Variance - Covariance Matrix of Stocks")
covariance
```

```
Variance - Covariance Matrix of Stocks
```

[75]:

Ticker	AMD	ASTS	BE	BMNR	HIMS	IREN	OUST
Ticker							
AMD	0.401630	0.187778	0.105625	-0.146724	0.177952	0.185965	0.213279
ASTS	0.187778	0.765972	0.197191	-0.685755	0.092569	0.243628	0.344382
BE	0.105625	0.197191	1.090754	1.278523	0.199400	0.254933	0.425998
BMNR	-0.146724	-0.685755	1.278523	108.074461	0.388136	0.883440	0.426635
HIMS	0.177952	0.092569	0.199400	0.388136	0.904950	0.120542	0.235195
IREN	0.185965	0.243628	0.254933	0.883440	0.120542	0.960861	0.207926
OUST	0.213279	0.344382	0.425998	0.426635	0.235195	0.207926	1.138900

[76]:
```
functions.plot_heatmap_covariance(covariance, False)
```



17.2 Indices & Portfolio

[77]:
```
# Recall Variance Covariance Matrix of Indices and Portfolio
print("Variance - Covariance Matrix of Indices and Portfolio")
merge_covariance
```

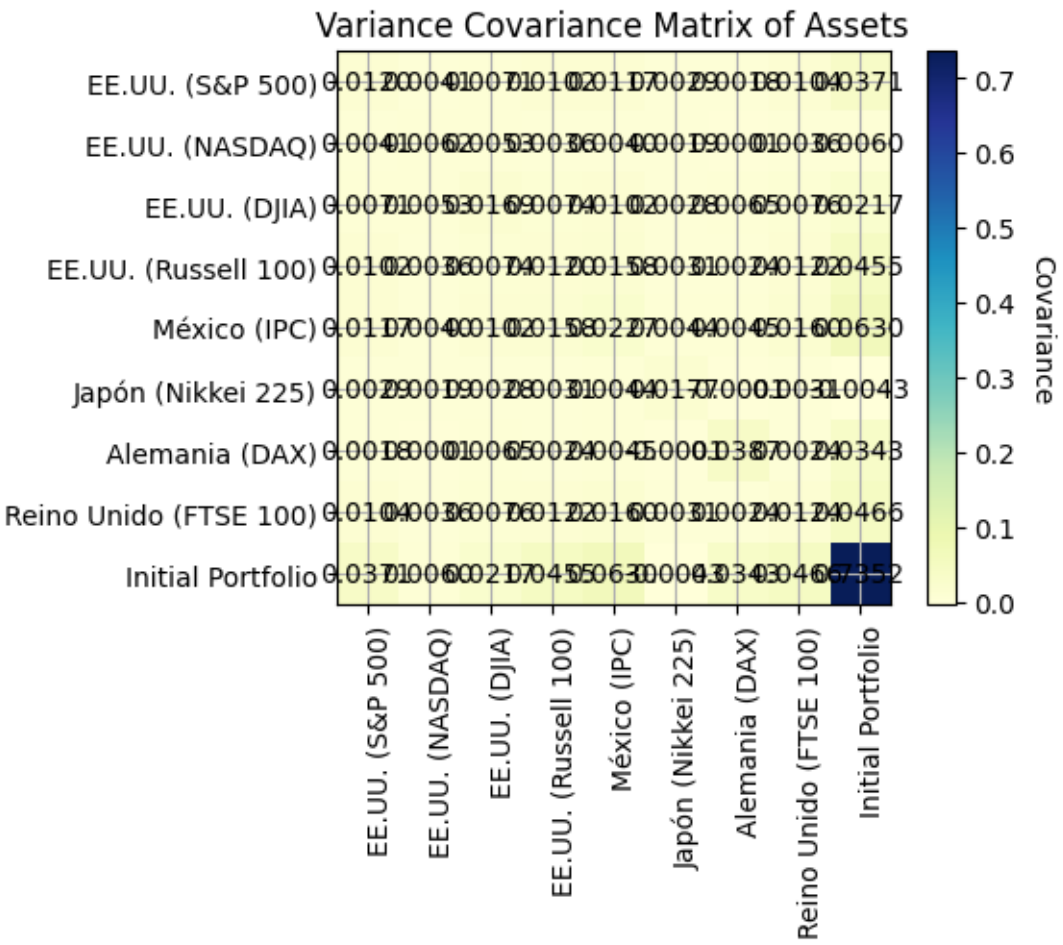
```
Variance - Covariance Matrix of Indices and Portfolio
```

[77]:

	EE.UU. (S&P 500)	EE.UU. (NASDAQ)	EE.UU. (DJIA)	\
EE.UU. (S&P 500)	0.011956	0.004073	0.007114	
EE.UU. (NASDAQ)	0.004073	0.006244	0.005320	
EE.UU. (DJIA)	0.007114	0.005320	0.016913	
EE.UU. (Russell 100)	0.010158	0.003576	0.007436	
México (IPC)	0.011672	0.004023	0.010219	
Japón (Nikkei 225)	0.002863	0.001943	0.002770	
Alemania (DAX)	0.001769	0.000142	0.006463	
Reino Unido (FTSE 100)	0.010419	0.003589	0.007619	
Initial Portfolio	0.037133	0.006019	0.021701	

