Carlitos

January 15, 2026

1 Libraries

[24]:
```
#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', 'pdfkit', 'wkhtmltopdf'] #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
```

2 Dates

[25]:
```
#Dates
global start_date
start_date = datetime(2021, 4, 25) # <- date in which brokerage account started
# start_date = datetime(2025, 1, 1) # <- 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}")
```

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.

[26]:
```
risk_free = 0.0330 #0.04152 10-year treasury

# Open list of tickers from original Robinhood's file_df_df
file_name = "Carlitos.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}")
```

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")
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

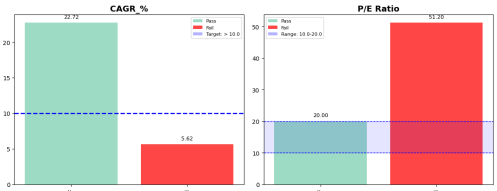
[27]:
```
## .info() Fundamental Analysis for all Tickers based on Financial Ratios

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

```
                              Name             Sector  \
Ticker
GMBXF   Grupo México, S.A.B. de C.V.    Basic Materials
MELI              MercadoLibre, Inc.  Consumer Cyclical

                              Industry  CAGR_%  P/E Ratio  P/B Ratio  \
Ticker
GMBXF   Other Industrial Metals & Mining   22.72  20.000000   3.769705
MELI                     Internet Retail    5.62  51.203953  17.036259

        ROIC_%    D/E_%   EPS_usd  ROE Ratio  …  Gross Margin_ratio  \
Ticker                                        …
GMBXF    20.53   39.642   0.59000    0.20863  …             0.55273
MELI     23.83  159.296  59.70074    0.40646  …             0.50357

        Net Margin_%  Current Ratio  Overall Risk   Beta  EBITDA_usd  \
Ticker
GMBXF          22.36          5.920           0.0  1.161  8641267712
MELI            9.20          1.174           8.0  1.421  3864000000

        EBITDA Margins Ratio  Earning Growth Ratio  Revenue Growth Ratio  \
Ticker
GMBXF                0.51220                 0.545                 0.113
MELI                 0.14752                 0.061                 0.395
```

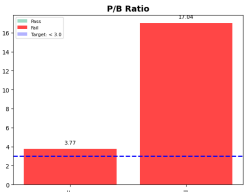
```
        Operating Margins Ratio
Ticker
GMBXF                   0.46904
MELI                    0.09772

[2 rows x 21 columns]
```

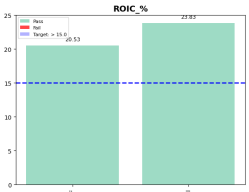
[28]:
```
ratio_criteria= {
    'CAGR_%': (">", 10.0), # %
    'P/E Ratio': ("between", (10.0, 20.0)), # ratio
    'P/B Ratio': ("<", 3.0), # ratio
    'ROIC_%': (">", 15.0), # %
    'D/E_%': ("<", 100.0), # %
    'EPS_usd': (">", 10.0), # USD. # 10% of asset's CAGR. Will automatically␣
 ↪trigger 10% 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
}
```

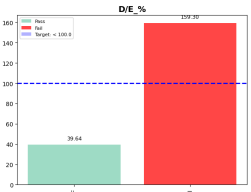
[29]:
```
# Plot financial Ratios
functions.plot_ratios(fa_df, ratio_criteria, plots_per_row = 2)
```

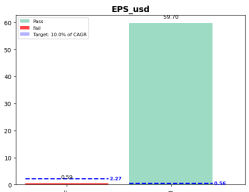

CAGR_%
Pass
Fail
Target: > 10.0
22.72
5.62
GMBXF
MELI

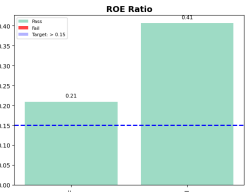
P/E Ratio
Pass
Fail
Range: 10.0-20.0
20.00
51.20
GMBXF
MELI

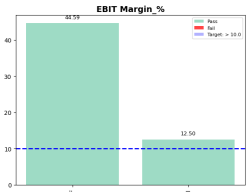

P/B Ratio
Pass
Fail
Target: < 3.0
3.77
17.04
GMBXF
MELI

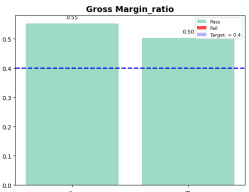

ROIC_%
Pass
Fail
Target: > 15.0
20.53
23.82
GMBXF
MELI

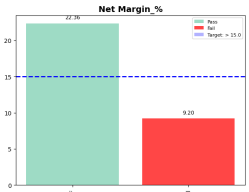

D/E_%
Pass
Fail
Target: < 100.0
39.64
159.30
GMBXF
MELI

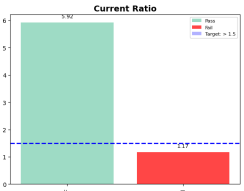

EPS_usd
Pass
Fail
Target: 10.0% of CAGR
0.58
2.27
59.70
0.56
GMBXF
MELI

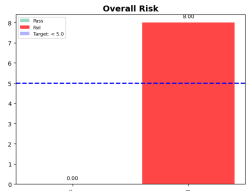

ROE Ratio
Pass
Fail
Target: > 0.15
0.21
0.41
GMBXF
MELI

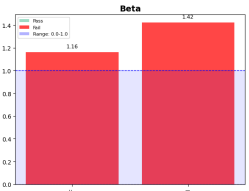

EBIT Margin_%
Pass
Fail
Target: > 10.0
44.59
12.50
GMBXF
MELI

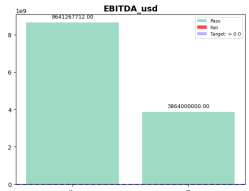

Gross Margin_ratio
Pass
Fail
Target: > 0.4
0.55
0.50
GMBXF
MELI

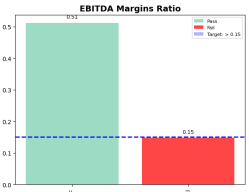

Net Margin_%
Pass
Fail
Target: > 15.0
22.36
9.20
GMBXF
MELI

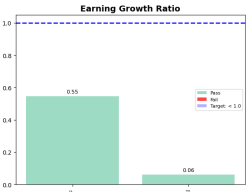

Current Ratio
Pass
Fail
Target: > 1.5
5.92
1.17
GMBXF
MELI

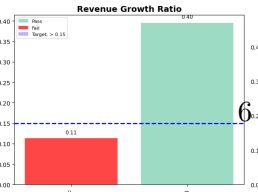

Overall Risk
Pass
Fail
Target: < 5.0
0.00
8.00
GMBXF
MELI

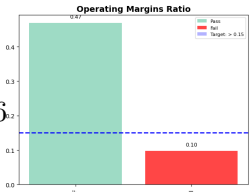

Beta
Pass
Fail
Range: 0.8-1.0
1.16
1.42
GMBXF
MELI


EBITDA_usd
1e9
Pass
Fail
Target: > 0.0
8641287712.00
3864000000.00
GMBXF
MELI


EBITDA Margins Ratio
Pass
Fail
Target: > 0.15
0.51
0.15
GMBXF
MELI


Earning Growth Ratio
Pass
Fail
Target: < 1.0
0.55
0.06
GMBXF
MELI

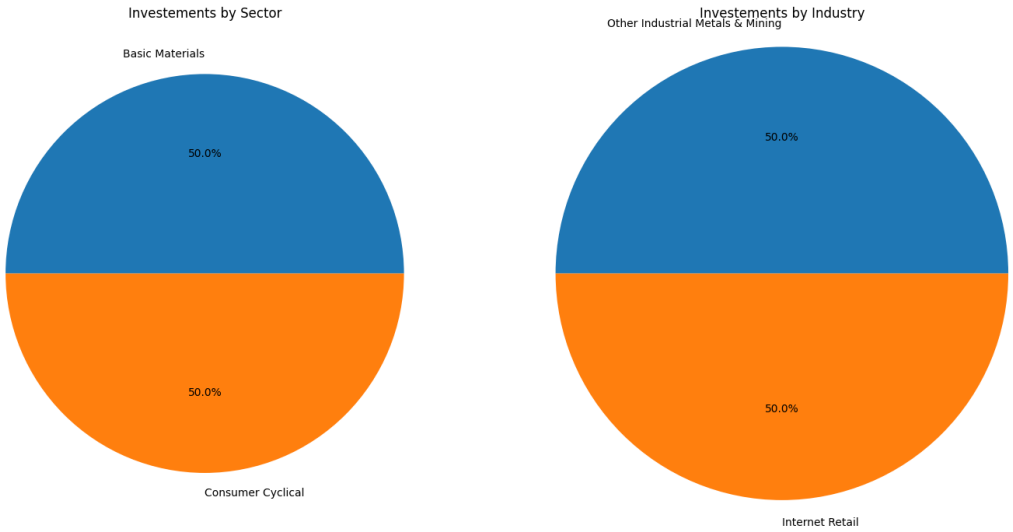

Revenue Growth Ratio
Pass
Fail
Target: > 0.15
0.11
0.40
GMBXF
MELI


Operating Margins Ratio
Pass
Fail
Target: > 0.15
0.47
0.10
GMBXF
MELI

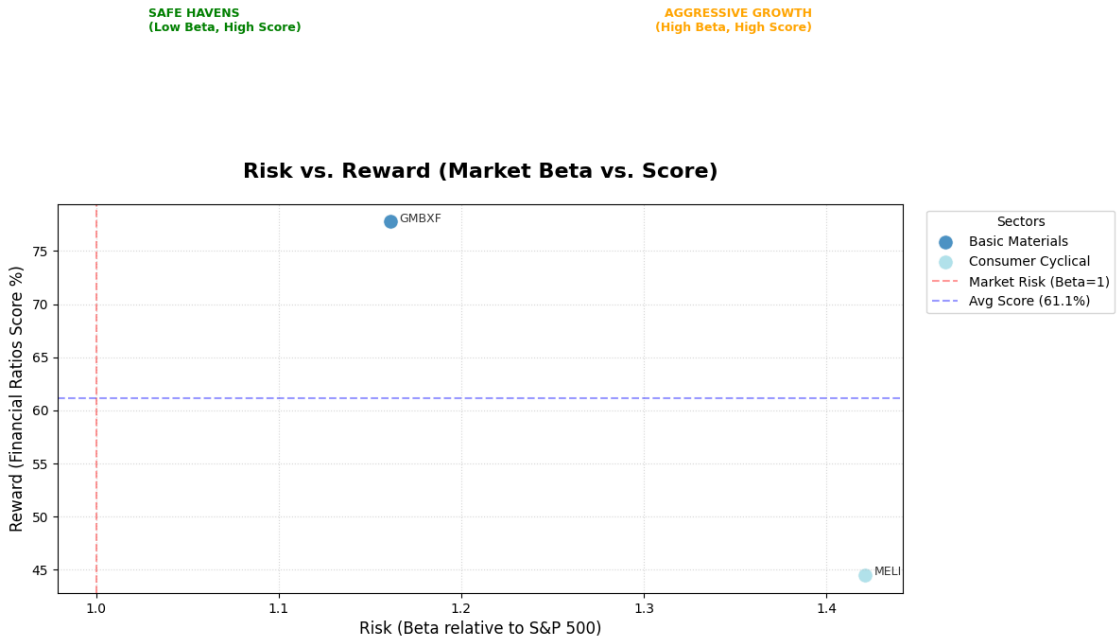
[30]:
```
# Generate the data
scorecard = functions.generate_scorecard(fa_df, ratio_criteria)

