EnriqueGarcia

January 6, 2026

1 Libraries

[142]:
```
#Instalar librerias
import importlib.util
import sys

def check_and_install_library(library_name_list: list):
    for library_name in library_name_list:
        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.")
    return

library_name_list = ['pandas', 'numpy', 'jupyter', 'notebook', 'yfinance',␣
 ↪'matplotlib.pyplot', 'json', 'ipynb', 'import_ipynb', 'datetime',
                     'ipywidgets', 'IPython.display', 'anywidget',# widgets
                     'nbconvert', 'pandoc', 'TeX'] #To export to HTML and PDF

check_and_install_library(library_name_list)



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

from plotly.graph_objects import FigureWidget

%reload_ext autoreload
%autoreload 2
```

```
Library 'TeX' not found. Installing…
Requirement already satisfied: TeX in
c:\users\egarcia\appdata\local\programs\python\python312\lib\site-packages (1.8)
Library 'TeX' installed successfully.
```

2 Dates

[2]:
```
#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},\n \
      End Date: {today},\n \
      number of days {no_days}")
```

```
       Start Date: 2021-04-25 00:00:00,
       End Date: 2026-01-06 00:00:00,
       number of days 1717 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.

[3]:
```
risk_free = 0.04152 #10-year treasury

# Open list of tickers from original Robinhood's file_df_df
file_name = "Robinhood.xlsx" # Enrique
# file_df = "App_GBM_Detalle_Portafolio_USA_1763936603841.xlsx"  # Beto
file_df = pd.read_excel(file_name)

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

no_assets = len(tickers)

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

```
Tickers: ['QYLD', 'NVDA', 'PBR', 'PLTR', 'MSFT', 'AAPL', 'NU', 'DIS', 'VOOV',
'TSM', 'GLD', 'QQQM', 'VOOG', 'KO', 'SOFI', 'QSR', 'UNH', 'TSLA', 'SPYD',
'SERV', 'SOXX', 'VOO', 'OMAB', 'VYM', 'VGT', 'GOOGL', 'CME', 'GOOG', 'META',
'BAC', 'CRWD', 'NFLX', 'MCD', 'SPYG', 'VUG', 'ADBE', 'MAR', 'VTV', 'CAT', 'LMT',
'ASML', 'BRK-B', 'SPG', 'PSA', 'HD']
Number of Assets: 45
```

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 10 and 20 (fair valuation)	info.trailingPE
P/B Ratio	Price per Share / Book Value	< 3 (not overvalued)	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")

Ratio	Formula	Criteria (great if)	Attribute
Gross Margin Ratio	Sales - COGS / Sales	> 0.40 (0.35-0.65)	info.grossMargins
Net Margin (%)	Net Income / Revenue	(15%-25%)	functions.get_net_margin("AAPL")
Current Ratio	Current Assets / Current Liabilities	(1.5-2.0)	info.currentRatio
Earning Growth Ratio (PEG Ratio)	P/E Ratio / Annual EPS Growth Rate (% as a whole number)	< 1.0 (undervalued)	info.earningsGrowth

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

[4]:
```python
## 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  \
Ticker
NFLX                                         Netflix, Inc.
GOOG                                         Alphabet Inc.
GOOGL                                        Alphabet Inc.
META                                   Meta Platforms, Inc.
DIS                                 The Walt Disney Company
TSLA                                           Tesla, Inc.
MAR                               Marriott International, Inc.
MCD                                   McDonald's Corporation
```

```
QSR                  Restaurant Brands International Inc.
HD                                   The Home Depot, Inc.
KO                                  The Coca-Cola Company
SOXX                            iShares Semiconductor ETF
VGT     Vanguard Information Technology Index Fund ETF…
VUG                 Vanguard Growth Index Fund ETF Shares
VOOG        Vanguard S&P 500 Growth Index Fund ETF Shares
QYLD                 Global X NASDAQ 100 Covered Call ETF
QQQM                               Invesco NASDAQ 100 ETF
SPYG       State Street SPDR Portfolio S&P 500 Growth ETF
VOO                                  Vanguard S&P 500 ETF
VOOV         Vanguard S&P 500 Value Index Fund ETF Shares
VTV                  Vanguard Value Index Fund ETF Shares
VYM     Vanguard High Dividend Yield Index Fund ETF Sh…
SPYD    State Street SPDR Portfolio S&P 500 High Divid…
GLD                                      SPDR Gold Shares
PBR                  Petróleo Brasileiro S.A. - Petrobras
SOFI                              SoFi Technologies, Inc.
NU                                       Nu Holdings Ltd.
CME                                        CME Group Inc.
BRK-B                             Berkshire Hathaway Inc.
BAC                           Bank of America Corporation
UNH                       UnitedHealth Group Incorporated
CAT                                      Caterpillar Inc.
LMT                           Lockheed Martin Corporation
OMAB    Grupo Aeroportuario del Centro Norte, S.A.B. d…
SERV                                  Serve Robotics Inc.
PSA                                        Public Storage
SPG                            Simon Property Group, Inc.
PLTR                           Palantir Technologies Inc.
NVDA                                   NVIDIA Corporation
ASML                                    ASML Holding N.V.
AAPL                                           Apple Inc.
MSFT                                Microsoft Corporation
TSM     Taiwan Semiconductor Manufacturing Company Lim…
ADBE                                           Adobe Inc.
CRWD                           CrowdStrike Holdings, Inc.

                        Sector                             Industry  CAGR_%  \
Ticker
NFLX    Communication Services                        Entertainment   13.21
GOOG    Communication Services       Internet Content & Information   23.97
GOOGL   Communication Services       Internet Content & Information   24.10
META    Communication Services       Internet Content & Information   18.12
DIS     Communication Services                        Entertainment   -9.24
TSLA         Consumer Cyclical                   Auto Manufacturers   13.78
MAR          Consumer Cyclical                              Lodging   17.93
MCD          Consumer Cyclical                          Restaurants    8.03
```

```
QSR         Consumer Cyclical                          Restaurants   3.70
HD          Consumer Cyclical             Home Improvement Retail   4.08
KO         Consumer Defensive           Beverages - Non-Alcoholic   8.37
SOXX              ETF, others                          ETF, others  18.82
VGT               ETF, others                          ETF, others  16.07
VUG               ETF, others                          ETF, others  13.41
VOOG              ETF, others                          ETF, others  13.71
QYLD              ETF, others                          ETF, others   7.28
QQQM              ETF, others                          ETF, others  14.21
SPYG              ETF, others                          ETF, others  13.77
VOO               ETF, others                          ETF, others  12.82
VOOV              ETF, others                          ETF, others  10.85
VTV               ETF, others                          ETF, others  10.73
VYM               ETF, others                          ETF, others  10.95
SPYD              ETF, others                          ETF, others   6.48
GLD               ETF, others                          ETF, others  20.98
PBR                    Energy              Oil & Gas Integrated  36.76
SOFI       Financial Services                   Credit Services  12.71
NU         Financial Services                  Banks - Regional  12.45
CME        Financial Services   Financial Data & Stock Exchanges  10.76
BRK-B      Financial Services           Insurance - Diversified  13.85
BAC        Financial Services               Banks - Diversified  10.81
UNH                Healthcare                  Healthcare Plans  -1.44
CAT               Industrials  Farm & Heavy Construction Machinery  25.49
LMT               Industrials               Aerospace & Defense  10.00
OMAB              Industrials           Airports & Air Services  24.74
SERV              Industrials      Specialty Industrial Machinery -12.53
PSA               Real Estate                 REIT - Industrial   3.31
SPG               Real Estate                     REIT - Retail  15.45
PLTR               Technology         Software - Infrastructure  52.23
NVDA               Technology                    Semiconductors  70.14
ASML               Technology  Semiconductor Equipment & Materials  14.83
AAPL               Technology              Consumer Electronics  16.30
MSFT               Technology         Software - Infrastructure  14.36
TSM                Technology                    Semiconductors  25.15
ADBE               Technology            Software - Application  -8.96
CRWD               Technology         Software - Infrastructure  16.29

         P/E Ratio  P/B Ratio  ROIC_%    D/E_%    EPS_usd  ROE Ratio  …  \
Ticker                                                                …
NFLX     38.108330  14.932244   28.58   65.822   3.242550    0.42861  …
GOOG     31.355732   9.906034   30.69   11.424  11.197380    0.35450  …
GOOGL    31.247780   9.881684   30.69   11.424  11.197380    0.35450  …
META     29.149998   8.557937   33.52   26.311  30.418530    0.32643  …
DIS      16.628279   1.859483    7.32   39.632   7.355800    0.12203  …
TSLA    311.496550  18.774212   10.61   17.082   2.203830    0.06791  …
MAR      32.774500 -26.863880   24.47    0.000  11.383310    0.00000  …
MCD      25.585323 -98.735590   20.02    0.000  13.229860    0.00000  …
```

```
QSR      23.666666   6.470816   10.81  306.718   4.015010    0.25245  …
HD       23.471350  28.257370   24.64  544.586  15.095210    1.62909  …
KO       22.496689   9.354261   20.50  144.771   3.220510    0.42442  …
SOXX     41.787422   0.749979    0.00    0.000   0.000000    0.00000  …
VGT      39.488186   0.000000    0.00    0.000   0.000000    0.00000  …
VUG      39.276190   2.263974    0.00    0.000   0.000000    0.00000  …
VOOG     35.788900   0.000000    0.00    0.000   0.000000    0.00000  …
QYLD     34.672573   0.000000    0.00    0.000   0.000000    0.00000  …
QQQM     34.027912   0.000000    0.00    0.000   0.000000    0.00000  …
SPYG     33.924810   1.701520    0.00    0.000   0.000000    0.00000  …
VOO      29.083656   1.618149    0.00    0.000   0.000000    0.00000  …
VOOV     23.608147   0.000000    0.00    0.000   0.000000    0.00000  …
VTV      21.379675   2.568654    0.00    0.000   0.000000    0.00000  …
VYM      20.940632   0.000000    0.00    0.000   0.000000    0.00000  …
SPYD     15.968643   0.000000    0.00    0.000   0.000000    0.00000  …
GLD       0.000000   2.404230    0.00    0.000   0.000000    0.00000  …
PBR       5.435185   1.940786    8.81   88.498   2.536670    0.19022  …
SOFI     52.285717   4.017012    0.00   31.999   0.572950    0.08593  …
NU       34.500004   8.236915    0.00    0.000   0.911740    0.27800  …
CME      26.678951   3.511150   13.50   13.372  11.683760    0.13346  …
BRK-B    15.947536   0.001027   12.90   18.166  24.190810    0.10170  …
BAC      15.543715   1.499038    0.00    0.000   4.357060    0.09871  …
UNH      17.822823   3.234996   12.62   75.734  17.767490    0.17476  …
CAT      31.643555  13.956912   21.86  201.046  22.386720    0.46283  …
LMT      28.483854  19.035872   25.81  358.987  29.223770    0.62776  …
OMAB     17.794788  75.197770   29.05  132.873   7.859000    0.54332  …
SERV      0.000000   3.024809 -355.90    1.461  -1.773330   -0.47177  …
PSA      27.092419   9.225600   12.74  106.758  10.247400    0.19933  …
SPG      26.653566  25.888590   13.78  884.863   6.882500    0.82455  …
PLTR    395.545440  62.943940    9.44    3.520   1.010140    0.19504  …
NVDA     46.564354  38.454617   90.19    9.102   7.566360    1.07359  …
ASML     43.291855  21.398302   79.56   14.240  30.810055    0.53852  …
AAPL     35.825737  53.548386   82.29  152.411   9.155080    1.71422  …
MSFT     33.678776   9.681614   27.78   33.154  18.742380    0.32241  …
TSM      33.324715  52.480240   36.86   20.436  13.083090    0.34657  …
ADBE     19.842012  11.895383   45.27   57.197  26.340080    0.55426  …
CRWD      0.000000  28.652567  -25.06   20.154   4.834480   -0.08815  …

        Gross Margin_ratio  Net Margin_%  Current Ratio  Overall Risk   Beta  \
Ticker
NFLX               0.48085         22.34          1.332           9.0  1.711
GOOG               0.59172         28.60          1.747           0.0  1.086
GOOGL              0.59172         28.60          1.747          10.0  1.086
META               0.82013         37.91          1.978          10.0  1.287
DIS                0.37764         13.14          0.710           3.0  1.442
TSLA               0.17007          7.30          2.066          10.0  1.835
MAR                0.81554          9.46          0.467           7.0  1.157
MCD                0.57425         31.72          1.000           5.0  0.531
```

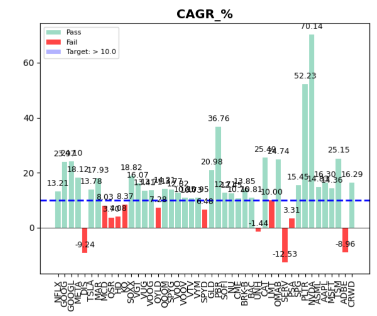
```
QSR                0.33528         12.15          1.059           4.0  0.605
HD                 0.33355          9.28          1.051           1.0  1.072
KO                 0.61633         22.59          1.211           3.0  0.387
SOXX               0.00000          0.00          0.000           0.0  0.000
VGT                0.00000          0.00          0.000           0.0  0.000
VUG                0.00000          0.00          0.000           0.0  0.000
VOOG               0.00000          0.00          0.000           0.0  0.000
QYLD               0.00000          0.00          0.000           0.0  0.000
QQQM               0.00000          0.00          0.000           0.0  0.000
SPYG               0.00000          0.00          0.000           0.0  0.000
VOO                0.00000          0.00          0.000           0.0  0.000
VOOV               0.00000          0.00          0.000           0.0  0.000
VTV                0.00000          0.00          0.000           0.0  0.000
VYM                0.00000          0.00          0.000           0.0  0.000
SPYD               0.00000          0.00          0.000           0.0  0.000
GLD                0.00000          0.00          0.000           0.0  0.000
PBR                0.48152          8.23          0.819           0.0 -0.032
SOFI               0.82510         19.09          1.150           9.0  1.932
NU                 0.00000         23.85          0.000           0.0  1.083
CME                1.00000         57.52          1.021          10.0  0.291
BRK-B              0.24361         20.98          2.722          10.0  0.710
BAC                0.00000         26.63          0.000           4.0  1.295
UNH                0.19701          3.60          0.823           8.0  0.425
CAT                0.30120         16.65          1.384           4.0  1.568
LMT                0.08252          7.51          1.129           5.0  0.245
OMAB               0.74202         32.70          1.138           0.0  0.611
SERV               0.00000      -2162.29         17.214          10.0  0.000
PSA                0.72701         44.13          0.269           5.0  0.991
SPG                0.81993         39.75          0.592          10.0  1.400
PLTR               0.80808         16.13          6.427          10.0  1.545
NVDA               0.70050         55.85          4.468           8.0  2.314
ASML               0.52711         26.79          1.308           0.0  1.341
AAPL               0.46905         26.92          0.893           1.0  1.093
MSFT               0.68764         36.15          1.401           5.0  1.073
TSM                0.58976         40.02          2.693           0.0  1.274
ADBE               0.89268         25.85          0.996           1.0  1.526
CRWD               0.74277         -0.49          1.811          10.0  1.029

          EBITDA_usd  EBITDA Margins Ratio  Earning Growth Ratio  \
Ticker
NFLX    1.296965e+10               0.29899                 0.087
GOOG    1.451740e+11               0.37661                 0.353
GOOGL   1.451740e+11               0.37661                 0.353
META    9.839900e+10               0.51937                -0.826
DIS     1.941900e+10               0.20566                 1.873
TSLA    1.076800e+10               0.11260                -0.371
MAR     4.600000e+09               0.66919                 0.290
MCD     1.429200e+10               0.54417                 0.016
```

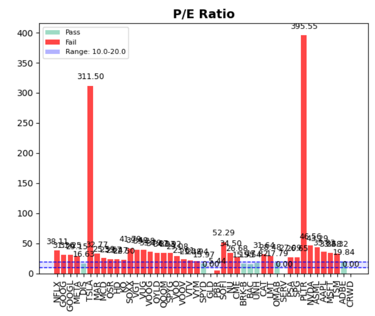
```
QSR     2.701000e+09            0.29156               0.217
HD      2.558700e+10            0.15396              -0.014
KO      1.630700e+10            0.34213               0.301
SOXX    0.000000e+00            0.00000               0.000
VGT     0.000000e+00            0.00000               0.000
VUG     0.000000e+00            0.00000               0.000
VOOG    0.000000e+00            0.00000               0.000
QYLD    0.000000e+00            0.00000               0.000
QQQM    0.000000e+00            0.00000               0.000
SPYG    0.000000e+00            0.00000               0.000
VOO     0.000000e+00            0.00000               0.000
VOOV    0.000000e+00            0.00000               0.000
VTV     0.000000e+00            0.00000               0.000
VYM     0.000000e+00            0.00000               0.000
SPYD    0.000000e+00            0.00000               0.000
GLD     0.000000e+00            0.00000               0.000
PBR     1.910370e+11            0.38872               0.005
SOFI    0.000000e+00            0.00000               1.059
NU      0.000000e+00            0.00000               0.409
CME     4.494100e+09            0.70383              -0.004
BRK-B   1.038640e+11            0.27911               0.172
BAC     0.000000e+00            0.00000               0.315
UNH     2.924200e+10            0.06720              -0.602
CAT     1.395800e+10            0.21583              -0.036
LMT     7.257000e+09            0.09894               0.022
OMAB    9.810583e+09            0.61446               0.092
SERV   -8.331123e+07            0.00000               0.000
PSA     3.379439e+09            0.70476               0.213
SPG     4.549947e+09            0.73919               0.274
PLTR    8.757970e+08            0.22478               2.000
NVDA    1.126960e+11            0.60220               0.667
ASML    1.215790e+10            0.37743               0.038
AAPL    1.447480e+11            0.34782               0.912
MSFT    1.664370e+11            0.56647               0.127
TSM     2.484714e+12            0.68423               0.391
ADBE    9.551333e+09            0.40184               0.172
CRWD   -9.382400e+07           -0.02055               0.000

        Revenue Growth Ratio  Operating Margins Ratio
Ticker
NFLX                   0.172                  0.28220
GOOG                   0.159                  0.30512
GOOGL                  0.159                  0.30512
META                   0.262                  0.40075
DIS                   -0.005                  0.11868
TSLA                   0.116                  0.06628
MAR                    0.056                  0.65934
MCD                    0.030                  0.46906
```

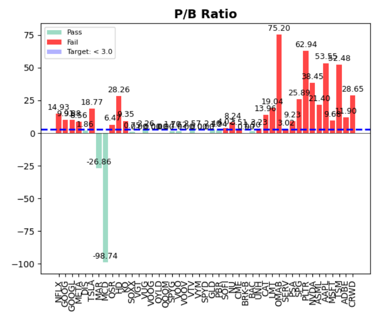
```
QSR                  0.069             0.27726
HD                   0.028             0.12945
KO                   0.051             0.32373
SOXX                 0.000             0.00000
VGT                  0.000             0.00000
VUG                  0.000             0.00000
VOOG                 0.000             0.00000
QYLD                 0.000             0.00000
QQQM                 0.000             0.00000
SPYG                 0.000             0.00000
VOO                  0.000             0.00000
VOOV                 0.000             0.00000
VTV                  0.000             0.00000
VYM                  0.000             0.00000
SPYD                 0.000             0.00000
GLD                  0.000             0.00000
PBR                 -0.013             0.36224
SOFI                 0.378             0.15598
NU                   0.363             0.58215
CME                 -0.030             0.63386
BRK-B                0.021             0.41103
BAC                  0.126             0.35293
UNH                  0.122             0.03813
CAT                  0.095             0.17746
LMT                  0.088             0.11693
OMAB                 0.061             0.61165
SERV                 2.095           -50.67831
PSA                  0.031             0.46947
SPG                  0.082             0.50757
PLTR                 0.628             0.33296
NVDA                 0.625             0.63169
ASML                 0.007             0.32842
AAPL                 0.079             0.31647
MSFT                 0.184             0.48873
TSM                  0.303             0.50578
ADBE                 0.105             0.36503
CRWD                 0.222            -0.05589

[45 rows x 21 columns]
```