```
                        EE.UU. (Russell 100)  México (IPC)  \
EE.UU. (S&P 500)                    0.010158      0.011672
EE.UU. (NASDAQ)                     0.003576      0.004023
EE.UU. (DJIA)                       0.007436      0.010219
EE.UU. (Russell 100)                0.012035      0.015808
México (IPC)                        0.015808      0.022679
Japón (Nikkei 225)                  0.003052      0.004449
Alemania (DAX)                      0.002370      0.004531
Reino Unido (FTSE 100)              0.012185      0.015965
Initial Portfolio                   0.045456      0.063008

                        Japón (Nikkei 225)  Alemania (DAX)  \
EE.UU. (S&P 500)                  0.002863        0.001769
EE.UU. (NASDAQ)                   0.001943        0.000142
EE.UU. (DJIA)                     0.002770        0.006463
EE.UU. (Russell 100)              0.003052        0.002370
México (IPC)                      0.004449        0.004531
Japón (Nikkei 225)                0.017670       -0.000126
Alemania (DAX)                   -0.000126        0.038700
Reino Unido (FTSE 100)            0.003109        0.002388
Initial Portfolio                -0.004310        0.034321

                        Reino Unido (FTSE 100)  Initial Portfolio
EE.UU. (S&P 500)                      0.010419           0.037133
EE.UU. (NASDAQ)                       0.003589           0.006019
EE.UU. (DJIA)                         0.007619           0.021701
EE.UU. (Russell 100)                  0.012185           0.045456
México (IPC)                          0.015965           0.063008
Japón (Nikkei 225)                    0.003109          -0.004310
Alemania (DAX)                        0.002388           0.034321
Reino Unido (FTSE 100)                0.012383           0.046647
Initial Portfolio                     0.046647           0.735169
```

[78]:
```
functions.plot_heatmap_covariance(merge_covariance, True)
```



18 Efficient Frontier (Sharpe Ratio)

$$Sharpe = \frac{R_p - R_f}{\sigma_p} = \frac{\mu_p - r_f}{\sigma_p}$$

Where: * R_p: Expected Portfolio Return, μ * R_f: Risk Free Rate (can be 0 if ignored) * σ_p: Portfolio Risk (StdDev)

Long-only: weights $w_i \in [0, 1]$ y $\sum w_i = 1$.

- Only purchases, no leverage (by shorting)
- Pros: Less operating risk
- Cons: Frontier less efficient than with shorts (due leverage)

Shorts allowed: weights $w_i \in [-1, 1]$ y $\sum w_i = 1$ (or similar).

- One can short sell and compensate with long positions
- Pros: Theoretical improvement of the Frontier with more options
- Cons:: Implied leverage, Margin requirements, Costs and Operational Risk

Number of starts: To avoid local minimas. 10-25 normal (fast and robust), 50-100 (for large amount of stocks)

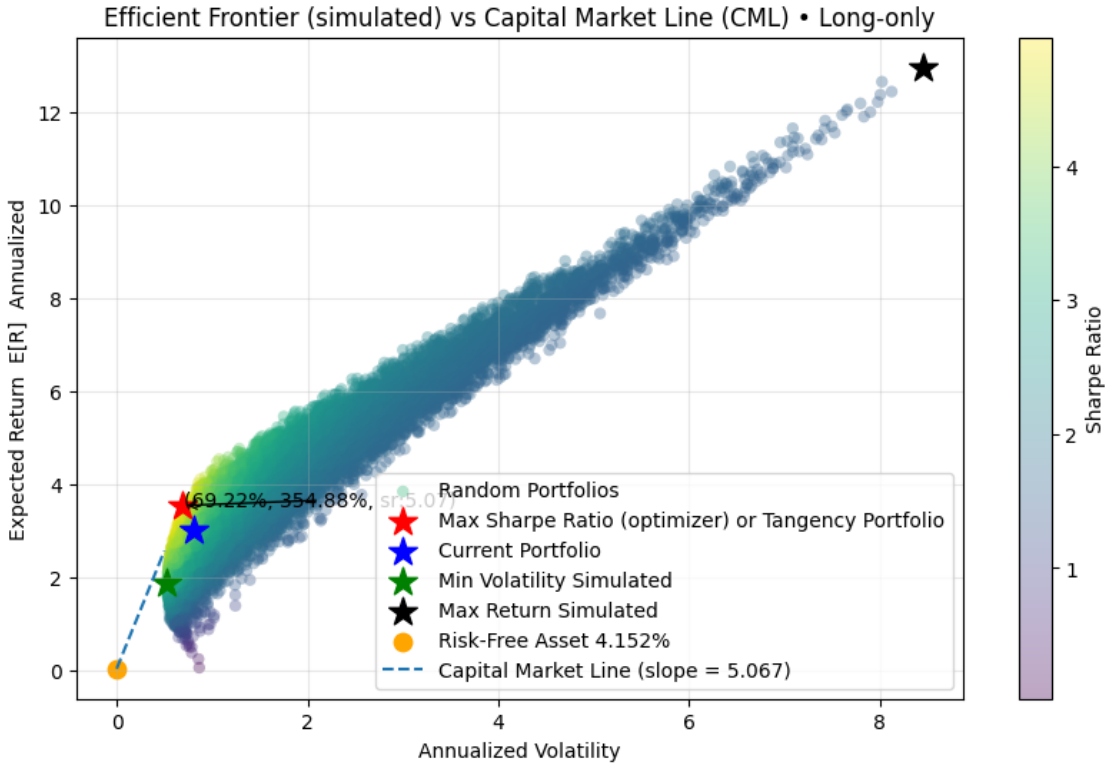
[79]:
```
no_starts = 100
no_simul = 20000
```

18.1 Stocks

[80]:
```
weights_optimal, vol_ret_sr_optimal = functions.efficient_froentier_sharp_ratio(
                daily_returns,
                [portfolio_annualized_volatility, portfolio_annualized_return],
                mean_annualized,
                covariance,
                risk_free, #Risk Free
                True, # Long only = True, Short allowed = False
                no_starts, # no. of starts (25 default)
                no_simul, # no. of simulations
                123 # seed
                )
```