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

```
<pandas.io.formats.style.Styler at 0x21cead49940>
```

[31]:
```
# Sectors and Industries
functions.print_sector_industry(fa_df)
```



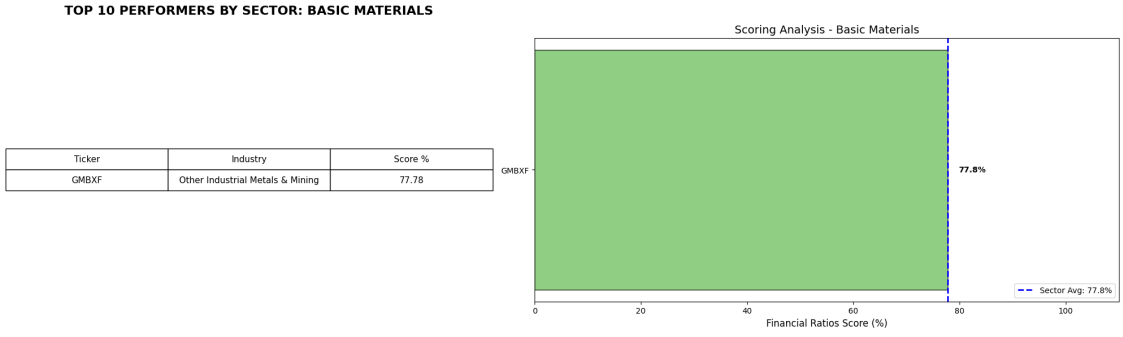
[32]:
```
# --- SCORE vs BETA by Sector ---
functions.plot_risk_reward(fa_df, ratio_criteria)
```



[33]:
```
# --- 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'))
```

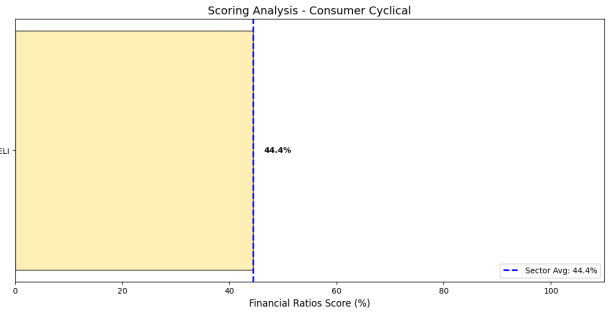
```
<pandas.io.formats.style.Styler at 0x21ceb18a120>
```

[34]:
```
# TOP N BY SECTOR AND INDUSTRY

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



TOP 10 PERFORMERS BY SECTOR: CONSUMER CYCLICAL

Ticker	Industry	Score %
MELI	Internet Retail	44.44



[35]:
```
# 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'))
```

```
<pandas.io.formats.style.Styler at 0x21ceb3e6690>
```

[36]:
```
#  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"))
```

```
<pandas.io.formats.style.Styler at 0x21ce73092e0>
```

```
<pandas.io.formats.style.Styler at 0x21cea3e4ce0>
```

```
<pandas.io.formats.style.Styler at 0x21ce7180230>
```

5 Fundamental Analysis ONE Ticker

TS302_Stock Full Analysis.ipynb

This section is to see in more detail any particular ticker

[37]:
```
# 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=('MELI', 'GMBXF'), value='MELI')
```

```
Output()
```

[38]:
```
# Choose any ticket from the list above
ticker = yf.Ticker(ticker_widget)
ticker_name = ticker.info.get('symbol')
print(ticker_name)
```

History

[39]:
```
# Example of history() of any ONE ticker for "1y"
hist = ticker.history(period="1y", auto_adjust=True)
print(ticker_name)
display(hist)
```

```
                                 Open          High           Low         Close  \
```

```
Date
2025-01-16 00:00:00-05:00  1841.380005  1851.390015  1815.060059  1837.180054
2025-01-17 00:00:00-05:00  1876.420044  1876.420044  1832.910034  1836.000000
2025-01-21 00:00:00-05:00  1852.660034  1863.310059  1829.119995  1834.510010
2025-01-22 00:00:00-05:00  1847.959961  1885.000000  1808.020020  1818.489990
2025-01-23 00:00:00-05:00  1818.000000  1834.479980  1793.000000  1828.729980
…                                  …            …            …            …
2026-01-09 00:00:00-05:00  2191.500000  2193.060059  2162.000000  2178.409912
2026-01-12 00:00:00-05:00  2164.270020  2165.030029  2122.570068  2149.899902
2026-01-13 00:00:00-05:00  2127.209961  2130.149902  2030.910034  2073.570068
2026-01-14 00:00:00-05:00  2055.699951  2108.000000  2041.020020  2101.949951
2026-01-15 00:00:00-05:00  2137.120117  2151.379883  2084.614990  2098.850098

                           Volume  Dividends  Stock Splits
Date
2025-01-16 00:00:00-05:00  242000        0.0           0.0
2025-01-17 00:00:00-05:00  245100        0.0           0.0
2025-01-21 00:00:00-05:00  262800        0.0           0.0
2025-01-22 00:00:00-05:00  402800        0.0           0.0
2025-01-23 00:00:00-05:00  267800        0.0           0.0
…                             …          …             …
2026-01-09 00:00:00-05:00  309200        0.0           0.0
2026-01-12 00:00:00-05:00  408500        0.0           0.0
2026-01-13 00:00:00-05:00  689900        0.0           0.0
2026-01-14 00:00:00-05:00  500100        0.0           0.0
2026-01-15 00:00:00-05:00  423459        0.0           0.0