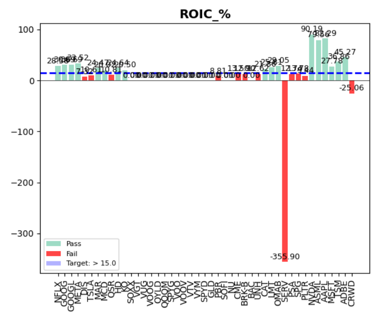
[5]:
```
ratio_criteria= {
    'CAGR_%': (">", 10.0), # %
    'P/E Ratio': ("between", (10.0, 20.0)), # ratio
    'P/B Ratio': ("<", 3.0), # ratio
    'ROIC_%': (">", 15.0), # %
    'D/E_%': ("<", 100.0), # %
    'EPS_usd': (">", 10.0), # USD. # 10% of asset's CAGR. Will automatically␣
 ↪trigger 10% CAGR logic in the function
    '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': ("between", (0.0, 1.0)),
    'EBITDA_usd': (">", 0.00), # USD
    'EBITDA Margins Ratio': (">", 0.15), # ratio
    'Earning Growth Ratio': ("<", 1.0), # ratio -  PEG < 1.0 (Undervalued):␣
 ↪This is the ideal range, as it suggests the stock price is low relative to␣
 ↪its earnings growth potential, indicating a potentially attractive␣
 ↪investment opportunity
    'Revenue Growth Ratio': (">", 0.15), # ratio
    'Operating Margins Ratio': (">", 0.15) # ratio
}
```

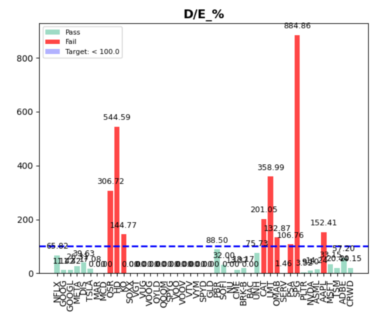
[6]:
```
# Plot financial Ratios
functions.plot_ratios(fa_df, ratio_criteria, plots_per_row = 3)
```

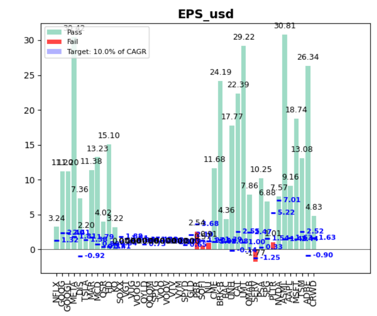

CAGR_%
Pass
Fail
Target: > 10.0

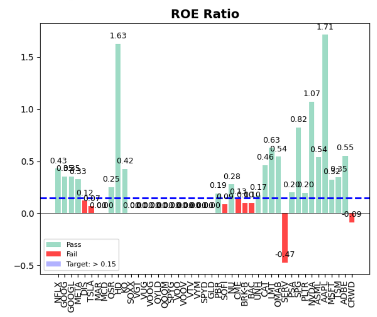

P/E Ratio
Pass
Fail
Range: 10.0-20.0

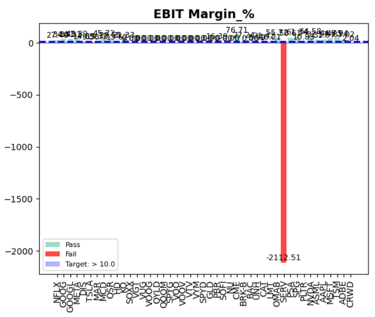

P/B Ratio
Pass
Fail
Target: < 3.0

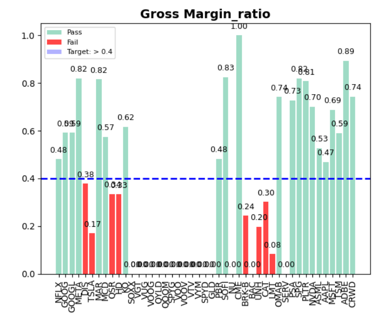

ROIC_%
Pass
Fail
Target: > 15.0

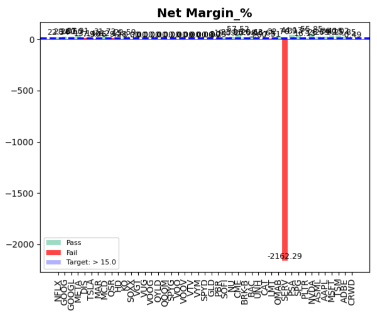

D/E_%
Pass
Fail
Target: < 100.0

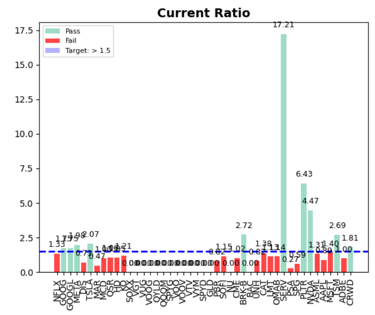

EPS_usd
Pass
Fail
Target: 10.0% of CAGR

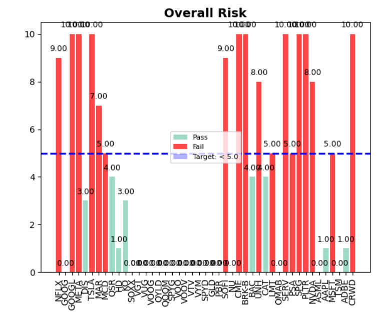

ROE Ratio
Pass
Fail
Target: > 0.15

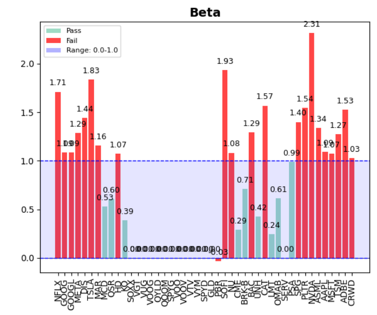

EBIT Margin_%
Pass
Fail
Target: > 10.0

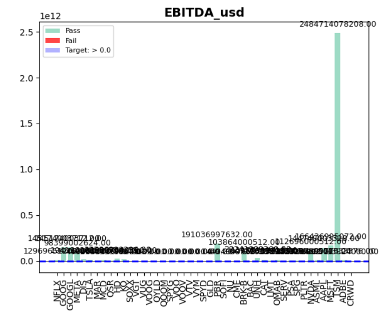

Gross Margin_ratio
Pass
Fail
Target: > 0.4

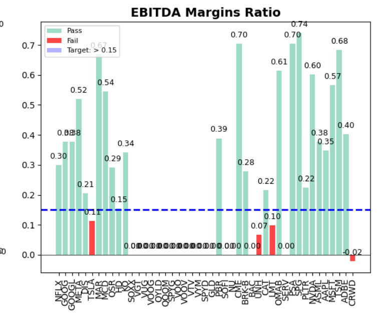

Net Margin_%
Pass
Fail
Target: > 15.0

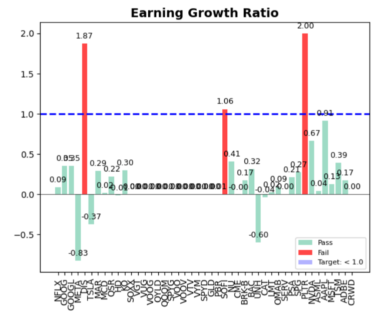

Current Ratio
Pass
Fail
Target: > 1.5

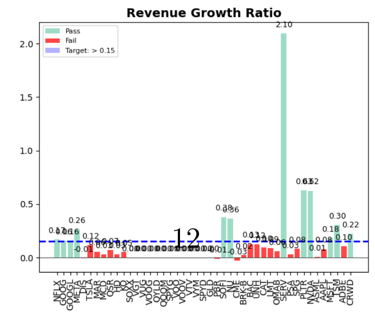

Overall Risk
Pass
Fail
Target: < 5.0

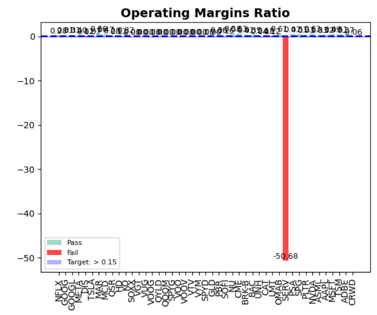

Beta
Pass
Fail
Range: 0.0-1.0


EBITDA_usd
1e12
Pass
Fail
Target: > 0.0


EBITDA Margins Ratio
Pass
Fail
Target: > 0.15


Earning Growth Ratio
Pass
Fail
Target: < 1.0

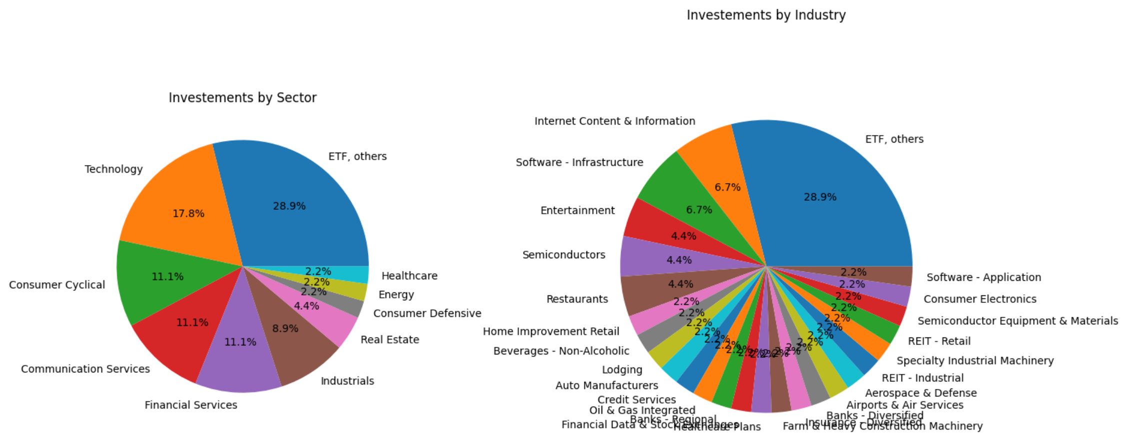

Revenue Growth Ratio
Pass
Fail
Target: > 0.15


Operating Margins Ratio
Pass
Fail
Target: > 0.15

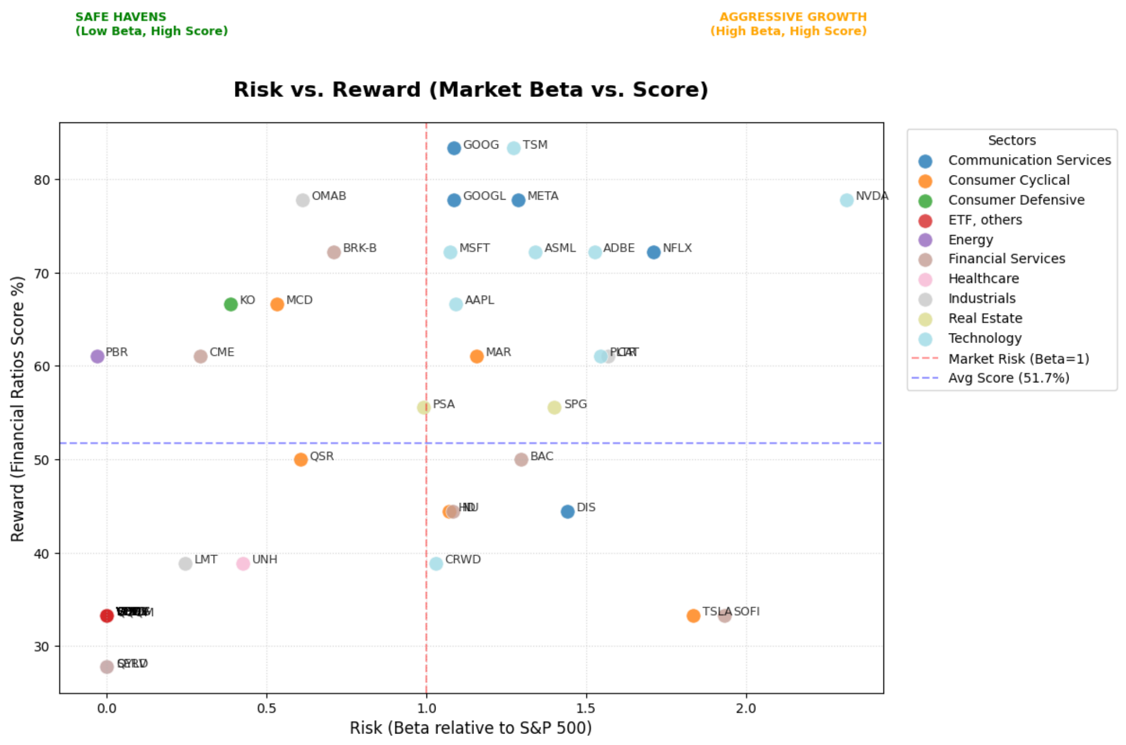
[7]:
```python
# Generate the data
scorecard = functions.generate_scorecard(fa_df, ratio_criteria)

# Display with a gradient highlight on the Score column
print("\n--- STOCK SELECTION SCORECARD ---")
display(scorecard.style.background_gradient(subset=['Score %'], cmap='RdYlGn'))
```

```
--- STOCK SELECTION SCORECARD ---

<pandas.io.formats.style.Styler at 0x27cb6852810>
```

[8]:
```python
# Sectors and Industries
functions.print_sector_industry(fa_df)
```



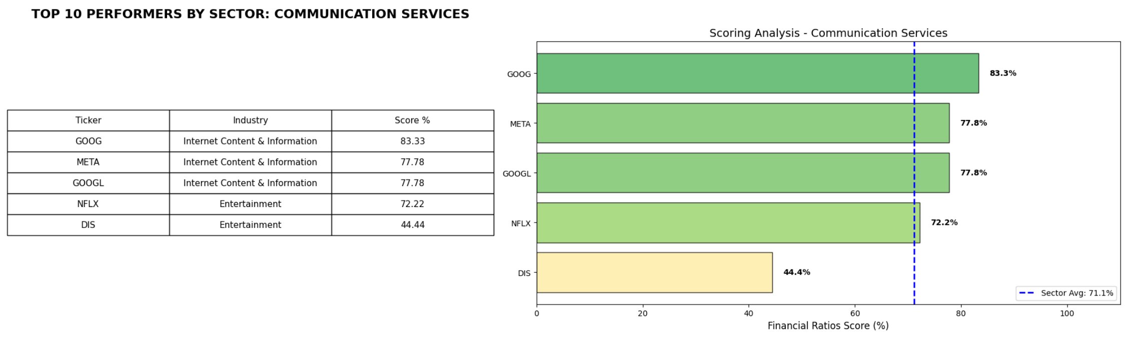
[9]:
```python
# --- SCORE vs BETA by Sector ---
functions.plot_risk_reward(fa_df, ratio_criteria)
```



```
[10]: # --- TOP N assets in all sectors ---
      top_N_data = functions.get_top_N_assets(fa_df, ratio_criteria, top_n=10)
      functions.plot_sector_treemap(top_N_data)

      # Grouping the top 5 to see the distribution
      summary = top_N_data.groupby(['Sector', 'Industry', 'Name', 'Ticker'])[['Score␣
       ↪%']].max().sort_values('Score %', ascending=False)
      display(summary.style.background_gradient(cmap='Greens'))
```

```
<pandas.io.formats.style.Styler at 0x27cbca01910>
```

```
[11]: # TOP N BY SECTOR AND INDUSTRY

      functions.top_N_sector_industry(fa_df, ratio_criteria, n_per_sector=10)
```

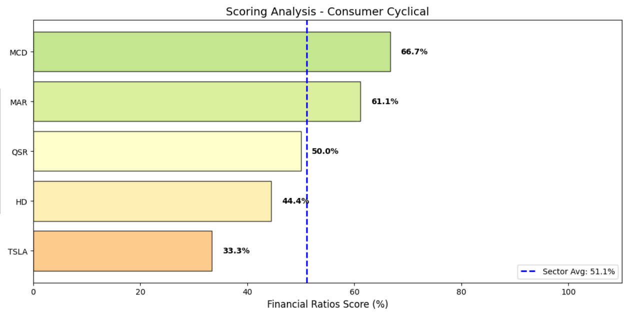


TOP 10 PERFORMERS BY SECTOR: COMMUNICATION SERVICES

Ticker	Industry	Score %
GOOG	Internet Content & Information	83.33
META	Internet Content & Information	77.78
GOOGL	Internet Content & Information	77.78
NFLX	Entertainment	72.22
DIS	Entertainment	44.44

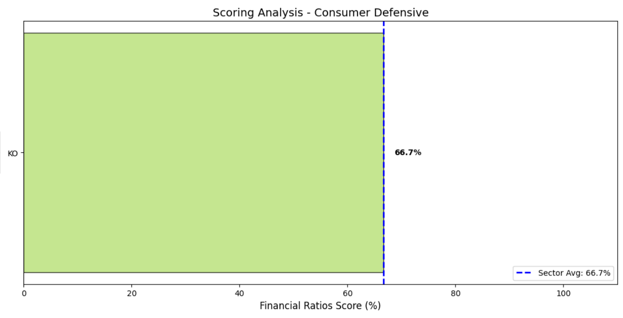
TOP 10 PERFORMERS BY SECTOR: CONSUMER CYCLICAL

Ticker	Industry	Score %
MCD	Restaurants	66.67
MAR	Lodging	61.11
QSR	Restaurants	50.0
HD	Home Improvement Retail	44.44
TSLA	Auto Manufacturers	33.33



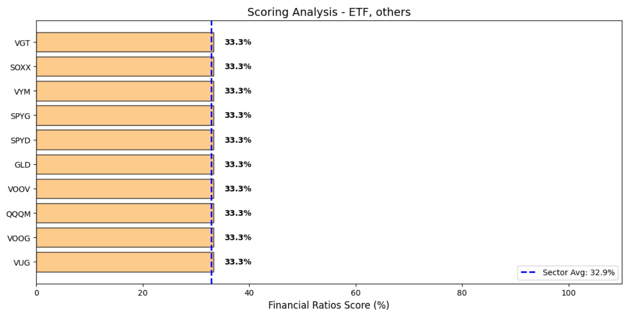
TOP 10 PERFORMERS BY SECTOR: CONSUMER DEFENSIVE

Ticker	Industry	Score %
KO	Beverages - Non-Alcoholic	66.67



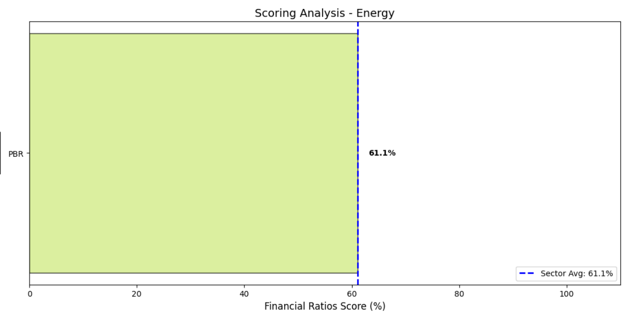
TOP 10 PERFORMERS BY SECTOR: ETF, OTHERS

Ticker	Industry	Score %
VGT	ETF, others	33.33
SOXX	ETF, others	33.33
VYM	ETF, others	33.33
SPYG	ETF, others	33.33
SPYD	ETF, others	33.33
GLD	ETF, others	33.33
VOOV	ETF, others	33.33
QQQM	ETF, others	33.33
VOOG	ETF, others	33.33
VUG	ETF, others	33.33