```
Optimizing Sharpe…

Optimum Weights (%) - Tangency Portfolio
Ticker
AMD     20.91
ASTS     1.61
BE      33.43
BMNR     1.24
HIMS     0.00
IREN    42.81
OUST     0.00

Optimum Sharpe Ratio: 5.067
Expected Annual Return: 354.88%
Annual Risk: 69.22%

Simulating random Portfolios…

Simulated Portfolio with Minimum Volatility:
Volatility: 52.56%
Return: 186.84%
Sharpe Ratio: 3.475

Simulated Portfolio with Maximum Return:
Volatility: 844.77%
Return: 1296.60%
Sharpe Ratio: 1.530
```

```
Current Portfolio:
Volatility: 81.50%
Return: 301.54%
Sharpe Ratio: 3.649
```



18.1.1 TO-DO:

- to bring the positions of each of the other key portfolios (max, min)

18.1.2 Positions from current to optimal

[81]:
```
# Recall Total Invested
Investment = Total_invested

# Current Portolio Weights and Positions
weights_df

# Optimal Weights from Efficient Frontier
weights_optimal

# Merge DataFrames
```

```
weights_df_Optimal = pd.merge(weights_df, weights_optimal, left_index=True,␣
 ↪right_index=True, how='outer')

# New Investment (USD)
weights_df_Optimal['New Investment usd'] = weights_df_Optimal['Optimal␣
 ↪Weights'] * Investment

# New QTY
weights_df_Optimal['New QTY'] = weights_df_Optimal['New Investment usd'] /␣
 ↪weights_df_Optimal['Close Price']

# Difference in QTY or buy/sell position
weights_df_Optimal['Position_to_Optimal'] = weights_df_Optimal['New QTY'] -␣
 ↪weights_df_Optimal['Current QTY']

weights_df_Optimal = round(weights_df_Optimal, 3)
display(weights_df_Optimal)

print(f"Current Investment: $ {round(Investment, 2)}")
print(f"New Investment: $ {round(weights_df_Optimal['New Investment usd'].
 ↪sum(), 2)}")
```

Ticker	Current QTY	Close Price	Investment	Weights	Optimal Weights	\
AMD	50.28	215.05	10811.87	0.11	0.209	
ASTS	201.13	55.00	11062.36	0.11	0.016	
BE	69.28	95.56	6620.74	0.07	0.334	
BMNR	146.64	31.10	4560.43	0.05	0.012	
HIMS	282.47	37.78	10671.68	0.11	0.000	
IREN	638.06	48.49	30939.77	0.32	0.428	
OUST	1100.79	21.37	23523.80	0.24	0.000	

Ticker	New Investment usd	New QTY	Position_to_Optimal
AMD	20531.666	95.474	45.194
ASTS	1580.870	28.743	-172.387
BE	32825.135	343.503	274.223
BMNR	1217.564	39.150	-107.490
HIMS	0.000	0.000	-282.470
IREN	42035.419	866.888	228.828
OUST	0.000	0.000	-1100.790