[251 rows x 7 columns]
```

- Info
- Income Statement
- Balance Sheet

[40]:
```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
```

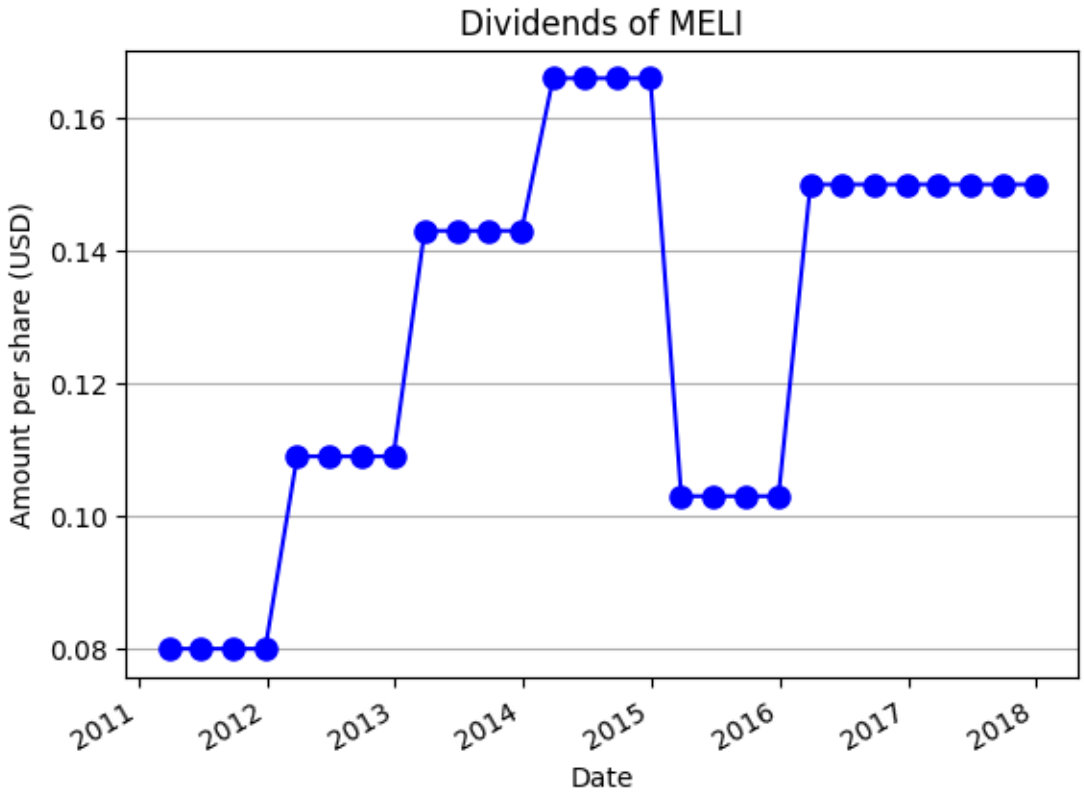
[41]:
```
# Print info by category (some selected data only)

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
```

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

Dividends, Splits and Recommendations

[42]:
```
# 7. Other info:
print("\n7. Other info:")
other_info = {'Dividends' : ticker.dividends,
              'Splits' : ticker.splits,
```

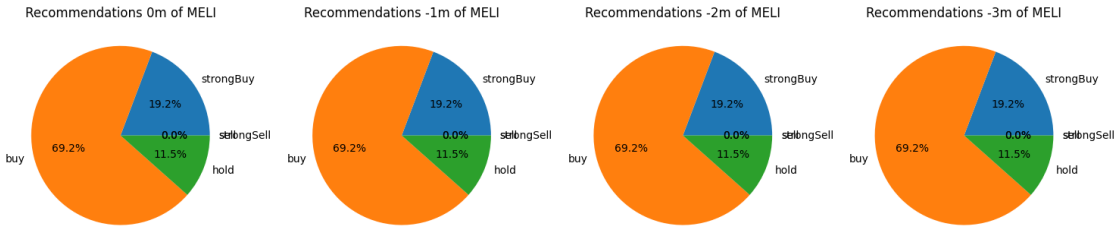
```
              'Recommendations' : ticker.recommendations
            #   'Recommendations Summary' : ticker.recommendations_summary
              }

functions.print_dividends_splits_recommendations(other_info, ticker_name)
```




```
Date
2011-03-29 00:00:00-04:00    0.080
2011-06-28 00:00:00-04:00    0.080
2011-09-28 00:00:00-04:00    0.080
2011-12-28 00:00:00-05:00    0.080
2012-03-28 00:00:00-04:00    0.109
2012-06-27 00:00:00-04:00    0.109
2012-09-26 00:00:00-04:00    0.109
2012-12-27 00:00:00-05:00    0.109
2013-03-26 00:00:00-04:00    0.143
2013-06-26 00:00:00-04:00    0.143
2013-09-26 00:00:00-04:00    0.143
2013-12-27 00:00:00-05:00    0.143
2014-03-27 00:00:00-04:00    0.166
2014-06-26 00:00:00-04:00    0.166
2014-09-26 00:00:00-04:00    0.166
```

```
2014-12-29 00:00:00-05:00    0.166
2015-03-27 00:00:00-04:00    0.103
2015-06-26 00:00:00-04:00    0.103
2015-09-28 00:00:00-04:00    0.103
2015-12-29 00:00:00-05:00    0.103
2016-03-29 00:00:00-04:00    0.150
2016-06-28 00:00:00-04:00    0.150
2016-09-28 00:00:00-04:00    0.150
2016-12-28 00:00:00-05:00    0.150
2017-03-29 00:00:00-04:00    0.150
2017-06-28 00:00:00-04:00    0.150
2017-09-28 00:00:00-04:00    0.150
2017-12-28 00:00:00-05:00    0.150
Name: Dividends, dtype: float64
```

	period	strongBuy	buy	hold	sell	strongSell
0	0m	5	18	3	0	0
1	-1m	5	18	3	0	0
2	-2m	5	18	3	0	0
3	-3m	5	18	3	0	0



[43]:
```python
import operator
from dataclasses import dataclass
from typing import Callable, Any, Optional

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

@dataclass
class FinancialRule:
    retrieval_func: RetrievalFunc   # function used to extract the specific␣
 ↪metric from the data source
    metric_name: str
    target_value: Optional[Any] = None
    comparison_func: Optional[Callable[[Any, Any], bool]] = None
          # for printing pruposes
```

```
    # target_value: float              # The target value to compare against
    # comparison_func: CompareFunc     # comparison operator function (e.g.,␣
 ↪operator.lt for <)


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

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

```
    'Net Margin_%':              FinancialRule(retrieval_func=lambda ticker_info:
 ↪ functions.get_net_margin(ticker_info.get('symbol')),   target_value=15.0,␣
 ↪comparison_func=operator.ge, metric_name='Net Margin_%'),
    'Current Ratio':             FinancialRule(retrieval_func=lambda ticker_info:
 ↪ ticker_info.get('currentRatio', 'N/A'),        target_value=1.5,␣
 ↪comparison_func=operator.ge, metric_name='Current Ratio'),
    'Overall Risk':              FinancialRule(retrieval_func=lambda ticker_info:
 ↪ ticker_info.get('overallRisk', 'N/A'),         target_value=5.0,␣
 ↪comparison_func=operator.lt, metric_name='Overall Risk'),
    'Beta':                      FinancialRule(retrieval_func=lambda ticker_info:
 ↪ ticker_info.get('beta', 'N/A'),                target_value=1.0,␣
 ↪comparison_func=operator.eq, metric_name='Beta'),
    'EBITDA_usd':                FinancialRule(retrieval_func=lambda ticker_info:
 ↪ ticker_info.get('ebitda', 'N/A'),              target_value=0.0,␣
 ↪comparison_func=operator.gt, metric_name='EBITDA_usd'),
    'EBITDA Margins Ratio':      FinancialRule(retrieval_func=lambda ticker_info:
 ↪ ticker_info.get('ebitdaMargins', 'N/A'),       target_value=0.15,␣
 ↪comparison_func=operator.gt, metric_name='EBITDA Margins Ratio'),
    'Earning Growth Ratio':      FinancialRule(retrieval_func=lambda ticker_info:
 ↪ ticker_info.get('earningsGrowth', 'N/A'),      target_value=0.15,␣
 ↪comparison_func=operator.gt, metric_name='Earning Growth Ratio'),
    'Revenue Growth Ratio':      FinancialRule(retrieval_func=lambda ticker_info:
 ↪ ticker_info.get('revenueGrowth', 'N/A'),       target_value=0.15,␣
 ↪comparison_func=operator.gt, metric_name='Revenue Growth Ratio'),
    'Operating Margins Ratio':   FinancialRule(retrieval_func=lambda ticker_info:
 ↪ ticker_info.get('operatingMargins', 'N/A'),    target_value=0.15,␣
 ↪comparison_func=operator.gt, metric_name='Operating Margins Ratio'),
}

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

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

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

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


        if rule.comparison_func is not None:
```

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

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

            if is_pass:
                print(f"Status:  PASS ({actual_value} is acceptable)")
            else:
                print(f"Status:  FAIL ({actual_value} fails rule to be {rule.
↪comparison_func.__name__} {rule.target_value})")

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

evaluate_ticker_rules(financial_rules, ticker_info)
```

6 All Tickers in Portfolio

6.1 Import Prices from yfinance (All Tickers)

TS302_Stock Full Analysis.ipynb

[44]:
```
# 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()
```

```
[**********************100%**********************]  2 of 2 completed
```

6.2 Prices (Close)

[45]:
```
prices_raw = df["Close"]
print('-- Last 5 days of prices --')
display(prices_raw.tail(5))
```

```
Ticker      GMBXF         MELI
Date
2026-01-08   9.78  2179.800049
2026-01-09  10.22  2178.409912
2026-01-12  10.63  2149.899902
2026-01-13  10.75  2073.570068
2026-01-14  11.05  2101.949951
```

[46]:
```
# drop NaN (days without price)
prices = prices_raw.dropna()
display(prices.tail(5))
```

```
Ticker      GMBXF         MELI
Date
2026-01-08   9.78  2179.800049
2026-01-09  10.22  2178.409912
2026-01-12  10.63  2149.899902
2026-01-13  10.75  2073.570068
2026-01-14  11.05  2101.949951
```

[47]:
```
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).")
```

[48]:
```
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)
```