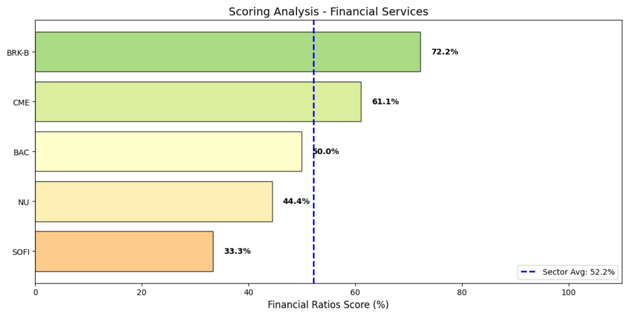
TOP 10 PERFORMERS BY SECTOR: ENERGY

Ticker	Industry	Score %
PBR	Oil & Gas Integrated	61.11



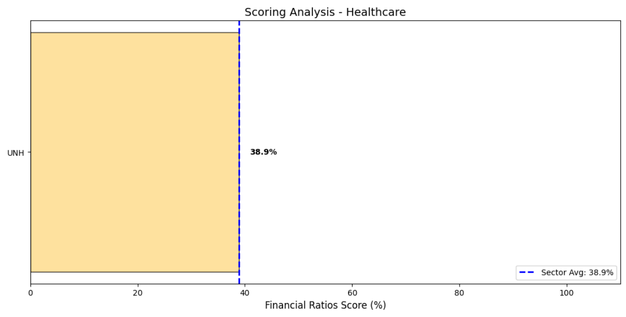
TOP 10 PERFORMERS BY SECTOR: FINANCIAL SERVICES

Ticker	Industry	Score %
BRK-B	Insurance - Diversified	72.22
CME	Financial Data & Stock Exchanges	61.11
BAC	Banks - Diversified	50.0
NU	Banks - Regional	44.44
SOFI	Credit Services	33.33



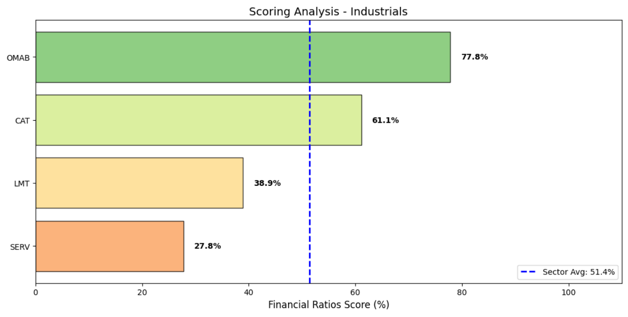
TOP 10 PERFORMERS BY SECTOR: HEALTHCARE

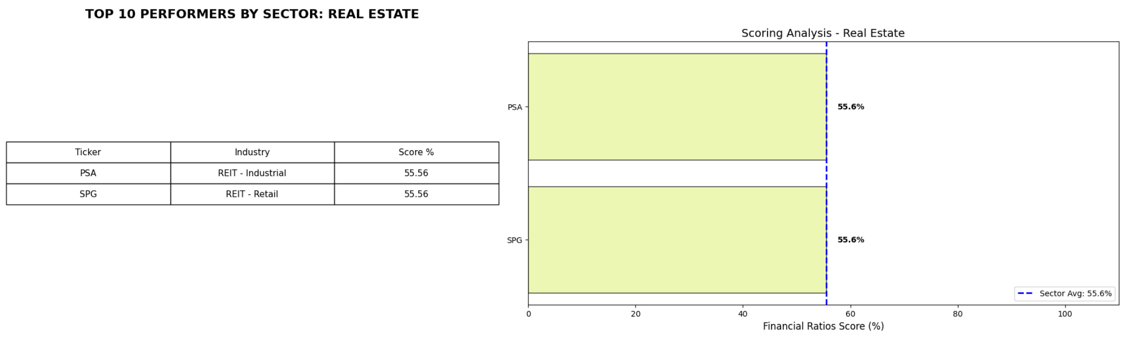
Ticker	Industry	Score %
UNH	Healthcare Plans	38.89



TOP 10 PERFORMERS BY SECTOR: INDUSTRIALS

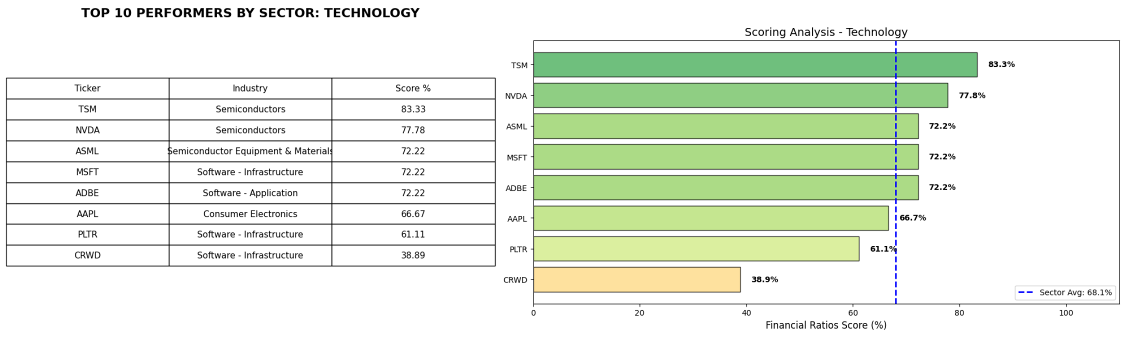
Ticker	Industry	Score %
OMAB	Airports & Air Services	77.78
CAT	Farm & Heavy Construction Machinery	61.11
LMT	Aerospace & Defense	38.89
SERV	Specialty Industrial Machinery	27.78





TOP 10 PERFORMERS BY SECTOR: REAL ESTATE

Ticker	Industry	Score %
PSA	REIT - Industrial	55.56
SPG	REIT - Retail	55.56



TOP 10 PERFORMERS BY SECTOR: TECHNOLOGY

Ticker	Industry	Score %
TSM	Semiconductors	83.33
NVDA	Semiconductors	77.78
ASML	Semiconductor Equipment & Materials	72.22
MSFT	Software - Infrastructure	72.22
ADBE	Software - Application	72.22
AAPL	Consumer Electronics	66.67
PLTR	Software - Infrastructure	61.11
CRWD	Software - Infrastructure	38.89

```
[12]: # TOP 1 BY SECTOR

final_picks = functions.top_1_sector(fa_df, ratio_criteria)
print("\n--- FINAL BEST-IN-CLASS PORTFOLIO SELECTION ---")
display(final_picks.style.hide(axis="index").background_gradient(subset=['Score␣
  ↪%'], cmap='RdYlGn'))
```

```
--- FINAL BEST-IN-CLASS PORTFOLIO SELECTION ---

<pandas.io.formats.style.Styler at 0x27cc18d0c80>
```

```
[13]: #  Portfolio Summary - Beta and Scores

# Option A: All Assets
stats_all = functions.get_portfolio_summary(fa_df, ratio_criteria, mode="all")
display(stats_all.style.hide(axis="index"))

# Option B: Top 10 Overall
stats_top10 = functions.get_portfolio_summary(fa_df, ratio_criteria,␣
  ↪mode="top_n", top_n=10)
display(stats_top10.style.hide(axis="index"))
```

```
# Option C: Top 1 per Sector
stats_sector = functions.get_portfolio_summary(fa_df, ratio_criteria,␣
 ↪mode="sector_best")
display(stats_sector.style.hide(axis="index"))
```

```
--- PORTFOLIO SUMMARY: ALL TICKERS ---
Tickers Included (45): GOOG, TSM, META, OMAB, GOOGL, NVDA, BRK-B, ASML, MSFT,
NFLX, ADBE, KO, MCD, AAPL, CAT, CME, PBR, MAR, PLTR, PSA, SPG, BAC, QSR, DIS,
HD, NU, UNH, CRWD, LMT, VGT, SOXX, TSLA, VYM, SPYG, SOFI, SPYD, GLD, VOOV, QQQM,
VOOG, VUG, VTV, VOO, QYLD, SERV

<pandas.io.formats.style.Styler at 0x27cc1b92de0>


--- PORTFOLIO SUMMARY: TOP 10 OVERALL ---
Tickers Included (10): GOOG, TSM, META, OMAB, GOOGL, NVDA, BRK-B, ASML, MSFT,
NFLX

<pandas.io.formats.style.Styler at 0x27cc1b67e60>


--- PORTFOLIO SUMMARY: TOP 1 PER SECTOR ---
Tickers Included (10): GOOG, TSM, OMAB, BRK-B, KO, MCD, PBR, PSA, UNH, VGT

<pandas.io.formats.style.Styler at 0x27cc1819790>
```

5 Fundamental Analysis ONE Ticker

TS302_Stock Full Analysis.ipynb

This section is to see in more detail any particular ticker

[14]:
```
# 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=('QYLD', 'NVDA', 'PBR', 'PLTR',␣
↪'MSFT', 'AAPL', 'NU', 'DIS', 'V…

Output()
```

[15]:
```
# Choose any ticket from the list above
ticker = yf.Ticker(ticker_widget)
ticker_name = ticker.info.get('symbol')
print(ticker_name)
```

```
QYLD
```

History

[16]:
```
# Example of history() of any ONE ticker for "1y"
hist = ticker.history(period="1y", auto_adjust=True)
print(ticker_name)
display(hist)
```

```
QYLD

                                Open       High        Low      Close  \
Date
2025-01-06 00:00:00-05:00  16.399330  16.434827  16.372709  16.399330
2025-01-07 00:00:00-05:00  16.443700  16.443700  16.248469  16.283966
2025-01-08 00:00:00-05:00  16.297274  16.328334  16.195223  16.310585
2025-01-10 00:00:00-05:00  16.275095  16.283969  16.062117  16.159731
2025-01-13 00:00:00-05:00  16.017742  16.133106  15.955624  16.133106
…                                …          …          …          …
2025-12-29 00:00:00-05:00  17.719999  17.750000  17.709999  17.740000
2025-12-30 00:00:00-05:00  17.730000  17.760000  17.725000  17.730000
2025-12-31 00:00:00-05:00  17.730000  17.740000  17.670000  17.670000
2026-01-02 00:00:00-05:00  17.750000  17.780001  17.620001  17.680000
2026-01-05 00:00:00-05:00  17.725000  17.789000  17.719999  17.760000
```

```
                              Volume  Dividends  Stock Splits  Capital Gains
Date
2025-01-06 00:00:00-05:00   4535000        0.0           0.0            0.0
2025-01-07 00:00:00-05:00   8486600        0.0           0.0            0.0
2025-01-08 00:00:00-05:00  11652700        0.0           0.0            0.0
2025-01-10 00:00:00-05:00  15684100        0.0           0.0            0.0
2025-01-13 00:00:00-05:00   8383300        0.0           0.0            0.0
…                                …        …             …              …
2025-12-29 00:00:00-05:00   4555200        0.0           0.0            0.0
2025-12-30 00:00:00-05:00   4812700        0.0           0.0            0.0
2025-12-31 00:00:00-05:00   4961300        0.0           0.0            0.0
2026-01-02 00:00:00-05:00  10828500        0.0           0.0            0.0
2026-01-05 00:00:00-05:00   5833600        0.0           0.0            0.0

[250 rows x 8 columns]
```

- Info
- Income Statement
- Balance Sheet

[17]:
```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
```

[18]:
```python
# Print info by category (some selected data only)

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: QYLD
longName: Global X NASDAQ 100 Covered Call ETF
sector: N/A
industry: N/A
country: N/A

2. Market info:
currentPrice: N/A
marketCap: N/A
volume: 5818312
52WeekChange: N/A
fiftyTwoWeekHigh: 18.89
fiftyTwoWeekLow: 14.475

3. Financial info:
priceToBook: N/A
forwardPE: N/A
trailingPE: 34.672573
profitMargins: N/A
```

```
totalRevenue: N/A
debtToEquity: N/A
epsForward: N/A
ebitda: N/A
floatShares: N/A
forwardEps: N/A
grossMargins: N/A
grossProfits: N/A
operatingCashflow: N/A
operatingMargins: N/A
returnOnAssets: N/A
returnOnEquity: N/A
revenueGrowth: N/A
revenuePerShare: N/A
impliedSharesOutstanding: N/A
totalCash: N/A
totalDebt: N/A

4. Dividends info:
dividendYield: 10.46
payoutRatio: N/A
dividendRate: N/A

5. Shares management info:
heldPercentInsiders: N/A
heldPercentInstitutions: N/A

6. Technical info:
sharesOutstanding: N/A
beta: N/A
currency: USD
```

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

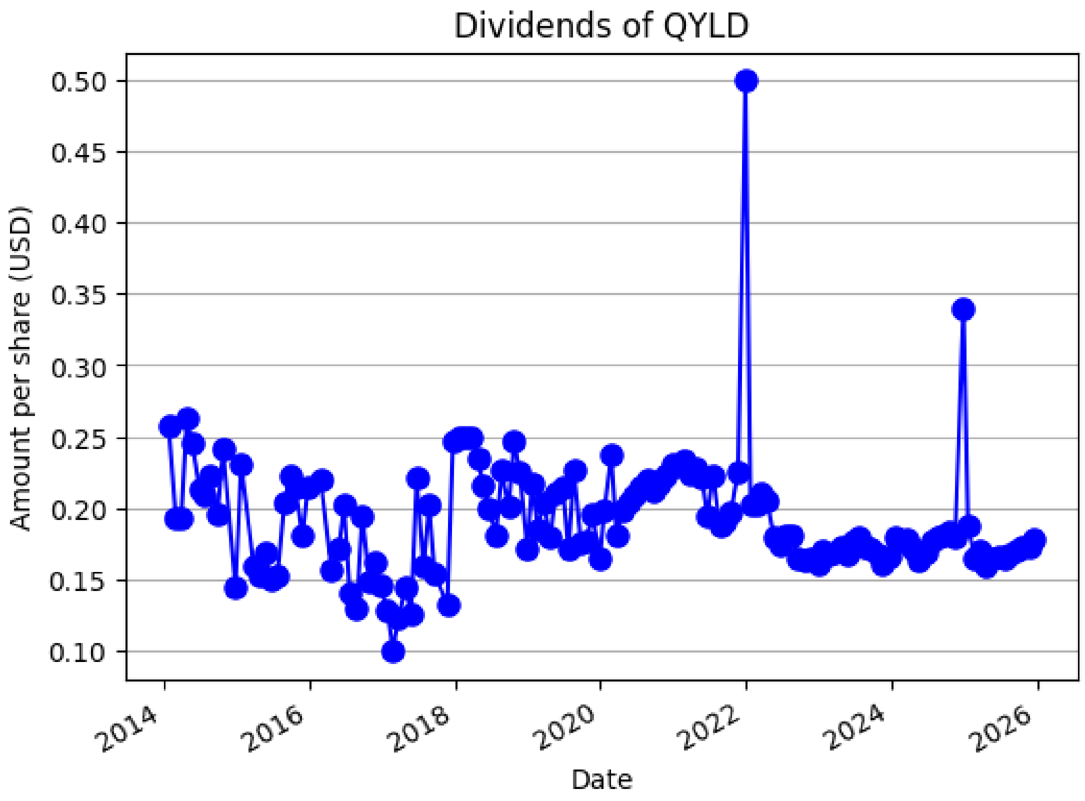
Dividends, Splits and Recommendations

[19]:
```python
# 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:
```

```
HTTP Error 404:
```

```
Dividends:
```



```
Dividends:

Date
2014-01-22 00:00:00-05:00    0.257
2014-02-26 00:00:00-05:00    0.193
2014-03-26 00:00:00-04:00    0.193
2014-04-23 00:00:00-04:00    0.263
2014-05-21 00:00:00-04:00    0.245
                               …
2025-08-18 00:00:00-04:00    0.168
2025-09-22 00:00:00-04:00    0.170
2025-10-20 00:00:00-04:00    0.173
2025-11-24 00:00:00-05:00    0.173
2025-12-22 00:00:00-05:00    0.178
Name: Dividends, Length: 139, dtype: float64
```

```
There are not Splits in this period for 'QYLD'
There are not Recommendations in this period for 'QYLD'
```

[20]:
```
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 QYLD---

Rule: Name
Metric: Name, Value found: Global X NASDAQ 100 Covered Call ETF
Status:   INFO (No comparison needed)

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

Rule: Industry
Metric: Industry, Value found: ETF, others
Status:   INFO (No comparison needed)
```

```
Rule: CAGR_%
Metric: CAGR_%, Value found: 7.28
Status:  FAIL (7.28 fails rule to be ge 10.0)

Rule: P/E Ratio
Metric: P/E Ratio, Value found: 34.672573
Status:  FAIL (34.672573 fails rule to be lt 25.0)

Rule: P/B Ratio
Metric: P/B Ratio, Value found: N/A
Status:  SKIP (Data for comparison not available)
Error (get_roic): Financial data for 'QYLD' not available or incomplete.

Rule: ROIC_%
Metric: ROIC_%, Value found: N/A
Status:  SKIP (Data for comparison not available)

Rule: D/E_%
Metric: D/E_%, Value found: N/A
Status:  SKIP (Data for comparison not available)

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

Rule: ROE Ratio
Metric: ROE Ratio, Value found: N/A
Status:  SKIP (Data for comparison not available)
(get_ebit_margin) No annual income statement found for 'QYLD'.

Rule: EBIT Margin_%
Metric: EBIT Margin_%, Value found: None
Status:  SKIP (Data for comparison not available)

Rule: Gross Margin_ratio
Metric: Gross Margin_ratio, Value found: N/A
Status:  SKIP (Data for comparison not available)
Error (get_net_margin): Financial data for 'QYLD' not available.

Rule: Net Margin_%
Metric: Net Margin_%, Value found: N/A
Status:  SKIP (Data for comparison not available)

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

```
Rule: Overall Risk
Metric: Overall Risk, Value found: N/A
Status:   SKIP (Data for comparison not available)

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

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

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

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: N/A
Status:   SKIP (Data for comparison not available)

Rule: Operating Margins Ratio
Metric: Operating Margins Ratio, Value found: N/A
Status:   SKIP (Data for comparison not available)
```

6 All Tickers in Portfolio

6.1 Import Prices from yfinance (All Tickers)

TS302_Stock Full Analysis.ipynb

[21]:
```
# 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()
```

```
[***                   7%                       ]  3 of 45 completed

Number of days since Brokerage account was opened: 1717 days, 0:00:00

[*********************100%***********************]  45 of 45 completed

<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 1180 entries, 2021-04-26 to 2026-01-05
Columns: 225 entries, ('Close', 'AAPL') to ('Volume', 'VYM')
dtypes: float64(182), int64(43)
memory usage: 2.0 MB
```

6.2 Prices (Close)

[22]:
```
prices_raw = df["Close"]
print('-- Last 5 days of prices --')
display(prices_raw.tail(5))
```

```
-- Last 5 days of prices --

Ticker            AAPL        ADBE         ASML        BAC       BRK-B  \
Date
2025-12-29  273.760010  353.160004  1066.000000  55.349998  501.049988
2025-12-30  273.079987  352.510010  1072.140015  55.279999  503.709991
2025-12-31  271.859985  349.989990  1069.859985  55.000000  502.649994
2026-01-02  271.010010  333.299988  1163.780029  55.950001  496.850006
2026-01-05  267.260010  331.559998  1228.189941  56.889999  498.519989

Ticker             CAT         CME        CRWD         DIS         GLD  …  \
Date                                                                    …
2025-12-29  578.609985  278.420013  475.910004  114.190002  398.600006  …
2025-12-30  577.390015  275.829987  475.630005  114.790001  398.890015  …
2025-12-31  572.869995  273.079987  468.760010  113.769997  396.309998  …
2026-01-02  598.409973  269.679993  453.579987  111.849998  398.279999  …
2026-01-05  616.099976  275.059998  456.549988  114.070000  408.760010  …

Ticker            TSLA         TSM         UNH         VGT         VOO  \
Date
2025-12-29  459.640015  300.920013  328.940002  763.099976  632.599976
2025-12-30  454.429993  299.579987  332.160004  760.890015  631.719971
2025-12-31  449.720001  303.890015  330.109985  753.780029  627.130005
2026-01-02  438.070007  319.609985  336.399994  755.979980  628.299988
2026-01-05  451.670013  322.250000  342.019989  757.419983  632.460022

Ticker            VOOG        VOOV         VTV         VUG         VYM
Date
2025-12-29  448.600006  206.630005  192.589996  492.540009  144.699997
2025-12-30  447.769989  206.410004  192.369995  491.690002  144.550003
2025-12-31  444.589996  204.850006  190.990005  487.859985  143.520004
2026-01-02  444.850006  205.610001  192.809998  486.200012  144.759995
2026-01-05  446.510010  207.509995  194.649994  488.450012  145.820007

[5 rows x 45 columns]
```

[23]:
```
# drop NaN (days without price)
prices = prices_raw.dropna()
display(prices.tail(5))
```

```
Ticker            AAPL        ADBE         ASML        BAC       BRK-B  \
Date
2025-12-29  273.760010  353.160004  1066.000000  55.349998  501.049988
```

```
2025-12-30  273.079987  352.510010  1072.140015  55.279999  503.709991
2025-12-31  271.859985  349.989990  1069.859985  55.000000  502.649994
2026-01-02  271.010010  333.299988  1163.780029  55.950001  496.850006
2026-01-05  267.260010  331.559998  1228.189941  56.889999  498.519989

Ticker             CAT         CME        CRWD         DIS         GLD  …  \
Date                                                                    …
2025-12-29  578.609985  278.420013  475.910004  114.190002  398.600006  …
2025-12-30  577.390015  275.829987  475.630005  114.790001  398.890015  …
2025-12-31  572.869995  273.079987  468.760010  113.769997  396.309998  …
2026-01-02  598.409973  269.679993  453.579987  111.849998  398.279999  …
2026-01-05  616.099976  275.059998  456.549988  114.070000  408.760010  …

Ticker            TSLA         TSM         UNH         VGT         VOO  \
Date
2025-12-29  459.640015  300.920013  328.940002  763.099976  632.599976
2025-12-30  454.429993  299.579987  332.160004  760.890015  631.719971
2025-12-31  449.720001  303.890015  330.109985  753.780029  627.130005
2026-01-02  438.070007  319.609985  336.399994  755.979980  628.299988
2026-01-05  451.670013  322.250000  342.019989  757.419983  632.460022

Ticker            VOOG        VOOV         VTV         VUG         VYM
Date
2025-12-29  448.600006  206.630005  192.589996  492.540009  144.699997
2025-12-30  447.769989  206.410004  192.369995  491.690002  144.550003
2025-12-31  444.589996  204.850006  190.990005  487.859985  143.520004
2026-01-02  444.850006  205.610001  192.809998  486.200012  144.759995
2026-01-05  446.510010  207.509995  194.649994  488.450012  145.820007

[5 rows x 45 columns]
```

[24]:
```python
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: (1180, 45)
prices shape: (458, 45)
Reduction of 722
It can be noticed that the amount of rows in the dataframe dropped from 1180 to
458 after the drop.na() operation. This is because there are a couple of assets
that are relatively new in the public market (post IPO).
```

[25]:
```
print(f"Reminder: the original 'start_date' was: {start_date.date().
↪strftime("%B %d, %Y")}, when the original portfolio was created.")