```
Current Investment: $ 98190.65
New Investment: $ 98190.65
```

18.2 Including Risk Free Asset

Once you have the tangency portfolio (long-only), the fraction of your wealth to put into the risky portfolio is .

Choose based on your target volatility, target return, or risk-aversion: * Objective 1. Target Volatility:

$$y = \frac{\sigma_p}{\sigma_t}$$

- Objective 2. Target Return:

$$y = \frac{E_p - R_f}{E[R_t] - R_f}$$

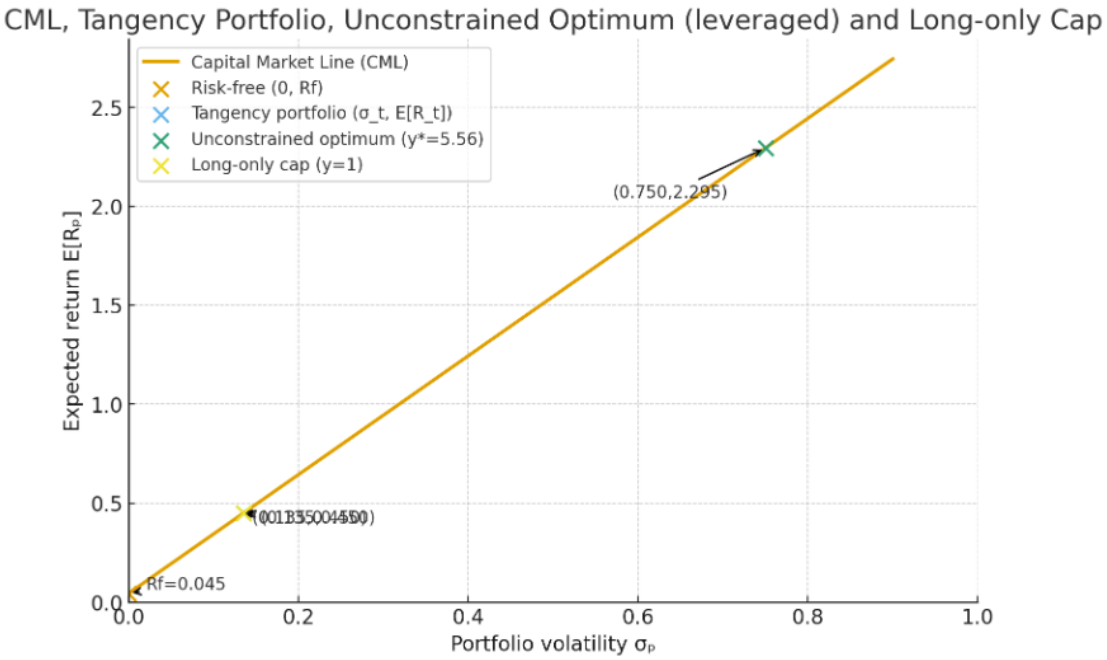
- Objective 3. Mean-Variance Utility (Expected Utility Funcion):

$$U = E[R_p] - \frac{1}{2}\gamma\sigma_p^2$$

$$y* = \frac{E[R_t] - R_f}{\gamma\,\sigma_t^2}$$

U : "Utility" — a scalar number representing the investor's satisfaction from a portfolio. Its a parabolic function, higher gama the steeper the curve, thus more return expected for unit of risk.

> 0: Risk aversion coefficient, a measure of how strongly the investor dislikes risk. Penalizes risk. Higher->more conservative.



Finally: invest in _ and (1−) in the risk-free asset, enforcing 0 1 for long-only/no-borrowing.

[82]:
```
# Objective 1. You want a target portfolio volatility   :

#   (target volatility):
# 90% of the Tangency portfolio volatility (can be any value)
target_volatility = vol_ret_sr_optimal[0] * 0.90

# The fraction of total wealth X invested in the tangency (risky) portfolio
y = target_volatility / vol_ret_sr_optimal[0]

print(f"Target Volatility: {target_volatility:.2%}")
print(f"The fraction of total wealth X invested in the tangency (risky)␣
 ↪portfolio 'y' is: {y:.2%}, that is ${y*Total_invested:,.2f}usd")
print(f"The fraction invested in the risk-free asset (1-y) is: {1-y:.2%}, that␣
 ↪is ${(1-y)*Total_invested:,.2f}usd")
ER_p = risk_free + y*(vol_ret_sr_optimal[1] - risk_free)
print(f"Expected Return E[Rp] = {ER_p:.2%}")
print(f"Sharpe Ratio = {(ER_p - risk_free) / target_volatility:.2f} ")
```