```
Ticker
GMBXF   2021-04-26
MELI    2021-04-26
Name: First Valid Date, dtype: datetime64[ns]
```

[49]:
```
# -- PLOT ASSET PRICES --
functions.plot_prices(prices=prices, yaxis_label="Price (USD)")
```

6.3 Stats (describe)

[50]:
```
# statistics about prices
prices.describe()
```

[50]:
Ticker	GMBXF	MELI
count	1187.000000	1187.000000
mean	4.817137	1545.265932
std	1.372546	488.124554
min	2.786575	612.700012
25%	3.868857	1180.909973
50%	4.551509	1529.160034
75%	5.226378	1951.325012
max	11.050000	2613.629883

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.

[51]:
```
# 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:

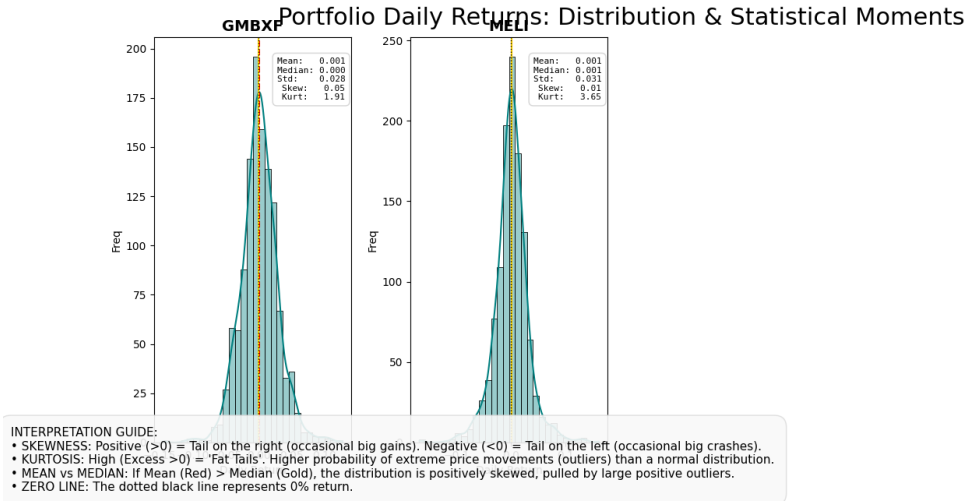
$$Return(R_t) = \frac{P_t - P_{t-1}}{P_{t-1}} = \frac{P_t}{P_{t-1}} - 1$$

[52]:
```
# 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)
```

[52]:
Ticker	GMBXF	MELI
Date		

```
2026-01-08 -0.012121  0.007949
2026-01-09  0.044990 -0.000638
2026-01-12  0.040117 -0.013088
2026-01-13  0.011289 -0.035504
2026-01-14  0.027907  0.013686
```

[53]:
```
results_df = functions.plot_returns_distributions(daily_returns, num_cols=4)
```




```
<pandas.io.formats.style.Styler at 0x21cec843500>
```

[54]:
```
# -- PLOT DAILY RETURNS --
functions.plot_daily_returns(daily_returns)
```

6.6 Summary of Daily Returns

[55]:
```
# 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}%")
```

6.7 Annualized Returns

[56]:
```
# 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}")
```

6.8 Cumulative Returns in Period

TS303_yfinance Indices Bursatiles.ipynb

[57]:
```
# 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)
```

[57]:
```
Ticker           GMBXF       MELI
Date
2026-01-08  133.066848  34.306014
2026-01-09  143.552486  34.220362
2026-01-12  153.323179  32.463749
2026-01-13  156.182892  27.760769
2026-01-14  163.332187  29.509365
```

[58]:
```
# -- PLOT CUMULATIVE RETURNS --
functions.plot_cumulative_returns(cumulative_returns)
```

6.9 Final Cumulated Returns in Period

[59]:
```
# 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)
```

```
<bound method Series.all of Ticker
GMBXF    163.3
MELI      29.5
Name: Final Cumulative Returns (%), dtype: float64>
```

6.10 Summary Final Cumulated Returns

[60]:
```
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")
```

6.11 Risk, Annualized Volatility

TS303_yfinance Indices Bursatiles.ipynb

Asset Volatility (risk or std)

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

[61]:
```
# 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)
```

```
        Volatility (%)
Ticker
MELI             49.96
GMBXF            44.91
```

Stats of Volatility

[62]:
```
display(volatility_df.describe()) # all stocks
```

```
       Volatility (%)
count        2.000000
mean        47.435000
std          3.570889
min         44.910000
25%         46.172500
50%         47.435000
75%         48.697500
max         49.960000
```

6.12 Summary Risk, Volatility

[63]:
```
# 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} %")
```

6.13 Dividend Yields (%)

[64]:
```
# ANNUAL DIVIDEND YIELDS
yields = functions.plot_annual_dividnd_yields(daily_returns)
```

```
GMBXF    2.97
MELI     0.00
Name: dividendYield, dtype: float64
```

7 Initial Portfolio (Original)

7.1 Assets Weights

Compute Assets weights from the original portfolio

[65]:
```
# 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))
```

```
        Current QTY  Close Price  Investment  Weights
Ticker
MELI            100      2101.95    210195.0     0.99
GMBXF           100        11.05      1105.0     0.01
```

TO-DO: Grafica de Pastel con los Porcentajes

Grafica por Sectores, Industrias

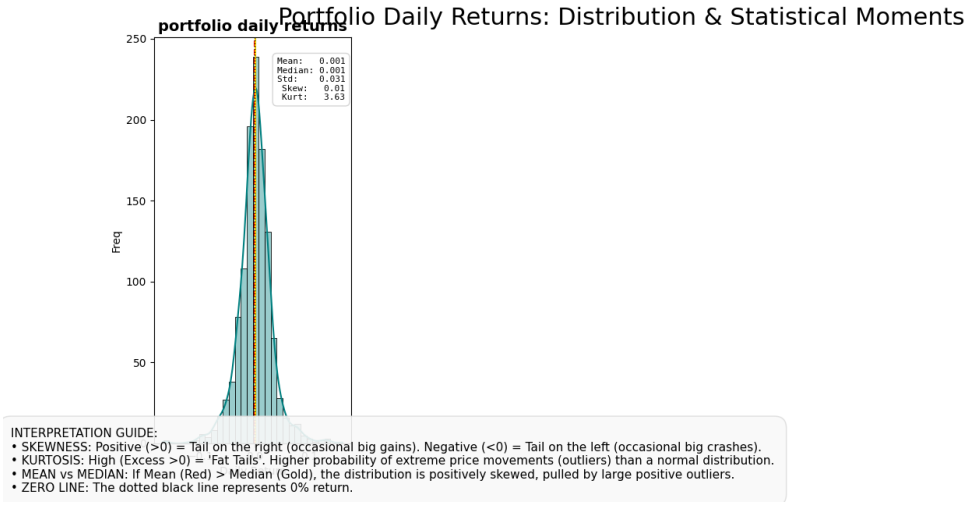
7.2 Portfolio Daily Returns

[66]:
```
#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]")
```

[67]:
```
# 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)
```

```
Date
2021-04-27   -0.011096
2021-04-28    0.010351
2021-04-29   -0.020933
2021-04-30   -0.011081
2021-05-03    0.000395
                 …
2026-01-08    0.007844
2026-01-09   -0.000399
2026-01-12   -0.012809
2026-01-13   -0.035259
2026-01-14    0.013761
Name: portfolio daily returns, Length: 1186, dtype: float64
```

[68]:
```
results_df = functions.plot_returns_distributions(pd.
 ↪DataFrame(portfolio_daily_returns), num_cols=4)
```



<pandas.io.formats.style.Styler at 0x21cec710500>

[69]:
```
# -- PLOT DAILY RETURNS --
functions.plot_daily_returns(pd.DataFrame(portfolio_daily_returns))
```

7.3 Portfolio Annualized Returns

[70]:
```
# Annualized Returns
annualized_return_portfolio = portfolio_daily_returns.mean() * 250
print("Annualized Returns (%):")
print(f"{round(annualized_return_portfolio, 2) * 100}%")
```

7.4 Portfolio Cumulative Returns

[71]:
```
# 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)
```

```
Date
2021-04-27    -1.109565
2021-04-28    -0.085933
2021-04-29    -2.177476
2021-04-30    -3.261423
2021-05-03    -3.223212
                 ...
2026-01-08    35.298223
2026-01-09    35.244222
```

```
2026-01-12    33.511839
2026-01-13    28.804319
2026-01-14    30.576776
Name: portfolio cumulative returns, Length: 1186, dtype: float64
```

[72]:
```
# -- PLOT CUMULATIVE RETURNS --
functions.plot_cumulative_returns(pd.DataFrame(portfolio_cumulative_returns))
```

Portfolio Cumulative Final

[73]:
```
# Portfolio's final comulated return
portafolio_cumulative_final = portfolio_cumulative_returns.iloc[-1]
print(f"portafolio_cumulative Returns_final: {portafolio_cumulative_final:,.
  ↪4}%")
```

7.5 Portfolio Volatility (Annualized)

[74]:
```
# 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}%")
```

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.