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


print(f"\nThe asset with the smallest amount of data available is:␣
↪'{first_valid_dates.index[0]}' starting from '{first_valid_dates.iloc[0].
↪date().strftime("%B %d, %Y")}' only.")

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

```
Reminder: the original 'start_date' was: April 25, 2021, when the original
portfolio was created.

The asset with the smallest amount of data available is: 'SERV' starting from
'March 08, 2024' only.

Oldest available dates for each asset:

Ticker
SERV    2024-03-08
NU      2021-12-09
AAPL    2021-04-26
SPYD    2021-04-26
PSA     2021-04-26
QQQM    2021-04-26
QSR     2021-04-26
QYLD    2021-04-26
SOFI    2021-04-26
SOXX    2021-04-26
SPG     2021-04-26
SPYG    2021-04-26
PBR     2021-04-26
TSLA    2021-04-26
TSM     2021-04-26
UNH     2021-04-26
VGT     2021-04-26
VOO     2021-04-26
VOOG    2021-04-26
VOOV    2021-04-26
VTV     2021-04-26
VUG     2021-04-26
PLTR    2021-04-26
```

```
OMAB     2021-04-26
ADBE     2021-04-26
GOOG     2021-04-26
ASML     2021-04-26
BAC      2021-04-26
BRK-B    2021-04-26
CAT      2021-04-26
CME      2021-04-26
CRWD     2021-04-26
DIS      2021-04-26
GLD      2021-04-26
GOOGL    2021-04-26
NVDA     2021-04-26
HD       2021-04-26
KO       2021-04-26
LMT      2021-04-26
MAR      2021-04-26
MCD      2021-04-26
META     2021-04-26
MSFT     2021-04-26
NFLX     2021-04-26
VYM      2021-04-26
Name: First Valid Date, dtype: datetime64[ns]
```

[26]:
```
# -- PLOT ASSET PRICES --
functions.plot_prices(prices=prices, yaxis_label="Price (USD)")
```

6.3 Stats (describe)

[27]:
```
# statistics about prices
prices.describe()
```

[27]:
```
Ticker         AAPL        ADBE         ASML         BAC       BRK-B  \
count    458.000000  458.000000   458.000000  458.000000  458.000000
mean     222.552679  437.168252   835.958809   43.304069  467.779913
std       28.227665   76.543920   137.833934    5.724365   36.503627
min      163.664917  312.399994   590.981628   33.286674  396.730011
25%      204.992336  361.214996   720.596664   38.356688  445.197495
50%      223.477219  438.600006   797.170013   43.228786  472.275009
75%      238.251236  502.427505   954.881851   46.758083  496.355011
max      286.190002  586.549988  1228.189941   56.889999  539.799988

Ticker          CAT         CME        CRWD         DIS         GLD  …  \
count    458.000000  458.000000  458.000000  458.000000  458.000000  …
mean     389.081520  236.470775  387.847784  105.330561  277.980262  …
std       78.722380   32.720417   82.221034   10.323902   56.767730  …
min      270.854095  183.360397  217.889999   80.826134  199.710007  …
```

```
25%     337.269882  203.922478  320.304993   96.814312  230.795002  …
50%     358.955170  237.254982  377.800003  109.514038  263.349991  …
75%     411.348167  268.550484  461.902496  112.741642  309.809990  …
max     625.609985  286.685730  557.530029  123.176605  416.739990  …

Ticker        TSLA         TSM         UNH         VGT         VOO  \
count   458.000000  458.000000  458.000000  458.000000  458.000000
mean    303.332074  201.119318  440.994565  617.778433  536.028584
std      94.636566   49.471145  104.222733   81.988133   50.927276
min     142.050003  124.747917  234.701340  468.845581  445.162018
25%     221.462502  165.846745  331.912491  556.428314  496.731300
50%     302.714996  190.106636  473.674393  605.396118  531.847992
75%     389.190002  231.559891  530.617584  682.324539  574.526367
max     489.880005  322.250000  607.890625  800.707397  634.840027

Ticker        VOOG        VOOV         VTV         VUG         VYM
count   458.000000  458.000000  458.000000  458.000000  458.000000
mean    365.229192  184.176589  169.630523  406.587243  126.395603
std      46.526895   10.670231   11.152038   49.141947    9.622135
min     283.491333  160.399734  148.670639  319.818878  109.263657
25%     331.431778  175.030312  161.032127  370.120850  118.651709
50%     360.575485  183.908020  169.886086  403.221786  126.483810
75%     405.175117  191.088543  176.452358  448.428337  133.308155
max     454.862305  207.509995  194.649994  503.744995  146.816788

[8 rows x 45 columns]
```

6.4 Normalized Prices

Normalize to 100

Normalizar precios de diferentes **magnitudes** dividiendo entre el primer registro. Todos los precios parten del mismo punto que es el 100.

$$\frac{P_t}{P_0} * 100$$

Nota: esto ya no es mas el Precio al Cierre sino un indice de crecimiento en el tiempo.

[28]:
```python
# Normalized Prices

prices_normalized = (prices / prices.iloc[0]) * 100



# -- PLOT ASSET PRICES --
functions.plot_prices(prices=prices_normalized, yaxis_label="Price (USD)")
```

6.5 Daily Returns

TS303_yfinance Indices Bursatiles.ipynb

Normal (simple or arithmetic) returns:

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

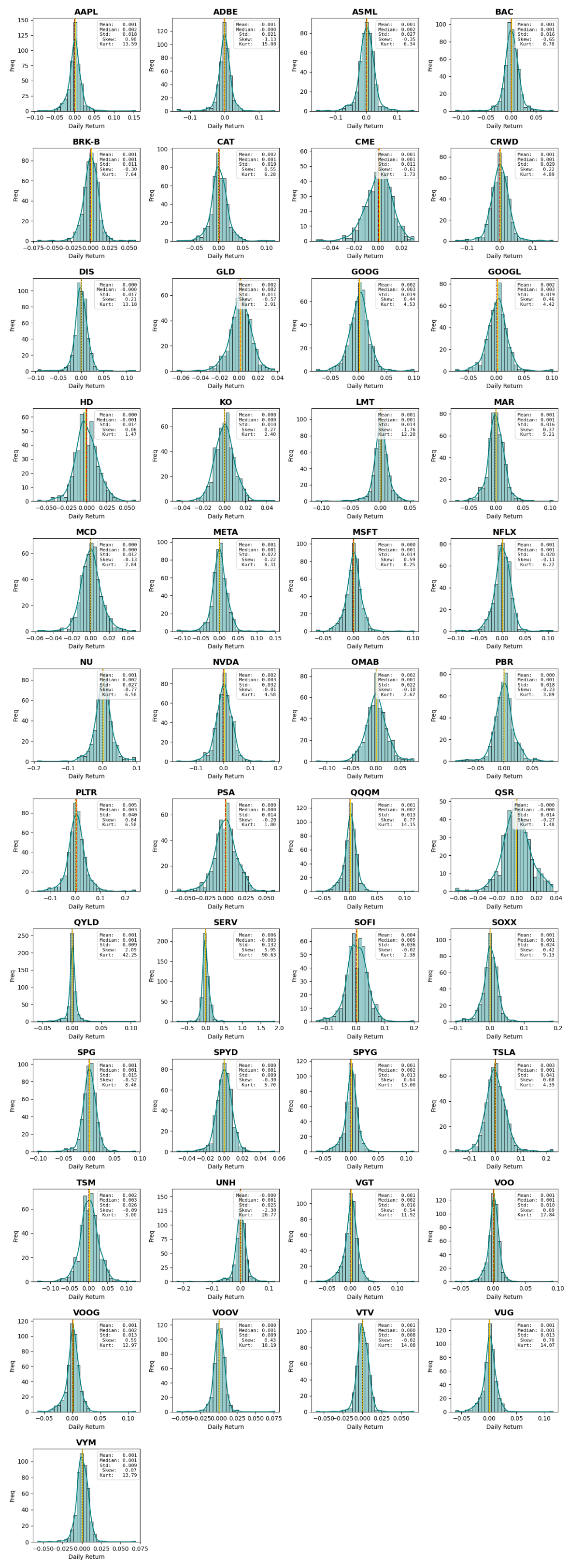
[29]:
```
# Daily Returns. Using pct_change()
# Normal (simple or arithmetic) returns
# Expresed in FRACTION.
# If Percentage is needed then multiply by 100.
daily_returns = prices.pct_change(fill_method=None)
daily_returns = daily_returns.dropna()
daily_returns.tail(5)
```

[29]:
```
Ticker          AAPL      ADBE      ASML       BAC     BRK-B       CAT  \
Date
2025-12-29  0.001317 -0.001809 -0.006292 -0.014599  0.005519 -0.007530
2025-12-30 -0.002484 -0.001841  0.005760 -0.001265  0.005309 -0.002108
2025-12-31 -0.004468 -0.007149 -0.002127 -0.005065 -0.002104 -0.007828
2026-01-02 -0.003127 -0.047687  0.087787  0.017273 -0.011539  0.044583
2026-01-05 -0.013837 -0.005220  0.055345  0.016801  0.003361  0.029562

Ticker           CME      CRWD       DIS       GLD  …      TSLA       TSM  \
Date                                                …
2025-12-29  0.006107 -0.010973  0.005548 -0.043528  … -0.032724 -0.006340
2025-12-30 -0.009303 -0.000588  0.005254  0.000728  … -0.011335 -0.004453
2025-12-31 -0.009970 -0.014444 -0.008886 -0.006468  … -0.010365  0.014387
2026-01-02 -0.012451 -0.032383 -0.016876  0.004971  … -0.025905  0.051729
2026-01-05  0.019950  0.006548  0.019848  0.026313  …  0.031045  0.008260

Ticker           UNH       VGT       VOO      VOOG      VOOV       VTV  \
Date
2025-12-29 -0.008709 -0.005305 -0.003529 -0.004637 -0.001401 -0.000986
2025-12-30  0.009789 -0.002896 -0.001391 -0.001850 -0.001065 -0.001142
2025-12-31 -0.006172 -0.009344 -0.007266 -0.007102 -0.007558 -0.007174
2026-01-02  0.019054  0.002919  0.001866  0.000585  0.003710  0.009529
2026-01-05  0.016706  0.001905  0.006621  0.003732  0.009241  0.009543

Ticker           VUG       VYM
Date
2025-12-29 -0.005090 -0.002688
2025-12-30 -0.001726 -0.001037
2025-12-31 -0.007789 -0.007126
2026-01-02 -0.003403  0.008640
2026-01-05  0.004628  0.007323
```

```
[5 rows x 45 columns]
```

[30]:
```
results_df = functions.plot_returns_distributions(daily_returns, num_cols=4)
```

Portfolio Daily Returns: Distribution & Statistical Moments



INTERPRETATION GUIDE:
• SKEWNESS: Positive (>0) = Tail on the right (occasional big gains). Negative (<0) = Tail on the left (occasional big crashes).
• KURTOSIS: High (Excess >0) = 'Fat Tails'. Higher probability of extreme price movements (outliers) than a normal distribution.
• MEAN vs MEDIAN: If Mean (Red) > Median (Gold), the distribution is positively skewed, pulled by large positive outliers.
• ZERO LINE: The dotted black line represents 0% return.

```
================================================================================
FAT-TAIL RISK RANKING (Sorted by Highest Kurtosis)
================================================================================

<pandas.io.formats.style.Styler at 0x27cbf6ab170>
```

```
[31]: # -- PLOT DAILY RETURNS --
      functions.plot_daily_returns(daily_returns)
```

6.6 Summary of Daily Returns

```
[32]: # 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().strftime("%B %d, %Y")} until␣
       ↪{today.date().strftime("%B %d, %Y")}")

      max_daily_return = daily_returns.describe().loc['max'].
       ↪sort_values(ascending=False)
      print(f"\nThe 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}%")

      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}%")
```

```
Number of days of evaluation: 666
since March 11, 2024 until January 06, 2026

The asset with the biggest Return in a single day is 'SERV' with 187.07%
The asset with the lowest Return in a single day is 'SERV' with -77.647%
The average of all Daily Returns is 0.1137%

The asset with the biggest StdDev in Daily Returns is 'SERV' with 13.175%
The asset with the smallest StdDev in Daily Returns is 'VTV' with 0.83946%
```

6.7 Annualized Returns

```
[33]: # Annualized Returns
annualized_return_tickers = daily_returns.mean() * 250
annualized_return_tickers.name = "Annualized Returns:"
annualized_return_tickers = annualized_return_tickers.
 ↪sort_values(ascending=False)
print("Annualized Returns (%):")
print(f"{round(annualized_return_tickers, 2)* 100}")
```

```
Annualized Returns (%):
Ticker
SERV     153.0
PLTR     123.0
SOFI      90.0
TSLA      72.0
NVDA      55.0
TSM       53.0
GOOGL     51.0
GOOG      51.0
GLD       40.0
CAT       39.0
OMAB      38.0
NU        36.0
BAC       31.0
CRWD      29.0
AAPL      29.0
NFLX      28.0
SOXX      26.0
SPYG      25.0
VOOG      25.0
VGT       24.0
VUG       23.0
ASML      22.0
QQQM      22.0
META      21.0
VOO       19.0
CME       19.0
SPG       18.0
MAR       16.0
VYM       16.0
VTV       15.0
LMT       14.0
BRK-B     13.0
QYLD      13.0
VOOV      12.0
```

```
KO        12.0
MSFT      12.0
SPYD      11.0
DIS        7.0
PBR        6.0
MCD        5.0
PSA        1.0
HD         0.0
QSR       -3.0
UNH       -8.0
ADBE     -22.0
Name: Annualized Returns:, dtype: float64

Ticker
SERV     153.0
PLTR     123.0
SOFI      90.0
TSLA      72.0
NVDA      55.0
TSM       53.0
GOOGL     51.0
GOOG      51.0
GLD       40.0
CAT       39.0
OMAB      38.0
NU        36.0
BAC       31.0
CRWD      29.0
AAPL      29.0
NFLX      28.0
SOXX      26.0
SPYG      25.0
VOOG      25.0
VGT       24.0
VUG       23.0
ASML      22.0
QQQM      22.0
META      21.0
VOO       19.0
CME       19.0
SPG       18.0
MAR       16.0
VYM       16.0
VTV       15.0
LMT       14.0
BRK-B     13.0
QYLD      13.0
VOOV      12.0
```

```
KO        12.0
MSFT      12.0
SPYD      11.0
DIS        7.0
PBR        6.0
MCD        5.0
PSA        1.0
HD         0.0
QSR       -3.0
UNH       -8.0
ADBE     -22.0
Name: Annualized Returns:, dtype: float64
```

6.8 Cumulative Returns in Period

TS303_yfinance Indices Bursatiles.ipynb

[34]:
```
# 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 = (cumulative_returns - 1)*100
print("Cumulative Returns in %:")
cumulative_returns.tail(5)
```

```
Cumulative Returns in %:
```

[34]:
```
Ticker           AAPL       ADBE       ASML        BAC      BRK-B        CAT  \
Date
2025-12-29  61.654779 -35.985789   8.910560  61.985575  24.283764  75.127795
2025-12-30  61.253226 -36.103607   9.537870  61.780716  24.943569  74.758547
2025-12-31  60.532818 -36.560389   9.304925  60.961281  24.680640  73.390473
2026-01-02  60.030909 -39.585639  18.900502  63.741523  23.241973  81.120654
2026-01-05  57.816541 -39.901032  25.481102  66.492494  23.656207  86.474885

Ticker            CME       CRWD       DIS         GLD  …        TSLA  \
Date                                                    …
2025-12-29  38.696582  47.409012  5.601360   97.688834  …  162.142138
2025-12-30  37.406345  47.322285  6.156230   97.832666  …  159.170755
2025-12-31  36.036416  45.194363  5.212944   96.553086  …  156.484550
2026-01-02  34.342689  40.492482  3.437356   97.530124  …  149.840320
2026-01-05  37.022770  41.412414  5.490383  102.727769  …  157.596682

Ticker             TSM        UNH        VGT        VOO       VOOG       VOOV  \
Date
2025-12-29  111.308356 -28.452585  47.946322  37.928557  51.768926  22.272159
2025-12-30  110.367379 -27.752206  47.517865  37.736686  51.488117  22.141975
```

```
2025-12-31  113.393914 -28.198104  46.139413  36.735916  50.412272  21.218855
2026-01-02  124.432598 -26.829970  46.565930  36.991012  50.500238  21.668577
2026-01-05  126.286437 -25.607571  46.845111  37.898043  51.061845  22.792888

Ticker            VTV        VUG        VYM
Date
2025-12-29  27.397622  46.504114  30.200598
2025-12-30  27.252092  46.251283  30.065634
2025-12-31  26.339234  45.112059  29.138844
2026-01-02  27.543152  44.618307  30.254584
2026-01-05  28.760304  45.287560  31.208380

[5 rows x 45 columns]
```

[35]:
```python
# -- PLOT CUMULATIVE RETURNS --
functions.plot_cumulative_returns(cumulative_returns)
```