```
Target Volatility: 62.29%
The fraction of total wealth X invested in the tangency (risky) portfolio 'y'
is: 90.00%, that is $88,371.59usd
The fraction invested in the risk-free asset (1-y) is: 10.00%, that is
$9,819.07usd
Expected Return E[Rp] = 319.81%
Sharpe Ratio = 5.07
```

[83]:
```
# Objective 2. You want a target expected return Ep

# Ep (target Expected Return):
# 90% of the Tangency portfolio return (can be any value)
target_return = vol_ret_sr_optimal[1] * 0.90

# The fraction of total wealth X invested in the tangency (risky) portfolio
y = (target_return - risk_free) / (vol_ret_sr_optimal[1] - risk_free)

print(f"Target Expected Return: {target_return:.2%}")
print(f"The fraction of total wealth X invested in the tangency (risky)␣
 ↪portfolio 'y' is: {y:.2%}, that is ${y*Total_invested:,.2f}usd")
print(f"The fraction invested in the risk-free asset (1-y) is: {1-y:.2%}, that␣
 ↪is ${(1-y)*Total_invested:,.2f}usd")
sigma_p = y * vol_ret_sr_optimal[0]
print(f"Volatility: {sigma_p:.2%}")
print(f"Sharpe Ratio = {(target_return - risk_free)/sigma_p:.2f}")
```

```
Target Expected Return: 319.39%
The fraction of total wealth X invested in the tangency (risky) portfolio 'y'
```

```
is: 89.88%, that is $88,255.35usd
The fraction invested in the risk-free asset (1-y) is: 10.12%, that is
$9,935.31usd
Volatility: 62.21%
Sharpe Ratio = 5.07
```

[84]:
```python
# Objective 3. You maximize mean-variance Utility (risk aversion )

# Gamma (risk aversion coefficient) :
# If   is large, the investor is very risk averse - even small increases in␣
 ↪variance are penalized heavily.
# → They prefer portfolios with lower volatility, even if returns are modest.
# If   is small, the investor is risk tolerant (or aggressive) - they are␣
 ↪willing to accept more variance for more expected return.
gamma = 4

# a) Allowing borrowing (short)
# The fraction of total wealth X invested in the tangency (risky) portfolio
y_star = (vol_ret_sr_optimal[1] - risk_free) / (gamma *␣
 ↪(vol_ret_sr_optimal[0]**2))
print("a) Allowing borrowing (short in risk asset):")
print(f"Mean-variance risk aversion coefficient: {gamma}")
print(f"The fraction of total wealth X invested in the tangency (risky)␣
 ↪portfolio y* is: {y_star:.2%}, that is ${y_star * Total_invested:,.2f}usd")
print(f"The fraction invested in the risk-free asset (1-y*) is: {1-y:.2%}, that␣
 ↪is ${(1-y_star) * Total_invested:,.2f}usd")
# E[R_p] = Rf + y* (E[Rt] - Rf)
ER_p = risk_free + (y_star * (vol_ret_sr_optimal[1] - risk_free))
sigma_p = y_star * vol_ret_sr_optimal[0]
print(f"Expected Return E[Rp] = {ER_p:.2%}")
print(f"Volatility sigma_p = {sigma_p:.2%}")
print(f"Sharpe Ratio = {(ER_p - risk_free)/sigma_p:.2}")