[75]:
```
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}")
```

7.7 Summary

[76]:
```
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}")
```

8 Benchmarks (Indices)

8.1 Daily Index Levels

[77]:
```
# 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
```

Date	EE.UU. (S&P 500)	EE.UU. (NASDAQ)	EE.UU. (DJIA)	\
2026-01-08	49266.109375	10044.700195	25127.460938	
2026-01-09	49504.070312	10124.599609	25261.640625	
2026-01-12	49590.199219	10140.700195	25405.339844	
2026-01-13	49191.988281	10137.400391	25420.660156	
2026-01-14	49149.628906	10184.400391	25286.240234	

Date	EE.UU. (Russell 100)	México (IPC)	Japón (Nikkei 225)	\
2026-01-08	6921.459961	23480.019531	65521.011719	
2026-01-09	6966.279785	23671.349609	66062.617188	

```
2026-01-12          6977.270020  23733.900391        66745.609375
2026-01-13          6963.740234  23709.869141        66337.421875
2026-01-14          6926.600098  23471.750000        67403.078125

            Alemania (DAX)  Reino Unido (FTSE 100)
Date
2026-01-08    51117.261719             3780.199951
2026-01-09    51939.890625             3803.899902
2026-01-12             NaN             3809.320068
2026-01-13    53549.160156             3802.500000
2026-01-14    54341.230469             3783.669922
```

[78]:
```
# -- PLOT ASSET PRICES --
functions.plot_prices(prices=benchmarks_prices, yaxis_label="Index Level")
```

8.2 Normalized Index Levels

[79]:
```
# 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

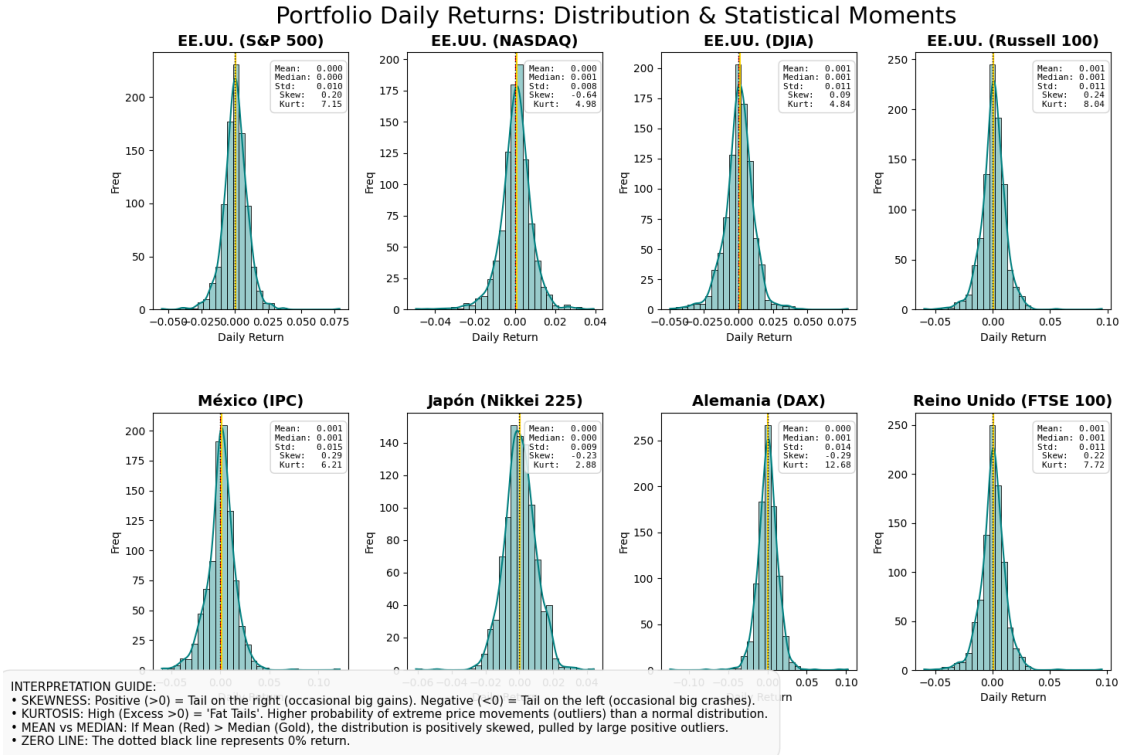
[80]:
```
# Daily Returns
daily_returns_bm = benchmarks_prices.pct_change(fill_method=None)

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

[80]:
```
            EE.UU. (S&P 500)  EE.UU. (NASDAQ)  EE.UU. (DJIA)  \
Date
2026-01-06          0.009900         0.011805       0.000945
2026-01-07         -0.009421        -0.007360       0.009242
2026-01-08          0.005511        -0.000348       0.000207
2026-01-09          0.004830         0.007954       0.005340
2026-01-14         -0.000861         0.004636      -0.005288

            EE.UU. (Russell 100)  México (IPC)  Japón (Nikkei 225)  \
Date
2026-01-06              0.006197      0.006469            0.000121
2026-01-07             -0.003440      0.001576           -0.002315
2026-01-08              0.000077     -0.004421            0.010009
```

2026-01-09	0.006475	0.008149	0.008266
2026-01-14	-0.005333	-0.010043	0.016064

Date	Alemania (DAX)	Reino Unido (FTSE 100)
2026-01-06	0.013221	0.006450
2026-01-07	-0.010589	-0.003535
2026-01-08	-0.016256	0.000074
2026-01-09	0.016093	0.006269
2026-01-14	0.014791	-0.004952

[81]:
```
results_df = functions.plot_returns_distributions(daily_returns_bm, num_cols=4)
```



<pandas.io.formats.style.Styler at 0x21cec983920>

[82]:
```
functions.plot_daily_returns(daily_returns_bm)
```

8.4 Annualized Returns

[83]:
```
# 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)
```

8.5 Cumulative Daily Returns

[84]:
```
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
2026-01-06         34.248604        33.539066      55.726740
2026-01-07         32.983799        32.556259      57.166009
2026-01-08         33.716711        32.510087      57.198548
2026-01-09         34.362578        33.564123      58.037982
2026-01-14         34.246878        34.183366      57.202305

            EE.UU. (Russell 100)  México (IPC)  Japón (Nikkei 225)  \
Date
2026-01-06             55.400121     71.853038           40.797690
2026-01-07             54.865557     72.123871           40.471716
2026-01-08             54.877411     71.362958           41.877723
2026-01-09             55.880318     72.759331           43.050503
2026-01-14             55.048952     71.024303           45.348492

            Alemania (DAX)  Reino Unido (FTSE 100)
Date
2026-01-06       36.248583               51.833991
2026-01-07       34.805889               51.297228
2026-01-08       32.614420               51.308437
2026-01-09       34.748581               52.257065
2026-01-14       36.741709               51.503084
```

[85]:
```
functions.plot_cumulative_returns(cumulative_returns_bm)
```

[86]:
```
# 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)
```

```
México (IPC)              71.024303
EE.UU. (DJIA)             57.202305
EE.UU. (Russell 100)      55.048952
```

```
Reino Unido (FTSE 100)    51.503084
Japón (Nikkei 225)        45.348492
Alemania (DAX)            36.741709
EE.UU. (S&P 500)          34.246878
EE.UU. (NASDAQ)           34.183366
Name: final cumulative returns Benchmarks (%), dtype: float64
```

8.6 Volatility

[87]:
```
#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)
```

```
México (IPC)              23.318459
Alemania (DAX)            22.176649
Reino Unido (FTSE 100)    17.877556
EE.UU. (Russell 100)      17.542846
EE.UU. (DJIA)             17.213699
EE.UU. (S&P 500)          15.093692
Japón (Nikkei 225)        15.049194
EE.UU. (NASDAQ)           12.722783
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)

[88]:
```
# 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)
```

```
EE.UU. (DJIA)             0.61
México (IPC)              0.60
EE.UU. (Russell 100)      0.58
```