6.9 Final Cumulated Returns in Period

[36]:
```python
# 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().strftime("%B %d, %Y")} until␣
 ↪{today.date().strftime("%B %d, %Y")}")

cumulative_returns_final = cumulative_returns.iloc[-1]
cumulative_returns_final.name = "Final Cumulative Returns (%)"
cumulative_returns_final = round(cumulative_returns_final.
 ↪sort_values(ascending=False) , 1)
print("Final Cumulative Returns in %:")
display(cumulative_returns_final.all)
```

```
Final Cumulative Returns in (%) in 666 days of evaluation:
since March 11, 2024 until January 06, 2026
Final Cumulative Returns in %:

<bound method Series.all of Ticker
PLTR     568.4
SOFI     279.8
TSLA     157.6
GOOGL    135.5
GOOG     134.6
TSM      126.3
NVDA     115.0
GLD      102.7
CAT       86.5
OMAB      79.5
BAC       66.5
```

```
NU         62.1
AAPL       57.8
NFLX       51.2
SPYG       51.1
VOOG       51.1
VGT        46.8
VUG        45.3
QQQM       42.5
CRWD       41.4
SOXX       41.3
VOO        37.9
CME        37.0
SPG        32.3
VYM        31.2
META       31.0
VTV        28.8
MAR        26.5
ASML       25.5
QYLD       24.3
LMT        23.9
BRK-B      23.7
VOOV       22.8
KO         21.1
SPYD       19.6
MSFT       17.9
MCD         6.8
DIS         5.5
PBR         3.5
PSA        -3.5
HD         -3.8
QSR       -10.2
UNH       -25.6
ADBE      -39.9
SERV      -46.7
Name: Final Cumulative Returns (%), dtype: float64>
```

6.10 Summary Final Cumulated Returns

[37]:
```python
print(f"Number of days of evaluation (period): {(today - cumulative_returns.
  ↪index[0]).days}")
print(f"From {cumulative_returns.index[0].date().strftime("%B %d, %Y")} until␣
  ↪{today.date().strftime("%B %d, %Y")}")

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

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

```
Number of days of evaluation (period): 666
From March 11, 2024 until January 06, 2026
The asset with the best cumulated return in the Period 'PLTR' with 568.4% in the
period
The asset with the worst cumulative return in the Period 'SERV' with -46.7% in
the period
```

6.11 Risk, Annualized Volatility

TS303_yfinance Indices Bursatiles.ipynb

Asset Volatility (risk or std)

$$\sigma = risk = \sqrt{\frac{\sum(r-\bar{r})^2}{n-1}}$$

[38]:
```
# 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:

        Volatility (%)
Ticker
SERV            209.15
TSLA             64.71
PLTR             63.15
SOFI             57.83
NVDA             50.56
CRWD             45.66
```

```
ASML               43.30
NU                 42.78
TSM                40.79
UNH                39.95
SOXX               37.78
META               35.01
OMAB               34.40
ADBE               34.03
NFLX               31.53
GOOGL              30.36
GOOG               29.97
CAT                29.92
PBR                29.26
AAPL               28.62
DIS                27.16
MAR                25.89
VGT                25.43
BAC                25.18
SPG                23.62
LMT                22.85
PSA                22.66
QSR                22.63
MSFT               22.50
HD                 22.24
QQQM               21.11
VUG                21.04
SPYG               20.98
VOOG               20.90
MCD                18.39
GLD                18.22
CME                17.54
BRK-B              17.04
VOO                16.15
KO                 15.68
QYLD               14.87
SPYD               14.34
VYM                13.64
VOOV               13.62
VTV                13.33
```

Stats of Volatility

[39]:
```
display(volatility_df.describe()) # all stocks
```

```
       Volatility (%)
count       45.000000
mean        32.794889
std         29.833583
min         13.330000
```

```
25%          20.900000
50%          25.430000
75%          35.010000
max         209.150000
```

6.12 Summary Risk, Volatility

[40]:
```python
# summary and stats

print(f"Number of days of evaluation: {(today - cumulative_returns.index[0]).
 ↪days}")
print(f"From {cumulative_returns.index[0].date().strftime("%B %d, %Y")} until␣
 ↪{today.date().strftime("%B %d, %Y")}")

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: 666
From March 11, 2024 until January 06, 2026
The asset with the biggest Annualized Volatility is 'SERV' with a 209.15 %
The asset with the lowest Annualized Volatility is 'VTV' with a 13.33 %
The Average Annualized Volatility of all assets is: 32.795 %
```

6.13 Dividend Yields (%)

[41]:
```python
# ANNUAL DIVIDEND YIELDS
yields = functions.plot_annual_dividnd_yields(daily_returns)
```

```
Annual Dividend Yields from yf 'dividendYield' (%):

AAPL      0.39
ADBE      0.00
ASML      0.60
BAC       1.97
BRK-B     0.00
CAT       0.98
CME       1.82
CRWD      0.00
DIS       1.31
GLD       0.00
GOOG      0.26
GOOGL     0.27
HD        2.67
KO        3.00
```

```
LMT       2.70
MAR       0.86
MCD       2.48
META      0.32
MSFT      0.77
NFLX      0.00
NU        0.00
NVDA      0.02
OMAB      4.48
PBR      14.31
PLTR      0.00
PSA       4.60
QQQM      0.49
QSR       3.72
QYLD     10.46
SERV      0.00
SOFI      0.00
SOXX      0.55
SPG       4.81
SPYD      4.46
SPYG      0.53
TSLA      0.00
TSM       1.04
UNH       2.58
VGT       0.41
VOO       1.12
VOOG      0.48
VOOV      1.80
VTV       2.05
VUG       0.42
VYM       2.42
Name: dividendYield, dtype: float64
```

7 Initial Portfolio (Original)

7.1 Assets Weights

Compute Assets weights from the original portfolio

[42]:
```
# call the original portfolio [Ticker, Current QTY]
file_df

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

# Merge original df with Close Prices
weights_df = pd.merge(file_df, 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)
print("Sorted by Invested amount ($):")
display(round(weights_df, 2))
```

```
Total Invested: $68,831.74
The sum of the weights is: 1.0
Sorted by Invested amount ($):

        Current QTY  Close Price  Investment  Weights
Ticker
NVDA         100.53       188.12    18911.48     0.27
MSFT          19.75       472.85     9339.75     0.14
AAPL          17.04       267.26     4554.12     0.07
PLTR          21.02       174.04     3658.95     0.05
TSM           10.03       322.25     3233.66     0.05
VOOG           6.30       446.51     2811.39     0.04
GLD            6.45       408.76     2636.24     0.04
TSLA           4.92       451.67     2224.30     0.03
VOOV          10.36       207.51     2150.75     0.03
QYLD         100.86        17.76     1791.28     0.03
UNH            5.04       342.02     1722.64     0.03
QQQM           6.38       254.43     1622.13     0.02
VGT            2.02       757.42     1529.25     0.02
DIS           13.03       114.07     1486.23     0.02
VOO            2.29       632.46     1447.28     0.02
META           1.58       658.79     1039.45     0.02
PBR           87.44        11.74     1026.60     0.01
SOXX           3.05       318.06      968.76     0.01
CRWD           1.46       456.55      664.40     0.01
GOOGL          2.01       316.54      636.79     0.01
CME            2.00       275.06      550.12     0.01
GOOG           1.61       317.32      512.10     0.01
VUG            1.01       488.45      492.50     0.01
KO             6.23        67.94      423.06     0.01
QSR            5.26        66.74      350.73     0.01
```

```
ASML           0.26     1228.19     324.45     0.00
MCD            1.05      299.86     315.95     0.00
VYM            2.10      145.82     306.92     0.00
NU            13.62       17.94     244.39     0.00
OMAB           2.16      109.26     236.47     0.00
CAT            0.33      616.10     205.93     0.00
LMT            0.32      511.57     163.25     0.00
SPYD           3.65       43.69     159.31     0.00
SOFI           5.32       29.28     155.76     0.00
ADBE           0.47      331.56     154.20     0.00
MAR            0.46      311.03     142.13     0.00
BRK-B          0.24      498.52     122.03     0.00
NFLX           1.20       91.46     109.50     0.00
SPYG           1.02      107.16     109.19     0.00
BAC            1.47       56.89      83.56     0.00
VTV            0.40      194.65      78.82     0.00
PSA            0.19      260.90      49.62     0.00
SERV           3.36       12.68      42.67     0.00
SPG            0.20      183.11      36.40     0.00
HD             0.02      344.09       7.21     0.00
```

TO-DO: Grafica de Pastel con los Porcentajes

Grafica por Sectores, Industrias

7.2 Portfolio Daily Returns

[43]:
```python
#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]")
```

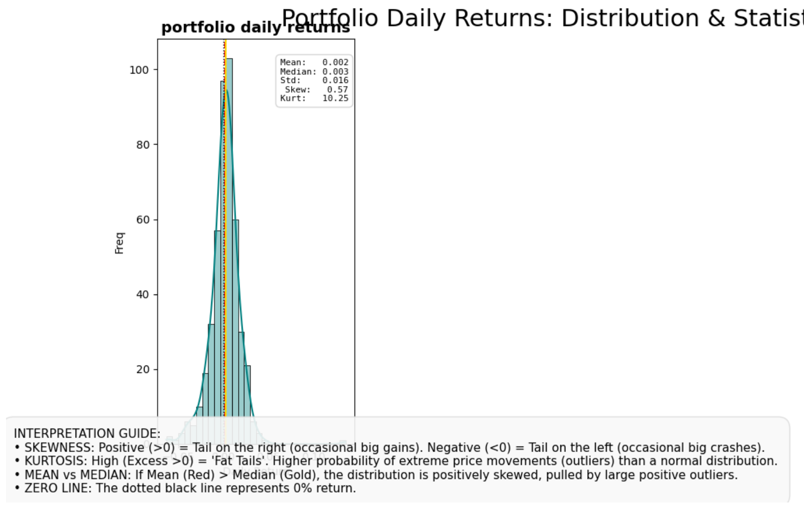
```
Portfolio daily returns = [Daily Returns] x [weights]
daily_returns shape: (457, 45)
weights_df shape: (45,)
Matrix multiplication [457 x 45] x [45 x 1] = [457 x 1]
```

[44]:
```python
# 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
2024-03-11   -0.009023
2024-03-12    0.027501
2024-03-13   -0.006010
2024-03-14   -0.009048
2024-03-15   -0.009670
                ...
2025-12-29   -0.008831
2025-12-30   -0.002100
2025-12-31   -0.006179
2026-01-02    0.000196
2026-01-05    0.005278
Name: portfolio daily returns, Length: 457, dtype: float64
```

```
[45]: results_df = functions.plot_returns_distributions(pd.
       ↪DataFrame(portfolio_daily_returns), num_cols=4)
```



INTERPRETATION GUIDE:
• SKEWNESS: Positive (>0) = Tail on the right (occasional big gains). Negative (<0) = Tail on the left (occasional big crashes).
• KURTOSIS: High (Excess >0) = 'Fat Tails'. Higher probability of extreme price movements (outliers) than a normal distribution.
• MEAN vs MEDIAN: If Mean (Red) > Median (Gold), the distribution is positively skewed, pulled by large positive outliers.
• ZERO LINE: The dotted black line represents 0% return.

```
================================================================================
FAT-TAIL RISK RANKING (Sorted by Highest Kurtosis)
================================================================================

<pandas.io.formats.style.Styler at 0x27cb6930fe0>
```

```
[46]: # -- PLOT DAILY RETURNS --
      functions.plot_daily_returns(pd.DataFrame(portfolio_daily_returns))
```

7.3 Portfolio Annualized Returns

[47]:
```
# Annualized Returns
annualized_return_portfolio = portfolio_daily_returns.mean() * 250
print("Annualized Returns (%):")
print(f"{round(annualized_return_portfolio, 2) * 100}%")
```

```
Annualized Returns (%):
38.0%
```

7.4 Portfolio Cumulative Returns

[48]:
```
# Portfolio's Cumulative daily Returns
portfolio_cumulative_returns = (1 + portfolio_daily_returns).cumprod()
portfolio_cumulative_returns = (portfolio_cumulative_returns - 1) * 100
portfolio_cumulative_returns = portfolio_cumulative_returns.rename('portfolio␣
↪cumulative returns')
print("Portfolio Daily Cumulative Returns:")
display(portfolio_cumulative_returns)
```

```
Portfolio Daily Cumulative Returns:

Date
2024-03-11    -0.902260
2024-03-12     1.823045
2024-03-13     1.211137
2024-03-14     0.295371
2024-03-15    -0.674509
                 …
2025-12-29    88.416738
2025-12-30    88.021052
2025-12-31    86.859277
2026-01-02    86.895959
2026-01-05    87.882465
Name: portfolio cumulative returns, Length: 457, dtype: float64
```

[49]:
```
# -- PLOT CUMULATIVE RETURNS --
functions.plot_cumulative_returns(pd.DataFrame(portfolio_cumulative_returns))
```

Portfolio Cumulative Final

[50]:
```
# Portfolio's final comulated return
portafolio_cumulative_final = portfolio_cumulative_returns.iloc[-1]
print(f"portafolio_cumulative Returns_final: {portafolio_cumulative_final:,.
↪4}%")
```

```
portafolio_cumulative Returns_final: 87.88%
```

7.5 Portfolio Volatility (Annualized)

[51]:
```
# 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: 26.13%
```

7.6 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.

[52]:
```
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: 1.3
```

7.7 Summary

[53]:
```
print(f"Number of days of evaluation: {(today - cumulative_returns.index[0]).
  ↪days} days")
print(f"From {cumulative_returns.index[0].date().strftime("%B %d, %Y")} until␣
  ↪{today.date().strftime("%B %d, %Y")}")

print(f"\nTotal Invested: ${Total_invested:,.2f}")
print(f"Portafolio_cumulative returns_final (all period):␣
  ↪{portafolio_cumulative_final:,.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: 666 days
From March 11, 2024 until January 06, 2026

Total Invested: $68,831.74
Portafolio_cumulative returns_final (all period): 87.88%
```

```
Portfolio Annualized Average Returns: 38.1843%
Portfolio Annualized Volatility: 26.13%
Portfolio Sharpe Ratio: 1.3
```

8 Benchmarks (Indices)

8.1 Daily Index Levels

[54]:
```
# 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_prices = yf.download(list(benchmark_indices.values()),␣
 ↪start=first_valid_dates.iloc[0], end=today, auto_adjust=True)["Close"]

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

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

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

Benchmark Indices levels:


            EE.UU. (S&P 500)  EE.UU. (NASDAQ)  EE.UU. (DJIA)  \
Date
2025-12-29      48461.929688      9866.500000   24351.119141
2025-12-30      48367.058594      9940.700195   24490.410156
2025-12-31      48063.289062      9931.400391            NaN
2026-01-02      48382.390625      9951.099609   24539.339844
2026-01-05      48977.179688     10004.599609   24868.689453

            EE.UU. (Russell 100)  México (IPC)  Japón (Nikkei 225)  \
Date
2025-12-29           6905.740234  23474.349609        65347.078125
2025-12-30           6896.240234  23419.080078        64366.699219
2025-12-31           6845.500000  23241.990234        64308.289062
```

```
2026-01-02           6858.470215  23235.630859        64141.359375
2026-01-05           6902.049805  23395.820312        65014.371094

            Alemania (DAX)  Reino Unido (FTSE 100)
Date
2025-12-29    50526.921875             3766.949951
2025-12-30    50339.480469             3761.540039
2025-12-31             NaN             3732.870117
2026-01-02             NaN             3742.669922
2026-01-05    51832.800781             3769.020020
```

[55]:
```
# -- PLOT ASSET PRICES --
functions.plot_prices(prices=benchmarks_prices, yaxis_label="Index Level")
```

8.2 Normalized Index Levels

[56]:
```
# Normalized Index Levels
benchmarks_prices_normalized = (benchmarks_prices / benchmarks_prices.iloc[0])␣
  ↪* 100

# -- PLOT ASSET PRICES --
functions.plot_prices(prices=benchmarks_prices_normalized, yaxis_label="Index␣
  ↪Level")
```

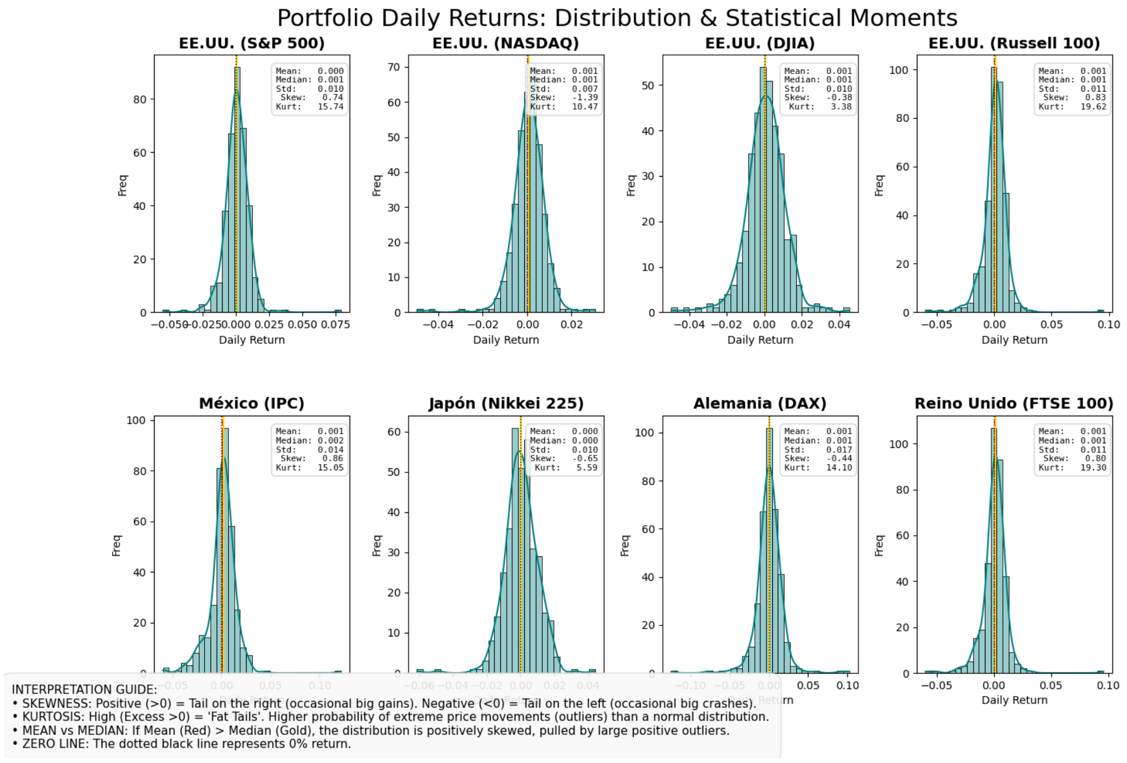
8.3 Daily Returns

[57]:
```
# Daily Returns
daily_returns_bm = benchmarks_prices.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)
```

[57]:
```
            EE.UU. (S&P 500)  EE.UU. (NASDAQ)  EE.UU. (DJIA)  \
Date
2025-12-18          0.001376         0.006497       0.009971
2025-12-19          0.003817         0.006058       0.003674
2025-12-22          0.004732        -0.003173      -0.000182
2025-12-23          0.001649         0.002352       0.002310
2025-12-30         -0.001958         0.007520       0.005720

            EE.UU. (Russell 100)  México (IPC)  Japón (Nikkei 225)  \
Date
2025-12-18              0.007934      0.013794            0.020459
2025-12-19              0.008818      0.013095            0.002502
2025-12-22              0.006436      0.005200            0.012682
2025-12-23              0.004550      0.005677            0.012616
```

```
2025-12-30            -0.001376     -0.002354           -0.015003

            Alemania (DAX)  Reino Unido (FTSE 100)
Date
2025-12-18       -0.010316                0.007753
2025-12-19        0.010320                0.008832
2025-12-22        0.018082                0.006793
2025-12-23        0.000208                0.003815
2025-12-30       -0.003710               -0.001436
```

[58]:
```python
results_df = functions.plot_returns_distributions(daily_returns_bm, num_cols=4)
```



INTERPRETATION GUIDE:
• SKEWNESS: Positive (>0) = Tail on the right (occasional big gains). Negative (<0) = Tail on the left (occasional big crashes).
• KURTOSIS: High (Excess >0) = 'Fat Tails'. Higher probability of extreme price movements (outliers) than a normal distribution.
• MEAN vs MEDIAN: If Mean (Red) > Median (Gold), the distribution is positively skewed, pulled by large positive outliers.
• ZERO LINE: The dotted black line represents 0% return.