# b) For long-only, no-borrowing constraint: set  =min(1,max(0, *)).
y = min(1, max(0, y_star))
print("\nb) For long-only, no-borrowing constraint:")
print(f"Mean-variance risk aversion coefficient: {gamma}")
print(f"The fraction of total wealth X invested in the tangency (risky)␣
 ↪portfolio 'y' is: {y:.2%}, that is ${y*Total_invested:,.2f}usd")
print(f"The fraction invested in the risk-free asset (1-y) is: {1-y:.2%}, that␣
 ↪is ${(1-y)*Total_invested:,.2f}usd")
ER_p = risk_free + (y * (vol_ret_sr_optimal[1] - risk_free))
sigma_p = y * vol_ret_sr_optimal[0]
print(f"Expected Return E[Rp] = {ER_p:.2%}")
print(f"Volatility sigma_p = {sigma_p:.2%}")
print(f"Sharpe Ratio = {(ER_p - risk_free)/sigma_p:.2}")
```

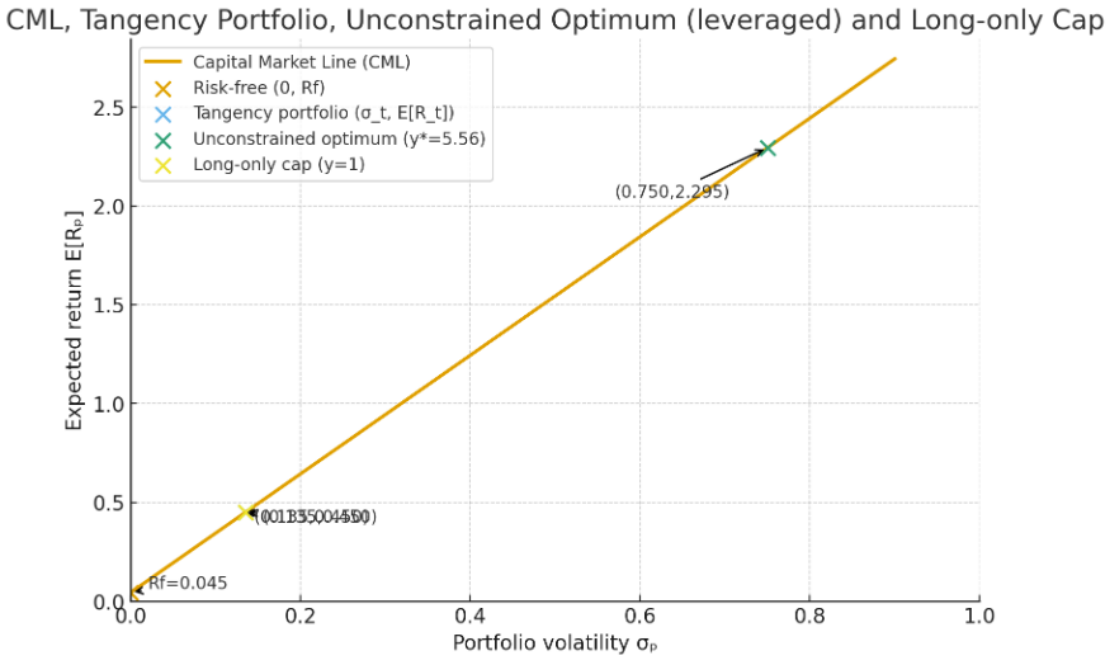
```
a) Allowing borrowing (short in risk asset):
```

```
Mean-variance risk aversion coefficient: 4
The fraction of total wealth X invested in the tangency (risky) portfolio y* is:
183.02%, that is $179,710.44usd
The fraction invested in the risk-free asset (1-y*) is: 10.12%, that is
$-81,519.78usd
Expected Return E[Rp] = 646.06%
Volatility sigma_p = 126.68%
Sharpe Ratio = 5.1