```
Japón (Nikkei 225)          0.53
Reino Unido (FTSE 100)      0.53
EE.UU. (NASDAQ)             0.43
EE.UU. (S&P 500)            0.38
Alemania (DAX)              0.34
Name: Sharpe Ratio, dtype: float64
```

8.8 Summary

[89]:
```
# 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}%")
```

9 Compare Initial Portfolio and Indices

9.1 Daily Returns

Combine dataframes

[90]:
```
# 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")
```

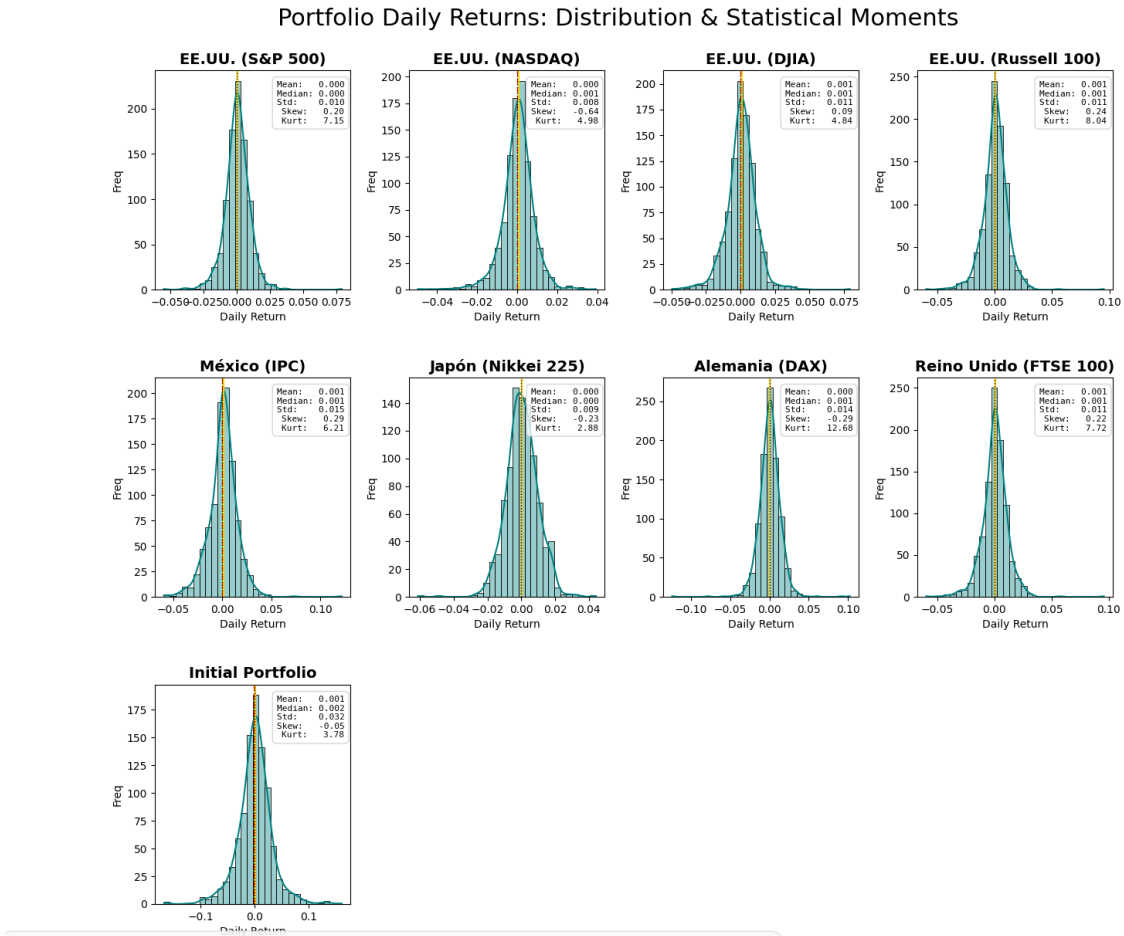
```
# Merge Daily Returns of Benchmark Indices and Initial Portfolio
# Note: Left-Merge on Benchmark daily returns because they don't operate on␣
 ↪weekends or bank holidays
# thus we compare both benchmark and portfolio using the same labor day dates␣
 ↪only.
merge_daily_returns = pd.merge(daily_returns_bm,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

Date	EE.UU. (S&P 500)	EE.UU. (NASDAQ)	EE.UU. (DJIA)	\
2026-01-06	0.009900	0.011805	0.000945	
2026-01-07	-0.009421	-0.007360	0.009242	
2026-01-08	0.005511	-0.000348	0.000207	
2026-01-09	0.004830	0.007954	0.005340	
2026-01-14	-0.000861	0.004636	-0.005288	

Date	EE.UU. (Russell 100)	México (IPC)	Japón (Nikkei 225)	\
2026-01-06	0.006197	0.006469	0.000121	
2026-01-07	-0.003440	0.001576	-0.002315	
2026-01-08	0.000077	-0.004421	0.010009	
2026-01-09	0.006475	0.008149	0.008266	
2026-01-14	-0.005333	-0.010043	0.016064	

Date	Alemania (DAX)	Reino Unido (FTSE 100)	Initial Portfolio
2026-01-06	0.013221	0.006450	0.017867
2026-01-07	-0.010589	-0.003535	-0.011159
2026-01-08	-0.016256	0.000074	0.007844
2026-01-09	0.016093	0.006269	-0.000399
2026-01-14	0.014791	-0.004952	0.013761

[91]:
```
results_df = functions.plot_returns_distributions(merge_daily_returns,␣
 ↪num_cols=4)
```




```
<pandas.io.formats.style.Styler at 0x21cea483530>
```

[92]:
```
functions.plot_daily_returns(merge_daily_returns)
```

9.2 Cumulative Returns

[93]:
```
# 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))
```

Date	EE.UU. (S&P 500)	EE.UU. (NASDAQ)	EE.UU. (DJIA)	\
2026-01-06	34.248604	33.539066	55.726740	
2026-01-07	32.983799	32.556259	57.166009	
2026-01-08	33.716711	32.510087	57.198548	
2026-01-09	34.362578	33.564123	58.037982	
2026-01-14	34.246878	34.183366	57.202305	

Date	EE.UU. (Russell 100)	México (IPC)	Japón (Nikkei 225)	\
2026-01-06	55.400121	71.853038	40.797690	
2026-01-07	54.865557	72.123871	40.471716	
2026-01-08	54.877411	71.362958	41.877723	
2026-01-09	55.880318	72.759331	43.050503	
2026-01-14	55.048952	71.024303	45.348492	

Date	Alemania (DAX)	Reino Unido (FTSE 100)	Initial Portfolio
2026-01-06	36.248583	51.833991	35.760188
2026-01-07	34.805889	51.297228	34.245238
2026-01-08	32.614420	51.308437	35.298223
2026-01-09	34.748581	52.257065	35.244222
2026-01-14	36.741709	51.503084	30.576776

[94]:
```
functions.plot_cumulative_returns(merge_cumulative_returns)
```

[95]:
```
# 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)
```

```
México (IPC)              71.024303
EE.UU. (DJIA)             57.202305
EE.UU. (Russell 100)      55.048952
Reino Unido (FTSE 100)    51.503084
Japón (Nikkei 225)        45.348492
Alemania (DAX)            36.741709
EE.UU. (S&P 500)          34.246878
EE.UU. (NASDAQ)           34.183366
Initial Portfolio         30.576776
Name: Final Cumulative Returns (%), dtype: float64
```

9.3 Volatility

[96]:
```
# 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)
```

```
Initial Portfolio         50.472403
México (IPC)              23.318459
Alemania (DAX)            22.176649
Reino Unido (FTSE 100)    17.877556
EE.UU. (Russell 100)      17.542846
EE.UU. (DJIA)             17.213699
EE.UU. (S&P 500)          15.093692
Japón (Nikkei 225)        15.049194
EE.UU. (NASDAQ)           12.722783
Name: Annualized Volatility (%), dtype: float64
```

9.4 Sharpe Ratio

[97]:
```
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
```

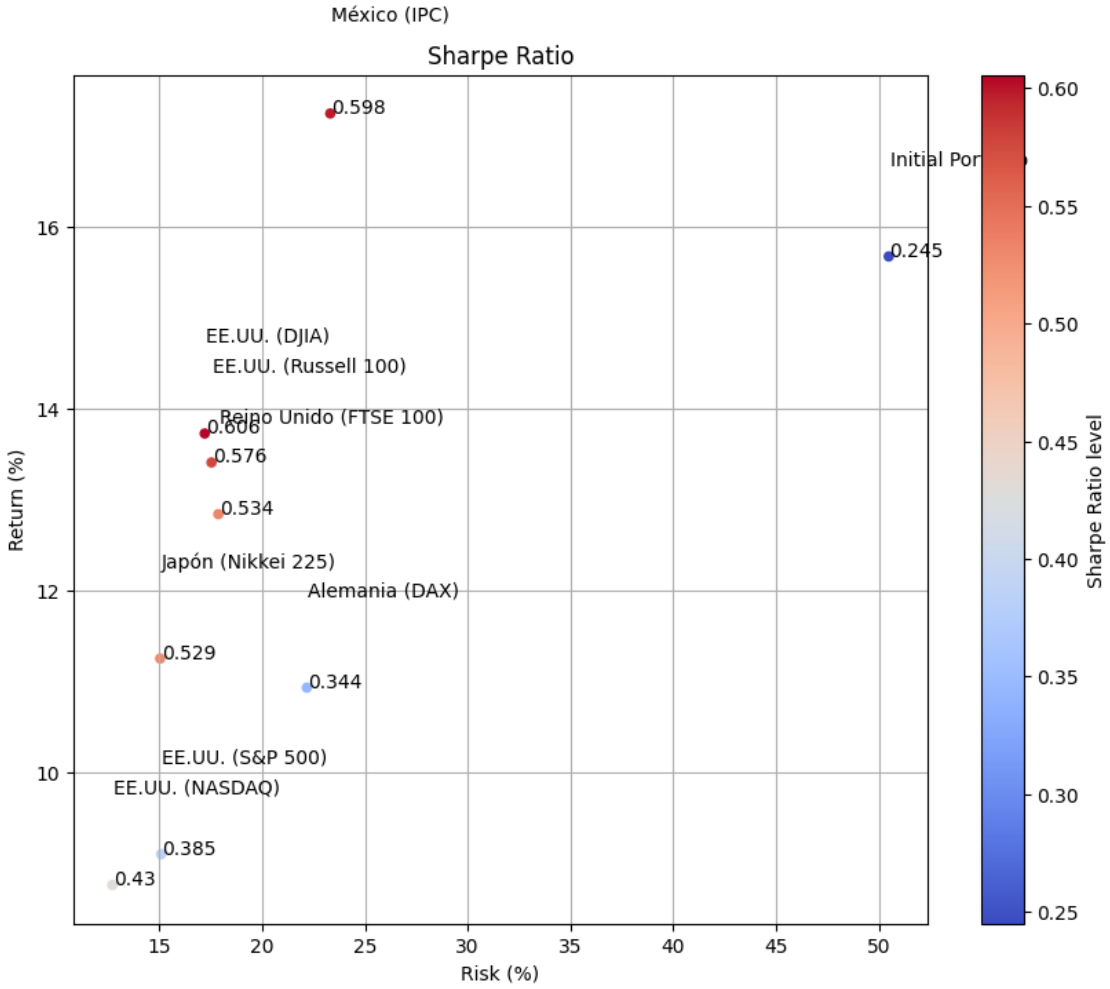
[97]:

	Annualized Returns (%)	Annualized Volatility (%)	\
EE.UU. (DJIA)	13.726229	17.213699	
México (IPC)	17.238259	23.318459	
EE.UU. (Russell 100)	13.408435	17.542846	
Reino Unido (FTSE 100)	12.841209	17.877556	
Japón (Nikkei 225)	11.256975	15.049194	
EE.UU. (NASDAQ)	8.771307	12.722783	
EE.UU. (S&P 500)	9.109451	15.093692	
Alemania (DAX)	10.937215	22.176649	
Initial Portfolio	15.669442	50.472403	

	Sharpe Ratio
EE.UU. (DJIA)	0.605694
México (IPC)	0.597735
EE.UU. (Russell 100)	0.576214
Reino Unido (FTSE 100)	0.533698
Japón (Nikkei 225)	0.528731
EE.UU. (NASDAQ)	0.430040
EE.UU. (S&P 500)	0.384893
Alemania (DAX)	0.344381
Initial Portfolio	0.245073

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

[98]:
```
#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

[99]:
```
# 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}")
```

10 Buy and Hold

10.1 B&H - Stocks (Initial Portfolio)

Initial Investment $100,000 in each Asset

[100]:
```
# B&H Stocks in initial Portfolio
functions.buy_and_hold_strategy(cumulative_returns, 100000, "Portfolio Stocks")
```

```
<pandas.io.formats.style.Styler at 0x21cebb328a0>
```

10.2 B&H - Indices and Portfolio

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

[101]:
```
# B&H Indices & Portfolio (merged)
functions.buy_and_hold_strategy(merge_cumulative_returns, 100000, "Market␣
  ↪Benchmarks Indices and Initital Portfolio")
```

```
<pandas.io.formats.style.Styler at 0x21ceadcd6d0>
```

11 Dollar Cost Averaging (DCA)

11.1 DCA - Stocks (Initial Portfolio)

Monthly Investment $100 in each Asset

[102]:
```
# For individual stocks
functions.dollar_cost_averaging_strategy(prices, monthly_investment=100,␣
  ↪title_suffix="Individual Stocks")
```

```
<pandas.io.formats.style.Styler at 0x21cea55ab70>
```

11.2 DCA - Indices

Monthly Investment $100 in each Asset

[103]:
```
# For Benchmarks and your Portfolio Index
functions.dollar_cost_averaging_strategy(benchmarks_prices,␣
  ↪monthly_investment=100, title_suffix="Indices & Portfolio")
```

```
<pandas.io.formats.style.Styler at 0x21ceadcc200>
```

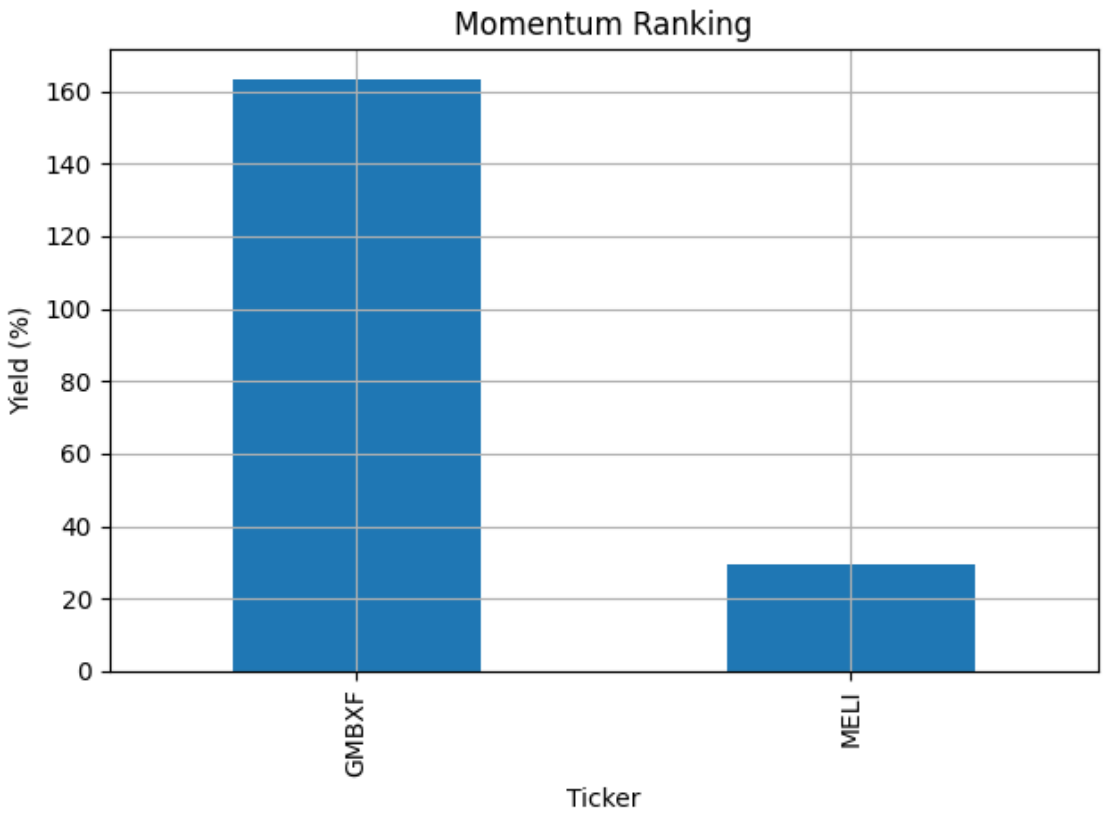
12 Momentum

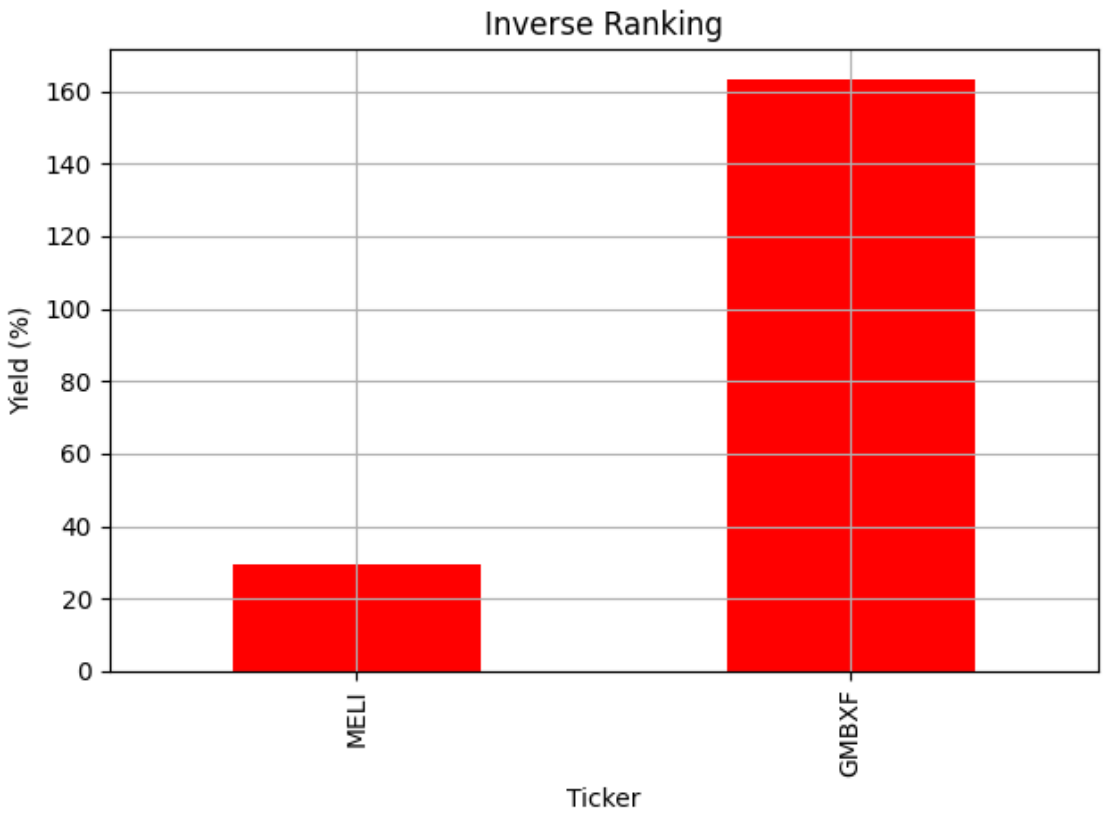
Applied to all **Stocks** in Initial Portfolio

[104]:
```
# 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()
```