```
================================================================================
FAT-TAIL RISK RANKING (Sorted by Highest Kurtosis)
================================================================================

<pandas.io.formats.style.Styler at 0x27cb5083a70>
```

[59]:
```python
functions.plot_daily_returns(daily_returns_bm)
```

8.4 Annualized Returns

[60]:
```
# Annualized Returns
annualized_return_bm = daily_returns_bm.mean() * 252
annualized_return_bm.name = "Annualized Returns (%):"
annualized_return_bm = annualized_return_bm.sort_values(ascending=False)
print("Annualized Returns (%):")
print(round(annualized_return_bm, 2)*100)
```

```
Annualized Returns (%):
México (IPC)             27.0
EE.UU. (DJIA)            17.0
EE.UU. (Russell 100)     17.0
Reino Unido (FTSE 100)   16.0
Alemania (DAX)           15.0
EE.UU. (NASDAQ)          13.0
EE.UU. (S&P 500)         11.0
Japón (Nikkei 225)        8.0
Name: Annualized Returns (%):, dtype: float64
```

8.5 Cumulative Daily Returns

[61]:
```
cumulative_returns_bm = (1 + daily_returns_bm).cumprod()
cumulative_returns_bm = (cumulative_returns_bm - 1) * 100
display(cumulative_returns_bm.tail(5))
```

```
            EE.UU. (S&P 500)  EE.UU. (NASDAQ)  EE.UU. (DJIA)  \
Date
2025-12-18         13.266130        17.128699      23.859387
2025-12-19         13.698483        17.838302      24.314403
2025-12-22         14.236539        17.464450      24.291731
2025-12-23         14.424869        17.740671      24.578813
2025-12-30         14.200866        18.626130      25.291417

            EE.UU. (Russell 100)  México (IPC)  Japón (Nikkei 225)  \
Date
2025-12-18             22.352113     37.085787            8.299930
2025-12-19             23.431021     38.880875            8.570921
2025-12-22             24.225485     39.603123            9.947766
2025-12-23             24.790759     40.395675           11.334842
2025-12-30             24.619089     40.065119            9.664525

            Alemania (DAX)  Reino Unido (FTSE 100)
Date
2025-12-18       14.406983               21.228400
2025-12-19       15.587699               22.299108
2025-12-22       17.677733               23.129847
2025-12-23       17.702203               23.599630
2025-12-30       17.265559               23.422122
```

[62]:
```
functions.plot_cumulative_returns(cumulative_returns_bm)
```

[63]:
```
# 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_bm.iloc[-1]
cumulative_returns_final_bm = cumulative_returns_final_bm.
 ↪sort_values(ascending=False)
cumulative_returns_final_bm = cumulative_returns_final_bm.rename("final␣
 ↪cumulative returns Benchmarks (%)")
display(cumulative_returns_final_bm)
```

```
Final Cumulative Returns (%)

México (IPC)              40.065119
EE.UU. (DJIA)             25.291417
EE.UU. (Russell 100)      24.619089
Reino Unido (FTSE 100)    23.422122
EE.UU. (NASDAQ)           18.626130
Alemania (DAX)            17.265559
EE.UU. (S&P 500)          14.200866
Japón (Nikkei 225)         9.664525
Name: final cumulative returns Benchmarks (%), dtype: float64
```

8.6 Volatility

[64]:
```
#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)            27.297079
México (IPC)              22.904904
Reino Unido (FTSE 100)    17.402071
EE.UU. (Russell 100)      17.223190
EE.UU. (DJIA)             16.435356
Japón (Nikkei 225)        15.980321
EE.UU. (S&P 500)          15.286125
EE.UU. (NASDAQ)           11.685753
Name: Volatility Benchmarks (%), dtype: float64
```

8.7 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)

[65]:
```
# Sharpe Ratios
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)
```

```
México (IPC)             0.98
EE.UU. (DJIA)            0.81
EE.UU. (Russell 100)     0.75
EE.UU. (NASDAQ)          0.74
Reino Unido (FTSE 100)   0.71
EE.UU. (S&P 500)         0.42
Alemania (DAX)           0.40
Japón (Nikkei 225)       0.23
Name: Sharpe Ratio, dtype: float64
```

8.8 Summary

[66]:
```
# 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().strftime("%B %d, %Y")} to:␣
  ↪{today.date().strftime("%B %d, %Y")}")

# 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: 669 days.
From: March 08, 2024 to: January 06, 2026

Summary: Benchmark Indices:
```

```
'México (IPC)' has the largest total return 40.065%
and'Japón (Nikkei 225)' the lowest total return 9.66%

'Alemania (DAX)' has the largest volatility 27.297%
and 'EE.UU. (NASDAQ)' the lowest volatility 11.686%
```

9 Compare Initial Portfolio and Indices

9.1 Daily Returns

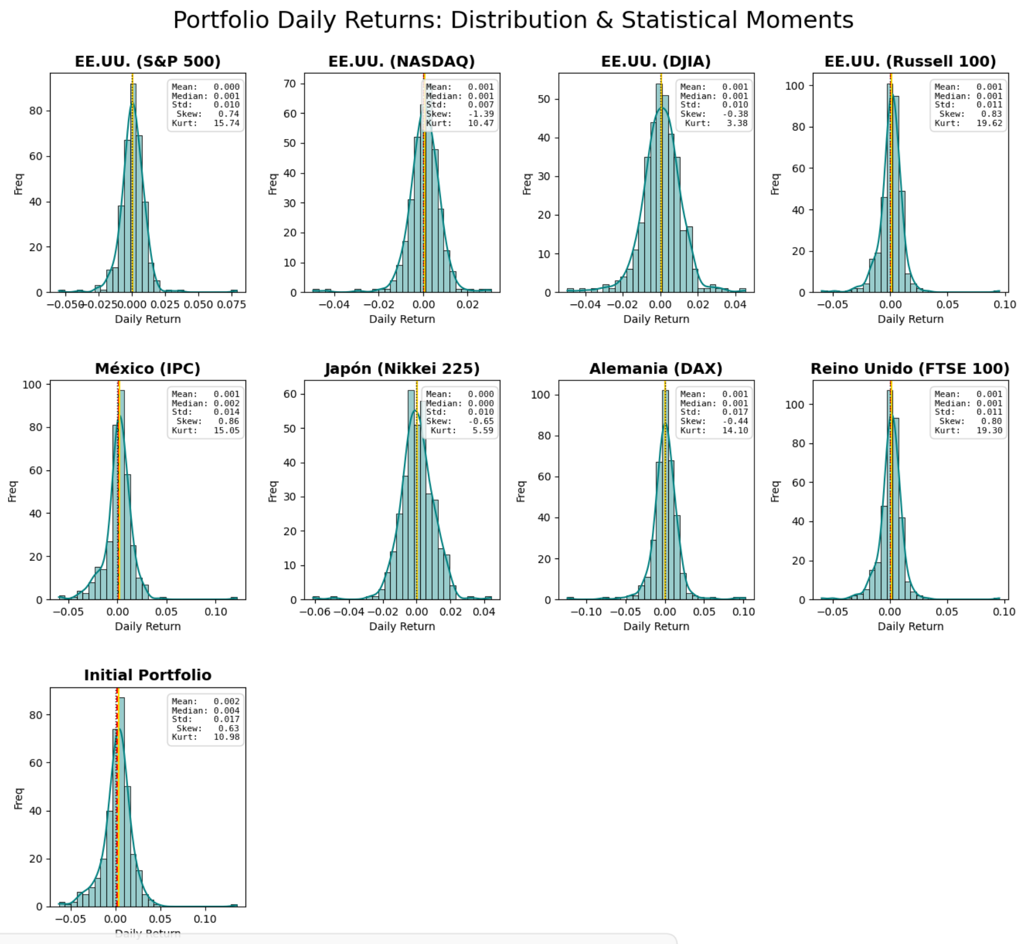
Combine dataframes

[67]:
```
# 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")
    print("There are more dates in one dataframe than the other")


# 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 457 vs 354 but a left merge will help
There are more dates in one dataframe than the other

Daily Returns (merged portfolio and indices):

            EE.UU. (S&P 500)  EE.UU. (NASDAQ)  EE.UU. (DJIA)  \
Date
2025-12-18          0.001376         0.006497       0.009971
2025-12-19          0.003817         0.006058       0.003674
2025-12-22          0.004732        -0.003173      -0.000182
```

```
2025-12-23          0.001649         0.002352       0.002310
2025-12-30         -0.001958         0.007520       0.005720

            EE.UU. (Russell 100)  México (IPC)  Japón (Nikkei 225)  \
Date
2025-12-18              0.007934      0.013794            0.020459
2025-12-19              0.008818      0.013095            0.002502
2025-12-22              0.006436      0.005200            0.012682
2025-12-23              0.004550      0.005677            0.012616
2025-12-30             -0.001376     -0.002354           -0.015003

            Alemania (DAX)  Reino Unido (FTSE 100)  Initial Portfolio
Date
2025-12-18       -0.010316                0.007753           0.014941
2025-12-19        0.010320                0.008832           0.017300
2025-12-22        0.018082                0.006793           0.007409
2025-12-23        0.000208                0.003815           0.011454
2025-12-30       -0.003710               -0.001436          -0.002100
```

[68]:
```
results_df = functions.plot_returns_distributions(merge_daily_returns,␣
  ↪num_cols=4)
```



INTERPRETATION GUIDE:
• SKEWNESS: Positive (>0) = Tail on the right (occasional big gains). Negative (<0) = Tail on the left (occasional big crashes).
• KURTOSIS: High (Excess >0) = 'Fat Tails'. Higher probability of extreme price movements (outliers) than a normal distribution.
• MEAN vs MEDIAN: If Mean (Red) > Median (Gold), the distribution is positively skewed, pulled by large positive outliers.
• ZERO LINE: The dotted black line represents 0% return.

```
================================================================================
FAT-TAIL RISK RANKING (Sorted by Highest Kurtosis)
================================================================================

<pandas.io.formats.style.Styler at 0x27cbd1bb3b0>
```

[69]:
```
functions.plot_daily_returns(merge_daily_returns)
```

9.2 Cumulative Returns

[70]:
```
# 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 457 vs 354 but a left merge will help

Daily Cumulative Returns (merged portfolio and indices):
            EE.UU. (S&P 500)  EE.UU. (NASDAQ)  EE.UU. (DJIA)  \
Date
2025-12-18         13.266130        17.128699      23.859387
2025-12-19         13.698483        17.838302      24.314403
2025-12-22         14.236539        17.464450      24.291731
2025-12-23         14.424869        17.740671      24.578813
2025-12-30         14.200866        18.626130      25.291417

            EE.UU. (Russell 100)  México (IPC)  Japón (Nikkei 225)  \
Date
2025-12-18             22.352113     37.085787            8.299930
2025-12-19             23.431021     38.880875            8.570921
2025-12-22             24.225485     39.603123            9.947766
2025-12-23             24.790759     40.395675           11.334842
2025-12-30             24.619089     40.065119            9.664525

            Alemania (DAX)  Reino Unido (FTSE 100)  Initial Portfolio
Date
2025-12-18       14.406983               21.228400          82.948254
2025-12-19       15.587699               22.299108          86.113241
2025-12-22       17.677733               23.129847          87.492159
2025-12-23       17.702203               23.599630          89.639700
2025-12-30       17.265559               23.422122          88.021052
```

[71]:
```
functions.plot_cumulative_returns(merge_cumulative_returns)
```

[72]:
```
# Total Cumulative Returns
print("\nTotal (Final) cumulative Returns (%):")
merge_cumulative_returns_finals = merge_cumulative_returns.iloc[-1].copy()
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          88.021052
México (IPC)               40.065119
EE.UU. (DJIA)              25.291417
EE.UU. (Russell 100)       24.619089
Reino Unido (FTSE 100)     23.422122
EE.UU. (NASDAQ)            18.626130
Alemania (DAX)             17.265559
EE.UU. (S&P 500)           14.200866
Japón (Nikkei 225)          9.664525
Name: Final Cumulative Returns (%), dtype: float64
```

9.3 Volatility

[73]:
```
# 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 (%):

Alemania (DAX)             27.297079
Initial Portfolio          27.296752
México (IPC)               22.904904
Reino Unido (FTSE 100)     17.402071
EE.UU. (Russell 100)       17.223190
EE.UU. (DJIA)              16.435356
Japón (Nikkei 225)         15.980321
EE.UU. (S&P 500)           15.286125
EE.UU. (NASDAQ)            11.685753
Name: Annualized Volatility (%), dtype: float64
```

9.4 Sharpe Ratio

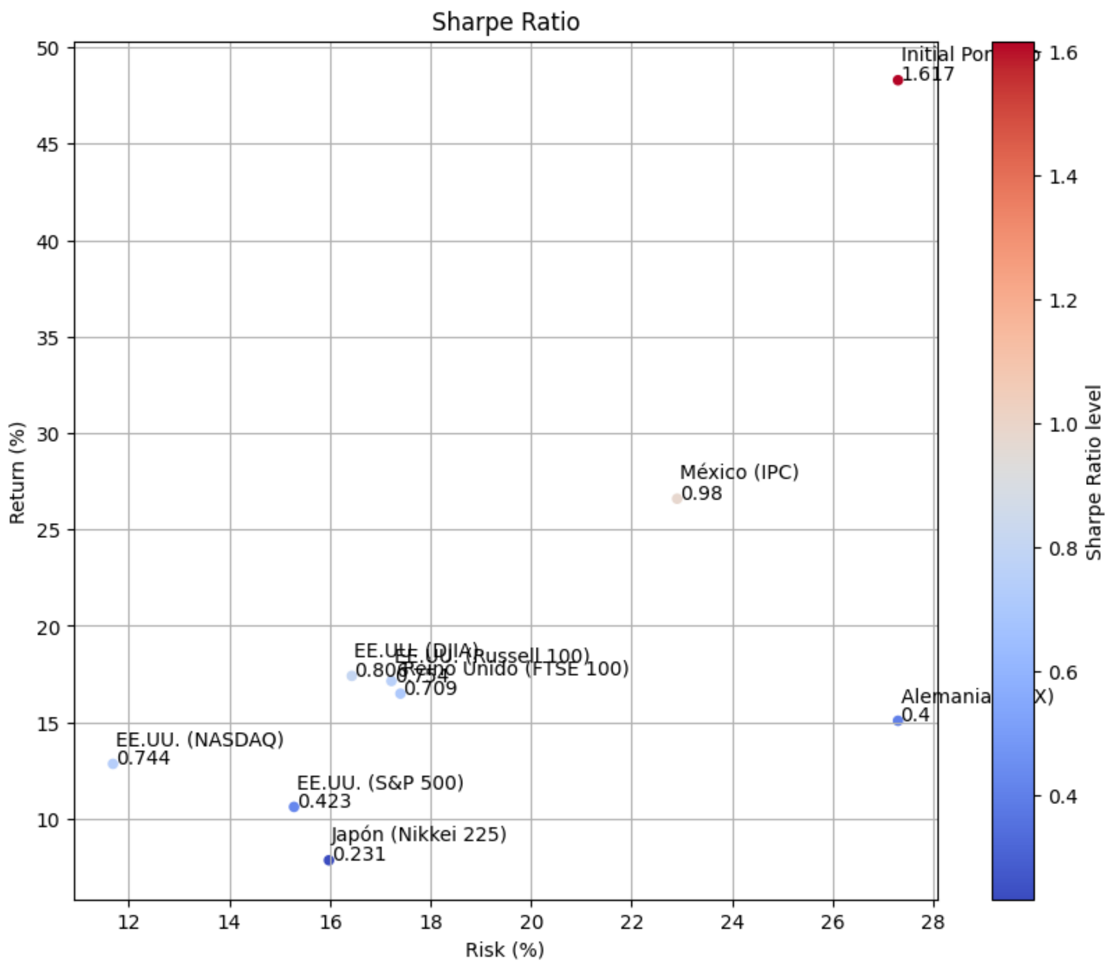
```
[74]: sharpe_ratio = pd.DataFrame()
      sharpe_ratio['Annualized Returns (%)'] = merge_daily_returns.mean()*252*100␣
       ↪#Annualized returns
      sharpe_ratio = pd.concat([sharpe_ratio['Annualized Returns (%)'],␣
       ↪merge_annualized_volatility], axis=1)
      sharpe_ratio['Sharpe Ratio']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

```
[74]:                         Annualized Returns (%)  Annualized Volatility (%)  \
      Initial Portfolio                   48.292658                  27.296752
      México (IPC)                        26.592593                  22.904904
      EE.UU. (DJIA)                       17.405433                  16.435356
      EE.UU. (Russell 100)                17.143433                  17.223190
      EE.UU. (NASDAQ)                     12.847404                  11.685753
      Reino Unido (FTSE 100)              16.486908                  17.402071
      EE.UU. (S&P 500)                    10.614472                  15.286125
      Alemania (DAX)                      15.081337                  27.297079
      Japón (Nikkei 225)                   7.847153                  15.980321

                              Sharpe Ratio
      Initial Portfolio           1.617066
      México (IPC)                0.979729
      EE.UU. (DJIA)               0.806398
      EE.UU. (Russell 100)        0.754299
      EE.UU. (NASDAQ)             0.744103
      Reino Unido (FTSE 100)      0.708818
      EE.UU. (S&P 500)            0.422767
      Alemania (DAX)              0.400385
      Japón (Nikkei 225)          0.231231
```

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

```
[75]: #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('Sharpe Ratio level')

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



9.5 Summary

```
[76]: # 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().strftime("%B %d, %Y")} to: {today.date().
↪strftime("%B %d, %Y")}")

# 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: 669 days. From: March 08, 2024 to: January 06,
2026

Summary: Benchmark Indices + Initial Portfolio

Returns:
'Initial Portfolio' has the largest total return 88.021%
and'Japón (Nikkei 225)' the lowest total return 9.66%

Volatility:
'Alemania (DAX)' has the largest volatility 27.297%
and 'EE.UU. (NASDAQ)' the lowest volatility 11.686%

Sharp Ratio:
'Initial Portfolio' has the best Sharpe Ratio 1.617
and 'Japón (Nikkei 225)' the worst Sharp Ratio 0.231
```

10 Buy and Hold

10.1 B&H - Stocks (Initial Portfolio)

Initial Investment $100,000 in each Asset

[77]:
```
# B&H Stocks in initial Portfolio
functions.buy_and_hold_strategy(cumulative_returns, 100000, "Portfolio Stocks")
```

```
--- Investment Analysis: Portfolio Stocks ---
From: March 11, 2024 to January 05, 2026
Period: 665 days

<pandas.io.formats.style.Styler at 0x27cc1d6ce00>
```

10.2 B&H - Indices and Portfolio

Initial Investment $100,000 in each Index and Initial Portfolio

[78]:
```
# B&H Indices & Portfolio (merged)
functions.buy_and_hold_strategy(merge_cumulative_returns, 100000, "Market␣
↪Benchmarks Indices and Initital Portfolio")
```

```
--- Investment Analysis: Market Benchmarks Indices and Initital Portfolio ---
From: March 11, 2024 to December 30, 2025
Period: 659 days

<pandas.io.formats.style.Styler at 0x27cc1d95880>
```

11 Dollar Cost Averaging (DCA)

11.1 DCA - Stocks (Initial Portfolio)

Monthly Investment $100 in each Asset

[79]:
```
# For individual stocks
functions.dollar_cost_averaging_strategy(prices, monthly_investment=100,␣
↪title_suffix="Individual Stocks")
```

```
--- DCA Investment Analysis: Individual Stocks ---

<pandas.io.formats.style.Styler at 0x27cc1d6e600>
```

11.2 DCA - Indices

Monthly Investment $100 in each Asset

[80]:
```
# For Benchmarks and your Portfolio Index
functions.dollar_cost_averaging_strategy(benchmarks_prices,␣
↪monthly_investment=100, title_suffix="Indices & Portfolio")
```