b) For long-only, no-borrowing constraint:
Mean-variance risk aversion coefficient: 4
The fraction of total wealth X invested in the tangency (risky) portfolio 'y'
is: 100.00%, that is $98,190.65usd
The fraction invested in the risk-free asset (1-y) is: 0.00%, that is $0.00usd
Expected Return E[Rp] = 354.88%
Volatility sigma_p = 69.22%
Sharpe Ratio = 5.1
```

The tangency portfolio has an enormously high risk-adjusted return — i.e., its Sharpe ratio([ ]−)/ =0.405/0.135=3.0(E[Rt]−Rf)/ t=0.405/0.135=3.0 is extremely high.

gamma = 4 That's a moderate risk aversion level. Even with moderate aversion, such a high Sharpe ratio pushes you toward aggressive leverage.

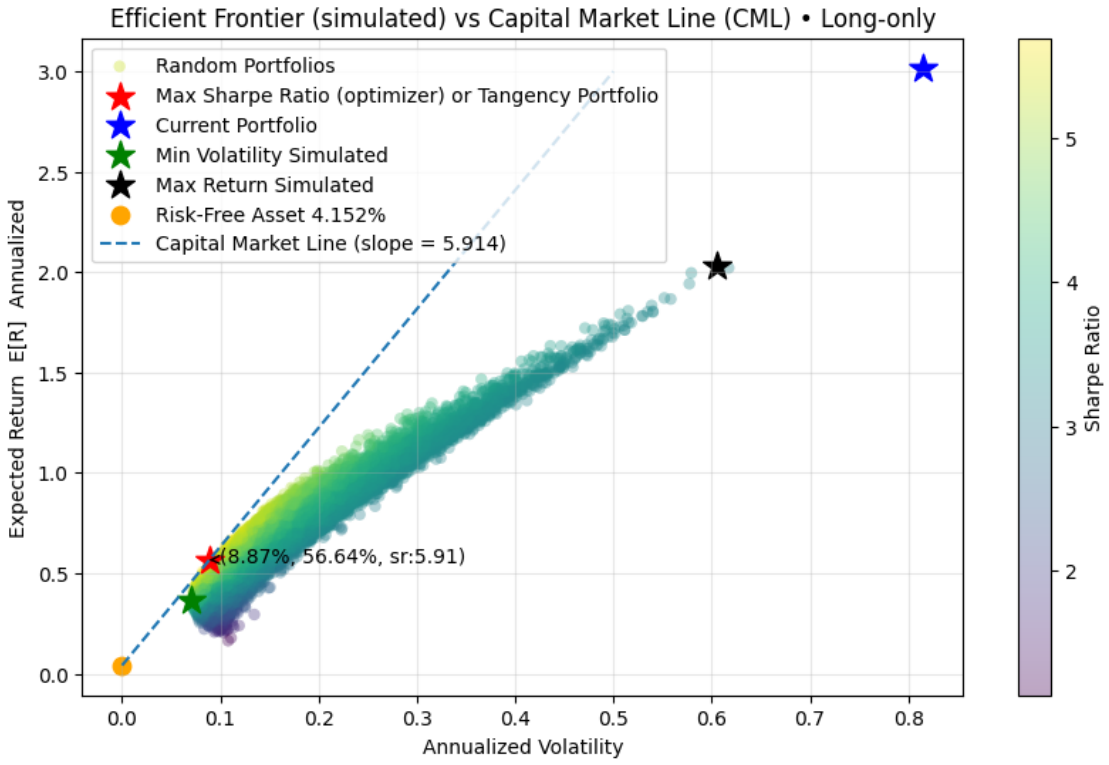
The tangency portfolio has an extremely high Sharpe (3.0). With moderate risk aversion =4, the utility-maximizing solution is to lever the tangency portfolio heavily (556% of wealth) because the reward-to-risk tradeoff is so favorable.



18.3 Indices & Portfolio

[85]:
```
weights_optimal, vol_ret_sr_optimal = functions.efficient_froentier_sharp_ratio(
                merge_daily_returns,
                [portfolio_annualized_volatility, portfolio_annualized_return],
                merge_mean_annualized,
                merge_covariance,
                risk_free, #Risk Free
                True, # Long only = True, Short allowed = False
                no_starts, # no. of starts (25 default)
                no_simul, # no. of simulations
                123 # seed
                )
```

```
Optimizing Sharpe…

Optimum Weights (%) - Tangency Portfolio
EE.UU. (S&P 500)           0.00
EE.UU. (NASDAQ)           50.45
EE.UU. (DJIA)              0.00
EE.UU. (Russell 100)       0.00
México (IPC)               0.97
Japón (Nikkei 225)        14.83
Alemania (DAX)            30.03
Reino Unido (FTSE 100)     0.00
Initial Portfolio          3.71

Optimum Sharpe Ratio: 5.914
Expected Annual Return: 56.64%
Annual Risk: 8.87%

Simulating random Portfolios…

Simulated Portfolio with Minimum Volatility:
Volatility: 7.01%
Return: 36.23%
Sharpe Ratio: 4.579

Simulated Portfolio with Maximum Return:
Volatility: 60.49%
Return: 202.63%
Sharpe Ratio: 3.281