```
--- DCA Investment Analysis: Indices & Portfolio ---

<pandas.io.formats.style.Styler at 0x27cc1ed4440>
```

12 Momentum

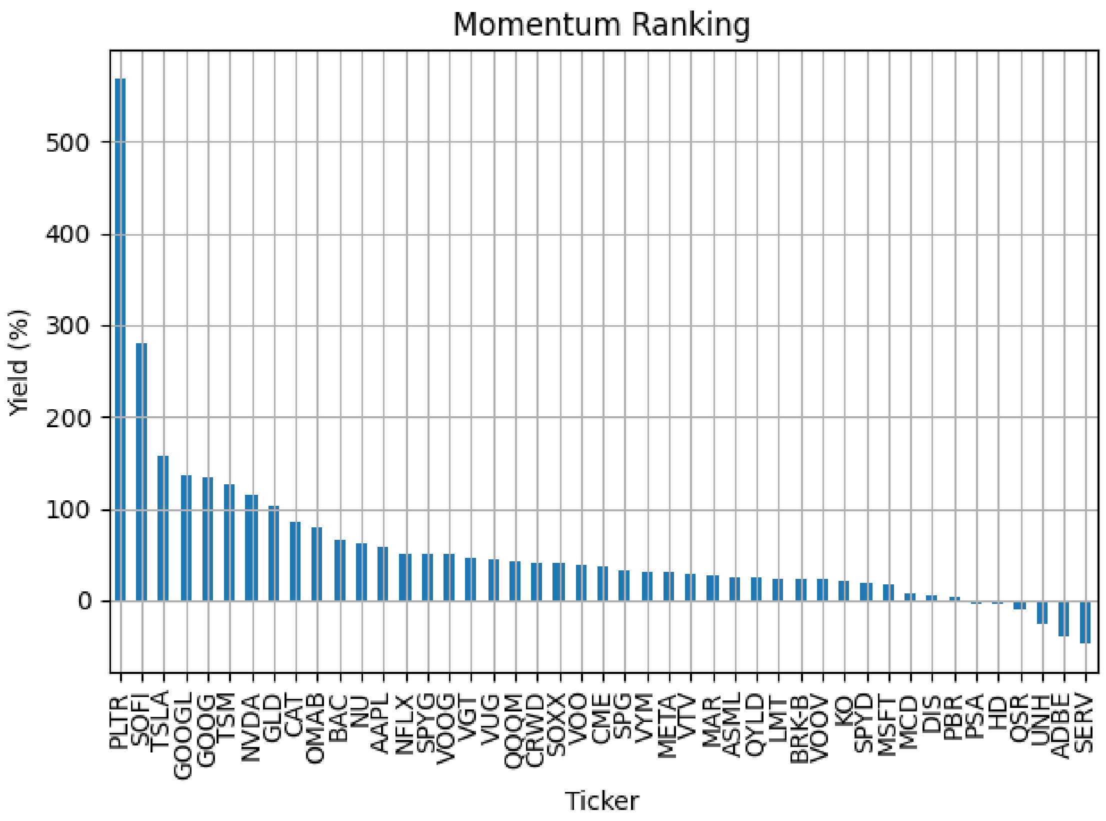
Applied to all **Stocks** in Initial Portfolio

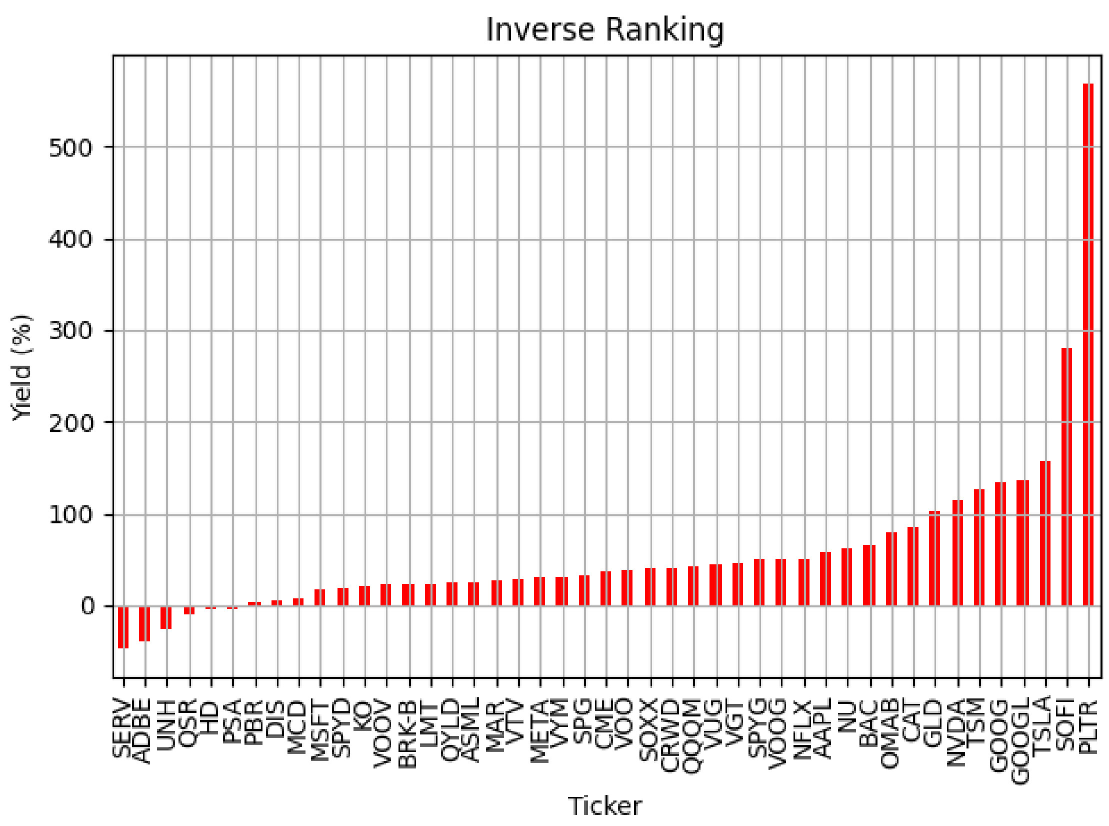
[81]:
```
# call the DataFrame of the Final Cumulative Returns, which is the total return␣
↪of each asset in the period.
cumulative_returns_final

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

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


# 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 (Annual yield):

Ticker
PLTR     568.4
SOFI     279.8
TSLA     157.6
GOOGL    135.5
GOOG     134.6
TSM      126.3
NVDA     115.0
GLD      102.7
CAT       86.5
OMAB      79.5
BAC       66.5
```

```
NU        62.1
AAPL      57.8
NFLX      51.2
SPYG      51.1
VOOG      51.1
VGT       46.8
VUG       45.3
QQQM      42.5
CRWD      41.4
SOXX      41.3
VOO       37.9
CME       37.0
SPG       32.3
VYM       31.2
META      31.0
VTV       28.8
MAR       26.5
ASML      25.5
QYLD      24.3
LMT       23.9
BRK-B     23.7
VOOV      22.8
KO        21.1
SPYD      19.6
MSFT      17.9
MCD        6.8
DIS        5.5
PBR        3.5
PSA       -3.5
HD        -3.8
QSR      -10.2
UNH      -25.6
ADBE     -39.9
SERV     -46.7
Name: Final Cumulative Returns (%), dtype: float64

Inverse Momentum Ranking (Annual yield):

Ticker
SERV     -46.7
ADBE     -39.9
UNH      -25.6
QSR      -10.2
HD        -3.8
PSA       -3.5
PBR        3.5
DIS        5.5
MCD        6.8
MSFT      17.9
```

```
SPYD      19.6
KO        21.1
VOOV      22.8
BRK-B     23.7
LMT       23.9
QYLD      24.3
ASML      25.5
MAR       26.5
VTV       28.8
META      31.0
VYM       31.2
SPG       32.3
CME       37.0
VOO       37.9
SOXX      41.3
CRWD      41.4
QQQM      42.5
VUG       45.3
VGT       46.8
SPYG      51.1
VOOG      51.1
NFLX      51.2
AAPL      57.8
NU        62.1
BAC       66.5
OMAB      79.5
CAT       86.5
GLD      102.7
NVDA     115.0
TSM      126.3
GOOG     134.6
GOOGL    135.5
TSLA     157.6
SOFI     279.8
PLTR     568.4
Name: Final Cumulative Returns (%), dtype: float64
```

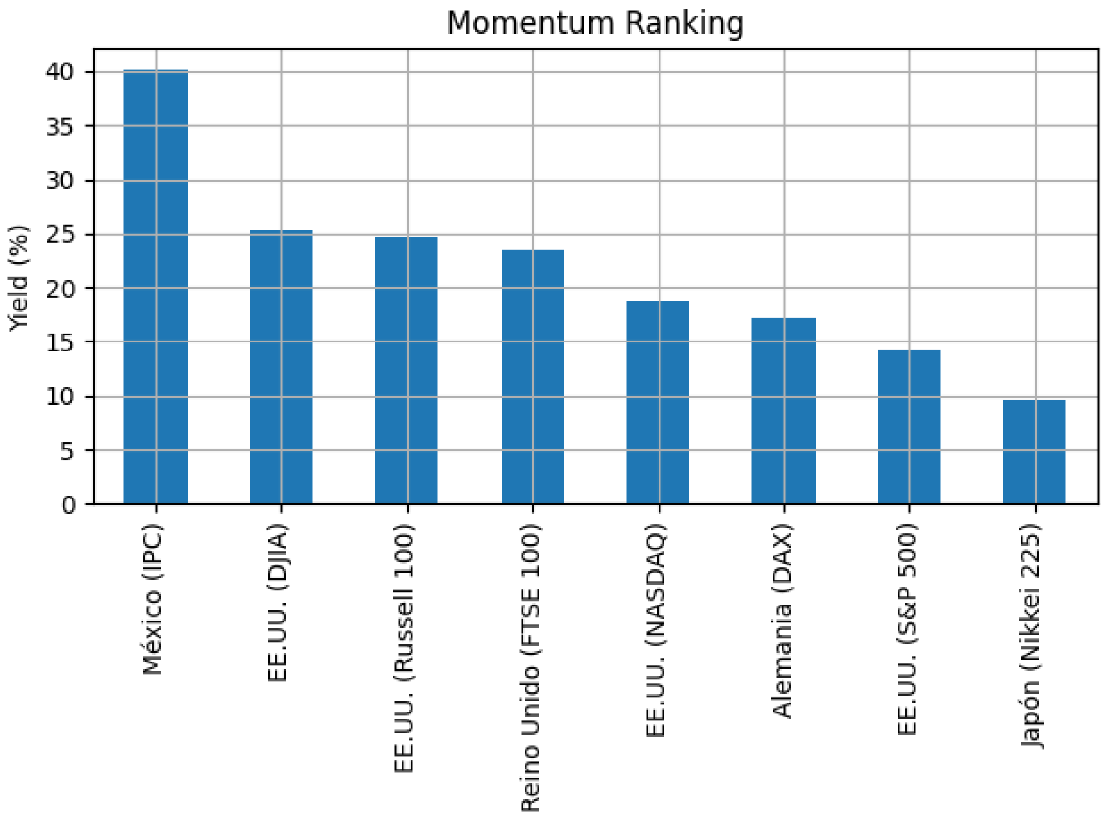

Momentum Ranking
Yield (%)
500
400
300
200
100
0
Ticker
PLTR
SOFI
TSLA
GOOGL
GOOG
TSM
NVDA
GLD
CAT
OMAB
BAC
NU
AAPL
NFLX
SPYG
VOOG
VGT
VUG
QQQM
CRWD
SOXX
VOO
CME
SPG
VYM
META
VTV
MAR
ASML
QYLD
LMT
BRK-B
VOOV
KO
SPYD
MSFT
MCD
DIS
PBR
PSA
HD
OSR
UNH
ADBE
SERV

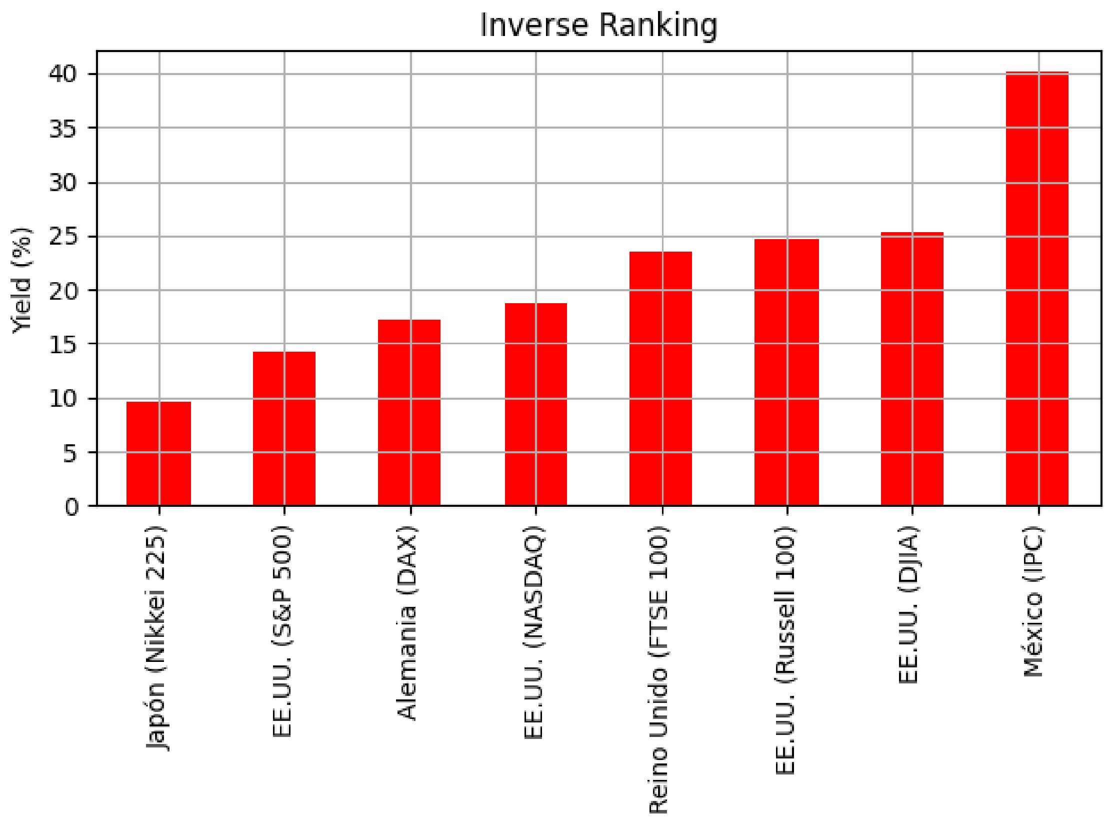


Applied to Indices

```
[82]: # 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:

México (IPC)              40.065119
EE.UU. (DJIA)             25.291417
EE.UU. (Russell 100)      24.619089
Reino Unido (FTSE 100)    23.422122
EE.UU. (NASDAQ)           18.626130
Alemania (DAX)            17.265559
EE.UU. (S&P 500)          14.200866
Japón (Nikkei 225)         9.664525
Name: final cumulative returns Benchmarks (%), dtype: float64

Inverse Momentum Ranking (Annual yield) of Benchmark Indices:

Japón (Nikkei 225)         9.664525
EE.UU. (S&P 500)          14.200866
Alemania (DAX)            17.265559
EE.UU. (NASDAQ)           18.626130
Reino Unido (FTSE 100)    23.422122
EE.UU. (Russell 100)      24.619089
EE.UU. (DJIA)             25.291417
México (IPC)              40.065119
Name: final cumulative returns Benchmarks (%), dtype: float64
```


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



13 Correlation

13.1 Correlation equation

Pearson correlation coefficient:

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

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

13.2 Stocks

[83]:
```
functions.plot_interactive_heatmap_correlation(daily_returns, triangle='half')
functions.plot_extreme_correlations(daily_returns, num_pairs=10)
```

```
================================================================================
REGRESSION & VOLATILITY SUMMARY TABLE
================================================================================
<pandas.io.formats.style.Styler at 0x27cc1f712b0>


--------------------------------------------------
INTERPRETATION GUIDE:
--------------------------------------------------
• p-value Significance: Probability the correlation happened by chance.
  Small values (e.g., < 1.42e-05) mean the relationship is mathematically solid.
• Significance?: 'Yes' if p_value < 0.05.
• Beta (Slope): The magnitude. A slope of 1.2 means for every 1% change in Stock
A,
  Stock B tends to move 1.2%.
• Alpha (Intercept): The 'excess' return of Stock B if Stock A's return was
zero.
• Low Standard Error: Relationship is tight; Beta is a reliable predictor.
• High Standard Error: Lots of 'noise'; Beta is less reliable for hedging/pairs
trading.
• Volatility (Vol):
  - High Vol + High Corr: High-octane comovers.
  - Low Vol + High Corr: Stable 'pairs trading' candidates.
  - Interpretation: If Vol Stock B > Vol Stock A, Beta will naturally be higher.
    Check this to see if Beta is driven by correlation or just swing scale.
```

13.3 Indices & Portfolio

[84]:
```python
functions.plot_interactive_heatmap_correlation(merge_daily_returns,␣
 ↪triangle='half')
functions.plot_extreme_correlations(merge_daily_returns, num_pairs=10)
```

```
================================================================================
REGRESSION & VOLATILITY SUMMARY TABLE
================================================================================

<pandas.io.formats.style.Styler at 0x27cc226c9b0>


--------------------------------------------------
INTERPRETATION GUIDE:
--------------------------------------------------
• p-value Significance: Probability the correlation happened by chance.
  Small values (e.g., < 1.42e-05) mean the relationship is mathematically solid.
• Significance?: 'Yes' if p_value < 0.05.
• Beta (Slope): The magnitude. A slope of 1.2 means for every 1% change in Stock
A,
  Stock B tends to move 1.2%.
• Alpha (Intercept): The 'excess' return of Stock B if Stock A's return was
zero.
• Low Standard Error: Relationship is tight; Beta is a reliable predictor.
• High Standard Error: Lots of 'noise'; Beta is less reliable for hedging/pairs
```

```
trading.
• Volatility (Vol):
  - High Vol + High Corr: High-octane comovers.
  - Low Vol + High Corr: Stable 'pairs trading' candidates.
  - Interpretation: If Vol Stock B > Vol Stock A, Beta will naturally be higher.
    Check this to see if Beta is driven by correlation or just swing scale.
```

14 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:

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

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

14.1 Stocks

[85]:
```
# Plot ANNUALIZED variance/covariance Matrix =  returns.cov() * 252
functions.plot_annualized_covariance_heatmap(daily_returns, triangle='half')
```

14.2 Indices & Portfolio

[86]:
```
# Plot ANNUALIZED variance/covariance Matrix =  returns.cov() * 252
functions.plot_annualized_covariance_heatmap(merge_daily_returns,␣
 ↪triangle='half')
```

15 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

[146]:
```
# --- EFFICIENT FRONTIER & CAPITAL MARKET LINE ANALYSIS---

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

functions.run_full_frontier_analysis(rets=daily_returns,
                          curr_port_weights = weights_df['Weights'],
                          curr_port_vol = portfolio_annualized_volatility,
                          curr_port_ret = portfolio_annualized_return,
                          mean_ann = daily_returns.mean() * 252,
                          cov_ann = daily_returns.cov() * 252,
                          rf_default = risk_free,
                          long_only = True,  # Set to True for standard␣
 ↪long-only constraints
                          portfolio_value = Total_invested, #100000
                          yields = yields/100
                          )
```

```
VBox(children=(HBox(children=(FloatText(value=4.152, description='RF Rate %:'),␣
 ↪FloatText(value=5.0, descripti…
```

15.1 Stocks

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

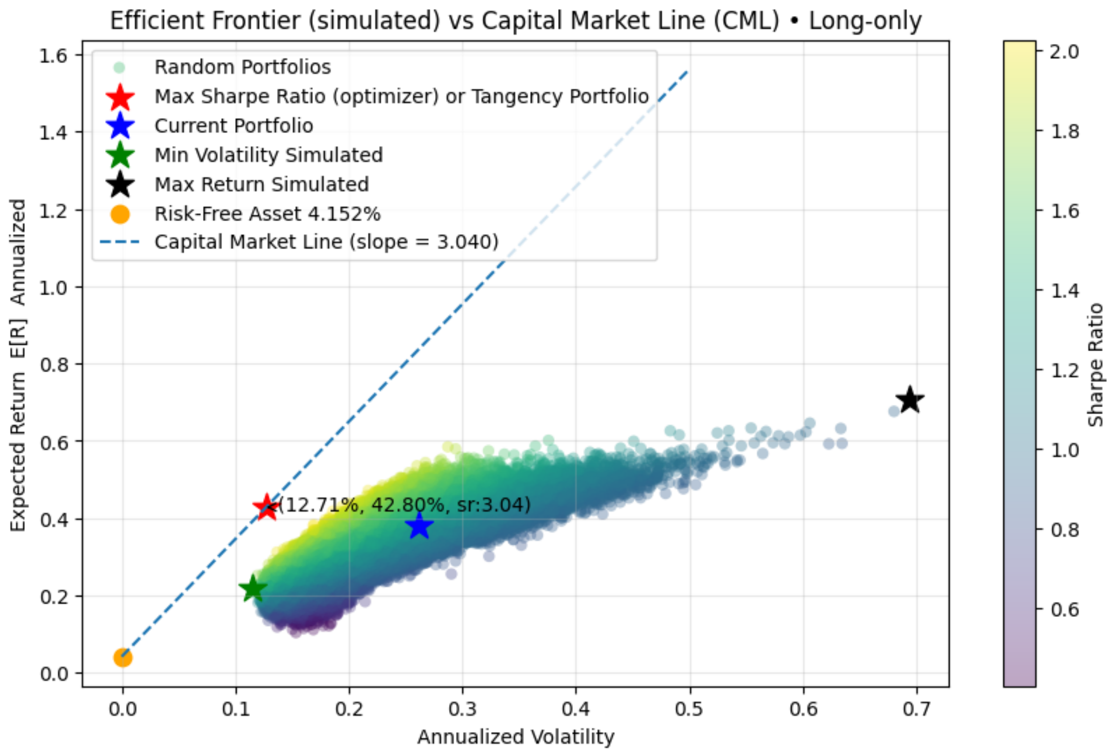
[148]:
```
no_starts = 25
no_simul = 2000000
```

[90]:
```
weights_optimal, vol_ret_sr_optimal = functions.efficient_froentier_sharp_ratio(
                daily_returns,
                [portfolio_annualized_volatility, portfolio_annualized_return],
                daily_returns.mean()*252, # Simple Arithmetic Mean Annualization
                daily_returns.cov()*252, #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
AAPL      0.00
ADBE      0.00
ASML      0.00
BAC       1.45
BRK-B     0.00
CAT       2.55
CME      19.14
CRWD      0.00
DIS       0.00
GLD      39.85
GOOG      0.00
GOOGL    11.69
HD        0.00
KO       12.11
LMT       0.10
MAR       0.00
MCD       0.00
META      0.00
MSFT      0.00
NFLX      0.00
NU        0.00
NVDA      0.00
OMAB      1.50
PBR       0.00
PLTR      8.47
PSA       0.00
QQQM      0.00
QSR       0.00
QYLD      0.00
SERV      0.78
SOFI      1.23
SOXX      0.00
SPG       0.00
SPYD      0.00
SPYG      0.00
TSLA      0.00
TSM       1.13
UNH       0.00
VGT       0.00
VOO       0.00
VOOG      0.00
VOOV      0.00
VTV       0.00
VUG       0.00
VYM       0.00
```

```
Optimum Sharpe Ratio: 3.040
Expected Annual Return: 42.80%
Annual Risk: 12.71%

Simulating random Portfolios…

Simulated Portfolio with Minimum Volatility:
Volatility: 11.44%
Return: 21.65%
Sharpe Ratio: 1.529

Simulated Portfolio with Maximum Return:
Volatility: 69.42%
Return: 70.52%
Sharpe Ratio: 0.956

Current Portfolio:
Volatility: 26.13%
Return: 38.18%
Sharpe Ratio: 1.302
```



15.1.1 Positions from current to optimal

```
[91]: # 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)}")
```

```
        Current QTY  Close Price  Investment  Weights  Optimal Weights  \
Ticker
AAPL         17.040       267.26    4554.121    0.066            0.000
ADBE          0.465       331.56     154.205    0.002            0.000
ASML          0.264      1228.19     324.451    0.005            0.000
BAC           1.469        56.89      83.563    0.001            0.014
BRK-B         0.245       498.52     122.028    0.002            0.000
CAT           0.334       616.10     205.932    0.003            0.026
CME           2.000       275.06     550.120    0.008            0.191
CRWD          1.455       456.55     664.399    0.010            0.000
DIS          13.029       114.07    1486.232    0.022            0.000
GLD           6.449       408.76    2636.236    0.038            0.398
GOOG          1.614       317.32     512.100    0.007            0.000
GOOGL         2.012       316.54     636.790    0.009            0.117
```

```
HD          0.021     344.09      7.210   0.000                0.000
KO          6.227      67.94    423.063   0.006                0.121
LMT         0.319     511.57    163.252   0.002                0.001
MAR         0.457     311.03    142.130   0.002                0.000
MCD         1.054     299.86    315.954   0.005                0.000
META        1.578     658.79   1039.452   0.015                0.000
MSFT       19.752     472.85   9339.751   0.136                0.000
NFLX        1.197      91.46    109.498   0.002                0.000
NU         13.623      17.94    244.388   0.004                0.000
NVDA      100.529     188.12  18911.482   0.275                0.000
OMAB        2.164     109.26    236.470   0.003                0.015
PBR        87.444      11.74   1026.597   0.015                0.000
PLTR       21.024     174.04   3658.947   0.053                0.085
PSA         0.190     260.90     49.624   0.001                0.000
QQQM        6.376     254.43   1622.126   0.024                0.000
QSR         5.255      66.74    350.734   0.005                0.000
QYLD      100.860      17.76   1791.281   0.026                0.000
SERV        3.365      12.68     42.666   0.001                0.008
SOFI        5.320      29.28    155.765   0.002                0.012
SOXX        3.046     318.06    968.755   0.014                0.000
SPG         0.199     183.11     36.404   0.001                0.000
SPYD        3.646      43.69    159.305   0.002                0.000
SPYG        1.019     107.16    109.187   0.002                0.000
TSLA        4.925     451.67   2224.304   0.032                0.000
TSM        10.035     322.25   3233.660   0.047                0.011
UNH         5.037     342.02   1722.644   0.025                0.000
VGT         2.019     757.42   1529.254   0.022                0.000
VOO         2.288     632.46   1447.284   0.021                0.000
VOOG        6.296     446.51   2811.387   0.041                0.000
VOOV       10.365     207.51   2150.748   0.031                0.000
VTV         0.405     194.65     78.817   0.001                0.000
VUG         1.008     488.45    492.502   0.007                0.000
VYM         2.105     145.82    306.923   0.004                0.000

        New Investment usd  New QTY  Position_to_Optimal
Ticker
AAPL                 0.000    0.000              -17.040
ADBE                 0.000    0.000               -0.465
ASML                 0.000    0.000               -0.264
BAC                998.060   17.544               16.075
BRK-B                0.000    0.000               -0.245
CAT               1755.209    2.849                2.515
CME              13174.395   47.896               45.896
CRWD                 0.000    0.000               -1.455
DIS                  0.000    0.000              -13.029
GLD              27429.449   67.104               60.655
GOOG                 0.000    0.000               -1.614
GOOGL             8046.430   25.420               23.408
```

```
HD                    0.000     0.000                   -0.021
KO                 8335.524   122.689                  116.462
LMT                  68.832     0.135                   -0.185
MAR                   0.000     0.000                   -0.457
MCD                   0.000     0.000                   -1.054
META                  0.000     0.000                   -1.578
MSFT                  0.000     0.000                  -19.752
NFLX                  0.000     0.000                   -1.197
NU                    0.000     0.000                  -13.623
NVDA                  0.000     0.000                 -100.529
OMAB               1032.476     9.450                    7.285
PBR                   0.000     0.000                  -87.444
PLTR               5830.048    33.498                   12.475
PSA                   0.000     0.000                   -0.190
QQQM                  0.000     0.000                   -6.376
QSR                   0.000     0.000                   -5.255
QYLD                  0.000     0.000                 -100.860
SERV                536.888    42.341                   38.976
SOFI                846.630    28.915                   23.595
SOXX                  0.000     0.000                   -3.046
SPG                   0.000     0.000                   -0.199
SPYD                  0.000     0.000                   -3.646
SPYG                  0.000     0.000                   -1.019
TSLA                  0.000     0.000                   -4.925
TSM                 777.799     2.414                   -7.621
UNH                   0.000     0.000                   -5.037
VGT                   0.000     0.000                   -2.019
VOO                   0.000     0.000                   -2.288
VOOG                  0.000     0.000                   -6.296
VOOV                  0.000     0.000                  -10.365
VTV                   0.000     0.000                   -0.405
VUG                   0.000     0.000                   -1.008
VYM                   0.000     0.000                   -2.105

Current Investment: $ 68831.74
New Investment: $ 68831.74
```

15.2 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:

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

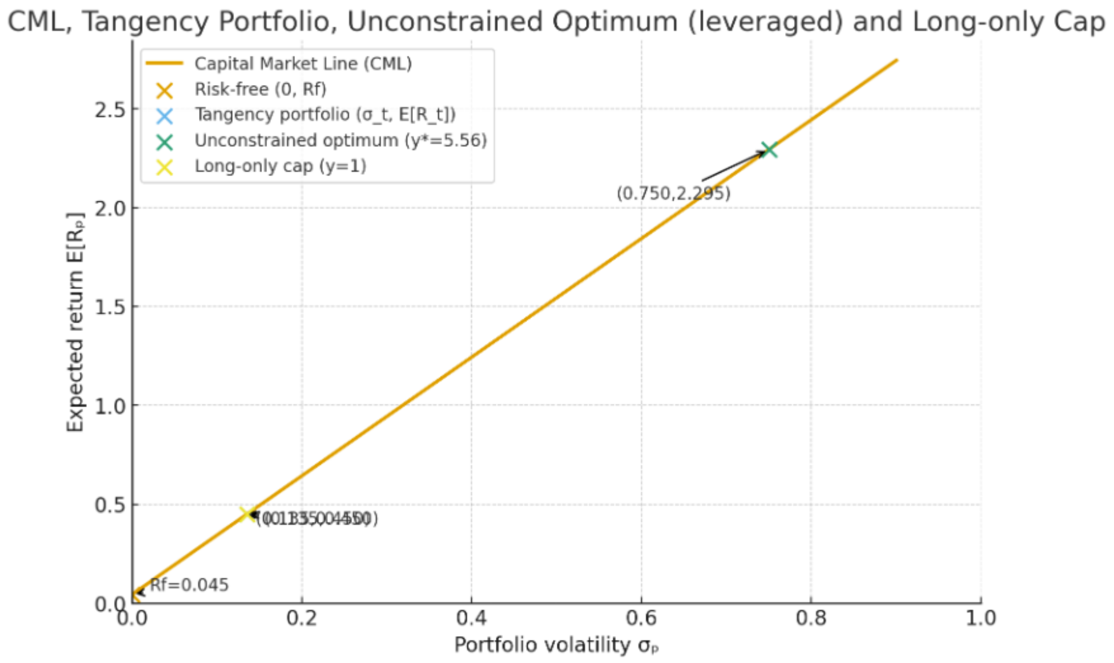
- Objective 3. Mean-Variance Utility (Expected Utility Funcion):

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

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

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.

[92]:
```
# 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: 11.44%
The fraction of total wealth X invested in the tangency (risky) portfolio 'y'
is: 90.00%, that is $61,948.57usd
The fraction invested in the risk-free asset (1-y) is: 10.00%, that is
$6,883.17usd
Expected Return E[Rp] = 38.94%
Sharpe Ratio = 3.04
```

[93]:
```
# 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: 38.52%
The fraction of total wealth X invested in the tangency (risky) portfolio 'y'
is: 88.93%, that is $61,209.13usd
The fraction invested in the risk-free asset (1-y) is: 11.07%, that is
$7,622.61usd
Volatility: 11.31%
Sharpe Ratio = 3.04
```

[94]:
```
# 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}")
```

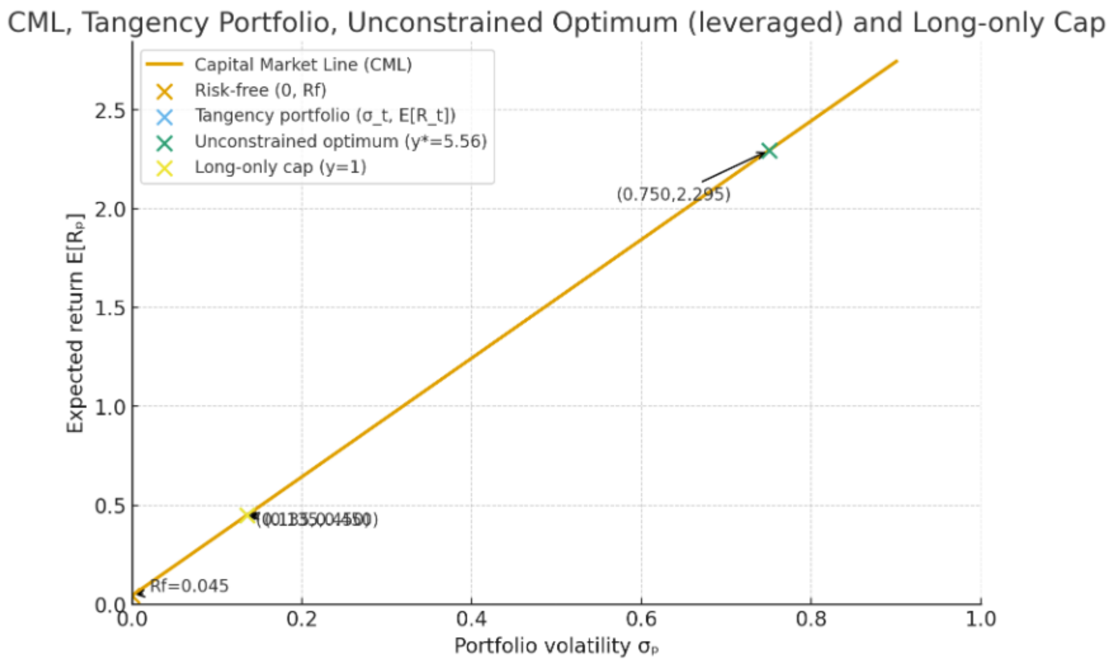
```
a) Allowing borrowing (short in risk asset):
Mean-variance risk aversion coefficient: 4
The fraction of total wealth X invested in the tangency (risky) portfolio y* is:
597.81%, that is $411,484.33usd
The fraction invested in the risk-free asset (1-y*) is: 11.07%, that is
$-342,652.59usd
Expected Return E[Rp] = 235.20%
Volatility sigma_p = 76.00%
Sharpe Ratio = 3.0

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 $68,831.74usd
The fraction invested in the risk-free asset (1-y) is: 0.00%, that is $0.00usd
Expected Return E[Rp] = 42.80%
Volatility sigma_p = 12.71%
Sharpe Ratio = 3.0
```

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.

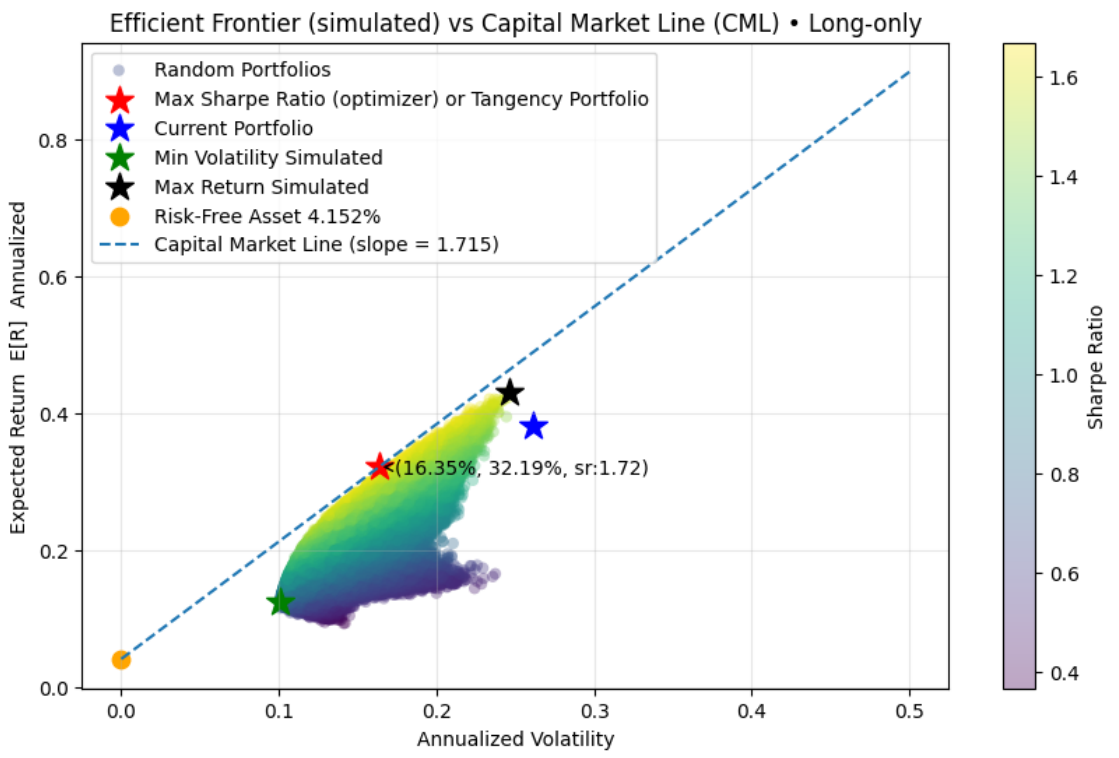
The tangency portfolio has an extremely high Sharpe (3.0). With moderate risk aversion =4, the utility-maximizing solution is to lever the tangency portfolio heavily (556% of wealth) because the reward-to-risk tradeoff is so favorable.



15.3 Indices & Portfolio

[95]:
```
weights_optimal, vol_ret_sr_optimal = functions.efficient_froentier_sharp_ratio(
                merge_daily_returns,
                [portfolio_annualized_volatility, portfolio_annualized_return],
                merge_daily_returns.mean()*252, # Simple Arithmetic Mean␣
↪Annualization
                merge_daily_returns.cov()*252, #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
EE.UU. (S&P 500)           0.00
EE.UU. (NASDAQ)           37.73
EE.UU. (DJIA)              7.24
EE.UU. (Russell 100)       0.00
México (IPC)               0.00
Japón (Nikkei 225)         0.00
Alemania (DAX)             1.48
Reino Unido (FTSE 100)     0.00
Initial Portfolio         53.55

Optimum Sharpe Ratio: 1.715
Expected Annual Return: 32.19%
Annual Risk: 16.35%

Simulating random Portfolios…

Simulated Portfolio with Minimum Volatility:
Volatility: 10.05%
Return: 12.42%
Sharpe Ratio: 0.823

Simulated Portfolio with Maximum Return:
Volatility: 24.61%
Return: 43.19%
Sharpe Ratio: 1.586

Current Portfolio:
Volatility: 26.13%
Return: 38.18%
Sharpe Ratio: 1.302
```



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

(see Frontera Eficiente for GUI)

16 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).
µ (Retorno anual)	> 8% estable, > 15% agresivo	Rendimiento esperado anualizado del portafolio.

Métrica	Valor Óptimo / Bueno	Interpretación
(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).

```
[ ]: #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'}
```

```
[ ]: # 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)
```

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

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

18 CAPM (Capital Asset Pricing Model)

CAPM Equation:

$$r_i = r_f + \beta_{im} * (r_m - r_f)$$

$\boldsymbol{where}:$
$r_i = $ Expected Asset Return

$r_f = $ Risk-free asset Return
$ _{i,m} = $ Asset Beta w.r.t market
$r_m = $ Market Return

Risk-free is the minimum Return an Investor can accept.
Difference between r_m and r_f is the Premium that the investor recieves by taking the risk (**equity risk Premium**).
β measures the cuantity of Risk of an asset with respect to the Market.

18.0.1 Asset Beta (β)

$$\beta = \frac{\text{Cov}(r_A, r_m)}{Var(r_m)} = \frac{\sigma_{A,m}}{\sigma_m^2} = \frac{\rho_{A,m}\sigma_m\sigma_A}{\sigma_m^2} = \frac{\rho_{A,m}\sigma_A}{\sigma_m}$$

19 Candles