Current Portfolio:
Volatility: 81.50%
Return: 301.54%
Sharpe Ratio: 3.649
```



18.4 **TO-DO: Include GUI and "make your own portfolio" with a variety of Stocks to choose.

(see Frontera Eficiente for GUI)

19 Indicators: alpha, Beta, R^2, SR, Sortino, VaR, CVaR - vs with Benchmark

Métrica	Valor Óptimo / Bueno	Interpretación
Alpha ( )	> 0 (ideal: +2% a +10% anual)	Exceso de retorno sobre lo esperado por el CAPM; positivo indica valor agregado.
Beta ( )	1 (mercado), <1 defensivo, >1 agresivo	Sensibilidad al mercado; >1 = más volátil, <1 = más estable.
R² (correlación)	> 0.8 (80% o más)	El portafolio se mueve casi igual que el benchmark (muy "indexado").
R² (correlación)	0.6 – 0.8	Buena relación con el mercado, pero con diferencias notables.
R² (correlación)	< 0.30	El portafolio se comporta muy distinto al mercado (alta independencia).

Métrica	Valor Óptimo / Bueno	Interpretación
μ (Retorno anual)	> 8% estable, > 15% agresivo	Rendimiento esperado anualizado del portafolio.
(Volatilidad anual)	10%–20% moderado, <10% defensivo, >25% muy riesgoso	Mide el riesgo total (desviación estándar).
Sharpe Ratio	> 1 bueno, > 1.5 muy bueno, > 2 excelente	Retorno ajustado por riesgo total.
Sortino Ratio	> 2 excelente	Retorno ajustado por riesgo a la baja (mejor si » Sharpe).
VaR (95%, 1d)	< 2%	Pérdida máxima esperada en un día con 95% de confianza.
CVaR (95%, 1d)	< 3%	Pérdida promedio en los peores días (cola izquierda de la distribución).

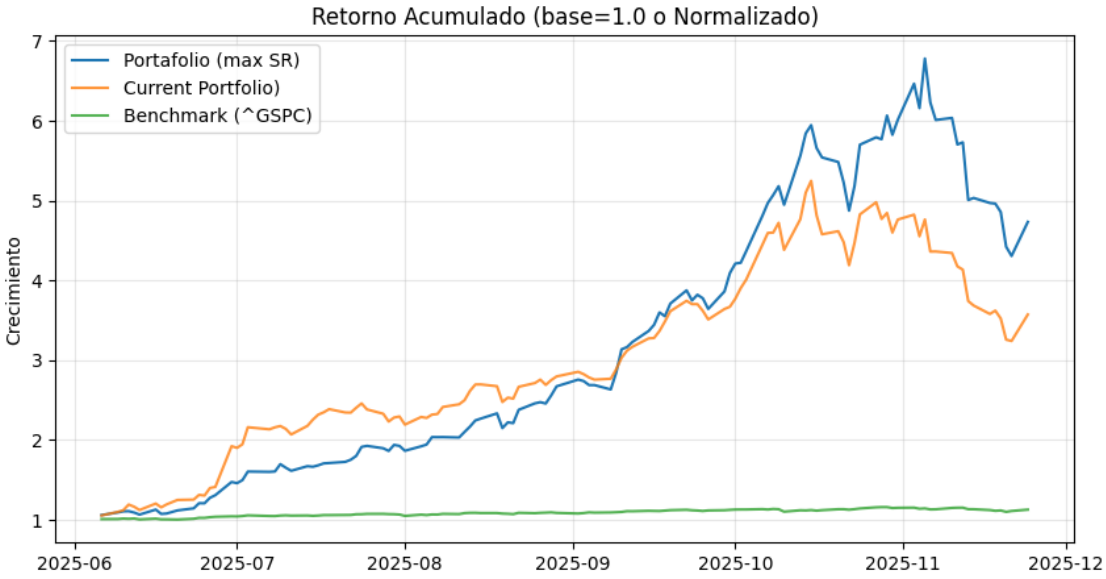
[86]:
```
#recall
display(benchmark_indices)
```

```
{'EE.UU. (S&P 500)': '^GSPC',
 'EE.UU. (NASDAQ)': '^IXIC',
 'EE.UU. (DJIA)': '^DJI',
 'EE.UU. (Russell 100)': '^RUI',
 'México (IPC)': '^MXX',
 'Japón (Nikkei 225)': '^N225',
 'Alemania (DAX)': '^GDAXI',
 'Reino Unido (FTSE 100)': '^FTSE'}
```

[87]:
```
# Choose a benchmark to compare for alpha, beta, R2...
index_benchmark = 0

functions.indicators(tickers, start_date, today, benchmark_indices,␣
 ↪index_benchmark,
                     risk_free, portfolio_weights=pd.
 ↪Series(weights_df['Weights']), no_starts=no_starts)
```

```
Descargando datos…

Weights:
        Optimized Portfolio (max SR) Weights  Current Portfolio Weights
Ticker
AMD                                      0.21                       0.11
ASTS                                     0.02                       0.11
BE                                       0.33                       0.07
BMNR                                     0.01                       0.05
HIMS                                     0.00                       0.11
IREN                                     0.43                       0.32
```

```
OUST                                    0.00                    0.24

Métricas (anualizadas)

                       anual    anual  Sharpe  Sortino
Current Portfolio     3.0154   0.8150  3.6488   8.6726
Portafolio (max SR)   3.5488   0.6922  5.0672  10.2549
Benchmark [^GSPC]     0.2631   0.1114  1.9886      NaN

Exceso risk-free usado: 4.15% anual | Rend: simple | VaR: 95% (historical)

Alpha/Beta/R²/VaR/CVaR con respecto a ^GSPC

                     Alpha (anual)    Beta      R²  VaR 1d  CVaR 1d
Current Portfolio           2.1598  3.6735  0.2523  0.0653   0.0805
Portafolio (max SR)         2.7373  3.4746  0.3129  0.0489   0.0831
```



19.1 ** TO-DO: EN Metricas Anualizadas incluir de forma automatica los otros 6 benchmarks

20 TO-DO: CAPM (en archivo de Indicadores)

21 Candles