```
Ticker
GMBXF    163.3
MELI      29.5
Name: Final Cumulative Returns (%), dtype: float64
```

```
Ticker
MELI      29.5
GMBXF    163.3
Name: Final Cumulative Returns (%), dtype: float64
```


Momentum Ranking
Yield (%)
160
140
120
100
80
60
40
20
0
GMBXF
MELI
Ticker



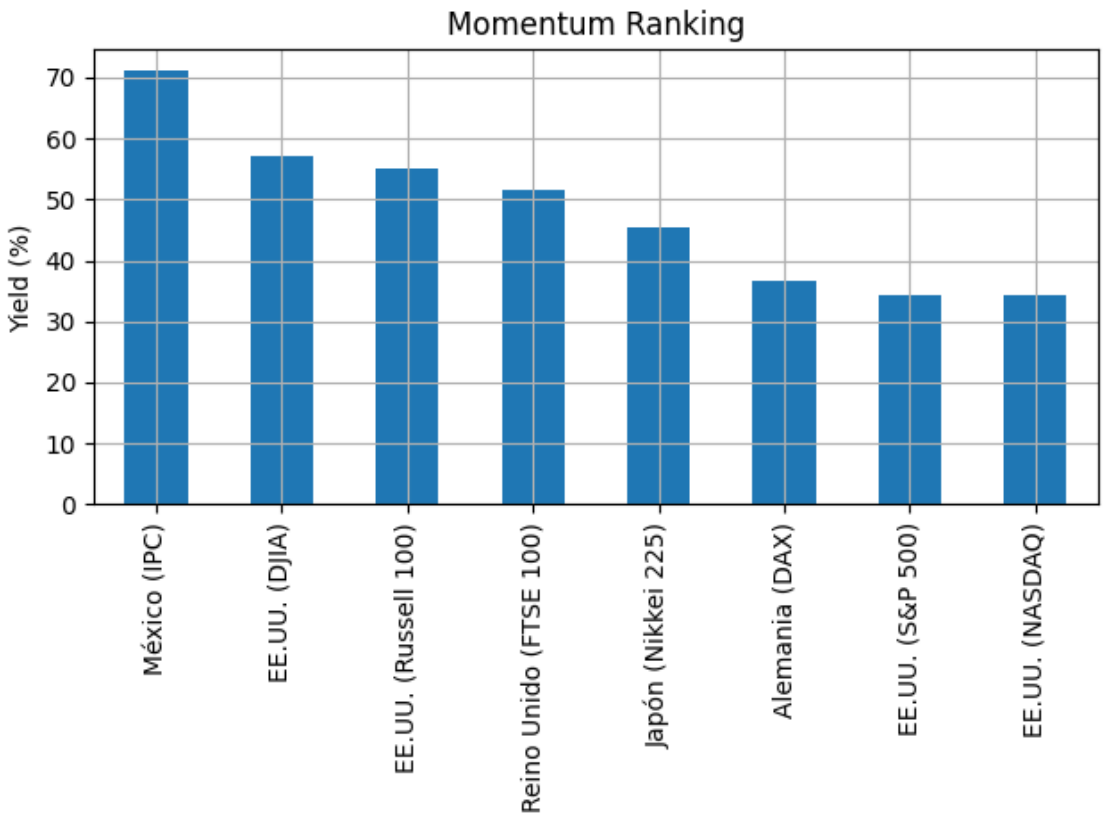
Applied to Indices

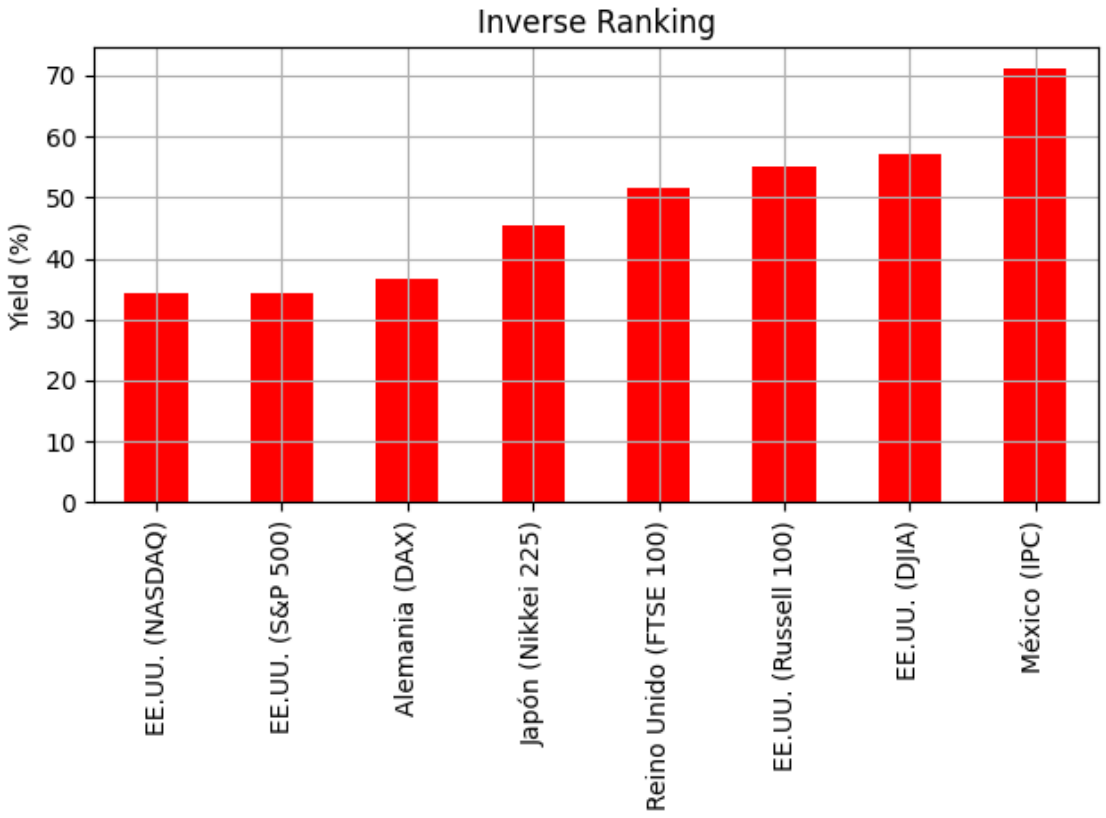
[105]:
```
# call the DataFrame of the Final Cumulative Returns, which is the total return␣
↪of each Index in the period.
cumulative_returns_final_bm

# 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()
```

```
México (IPC)              71.024303
EE.UU. (DJIA)             57.202305
EE.UU. (Russell 100)      55.048952
Reino Unido (FTSE 100)    51.503084
Japón (Nikkei 225)        45.348492
Alemania (DAX)            36.741709
EE.UU. (S&P 500)          34.246878
EE.UU. (NASDAQ)           34.183366
Name: final cumulative returns Benchmarks (%), dtype: float64
```

```
EE.UU. (NASDAQ)           34.183366
EE.UU. (S&P 500)          34.246878
Alemania (DAX)            36.741709
Japón (Nikkei 225)        45.348492
Reino Unido (FTSE 100)    51.503084
EE.UU. (Russell 100)      55.048952
EE.UU. (DJIA)             57.202305
México (IPC)              71.024303
Name: final cumulative returns Benchmarks (%), dtype: float64
```


Momentum Ranking
Yield (%)
0
10
20
30
40
50
60
70
México (IPC)
EE.UU. (DJIA)
EE.UU. (Russell 100)
Reino Unido (FTSE 100)
Japón (Nikkei 225)
Alemania (DAX)
EE.UU. (S&P 500)
EE.UU. (NASDAQ)



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

[106]:
```
functions.plot_interactive_heatmap_correlation(daily_returns, triangle='half')
functions.plot_extreme_correlations(daily_returns, num_pairs=10)
```

```
<pandas.io.formats.style.Styler at 0x21cecbbb200>
```

13.3 Indices & Portfolio

[107]:
```
functions.plot_interactive_heatmap_correlation(merge_daily_returns,␣
↪triangle='half')
functions.plot_extreme_correlations(merge_daily_returns, num_pairs=10)
```

```
<pandas.io.formats.style.Styler at 0x21cead59c40>
```

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:
$$\text{Cov}(X, Y) = E[(X - \mu_X)(Y - \mu_Y)]$$
- Values:
  - Positive → one rises, other rises too.
  - Negative → one rises, other goes down.
  - near 0 → there's no clear relationtip.

14.1 Stocks

[108]:
```
# Plot ANNUALIZED variance/covariance Matrix =  returns.cov() * 252
functions.plot_annualized_covariance_heatmap(daily_returns, triangle='half')
```

14.2 Indices & Portfolio

[109]:
```
# 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

15.1 Stocks

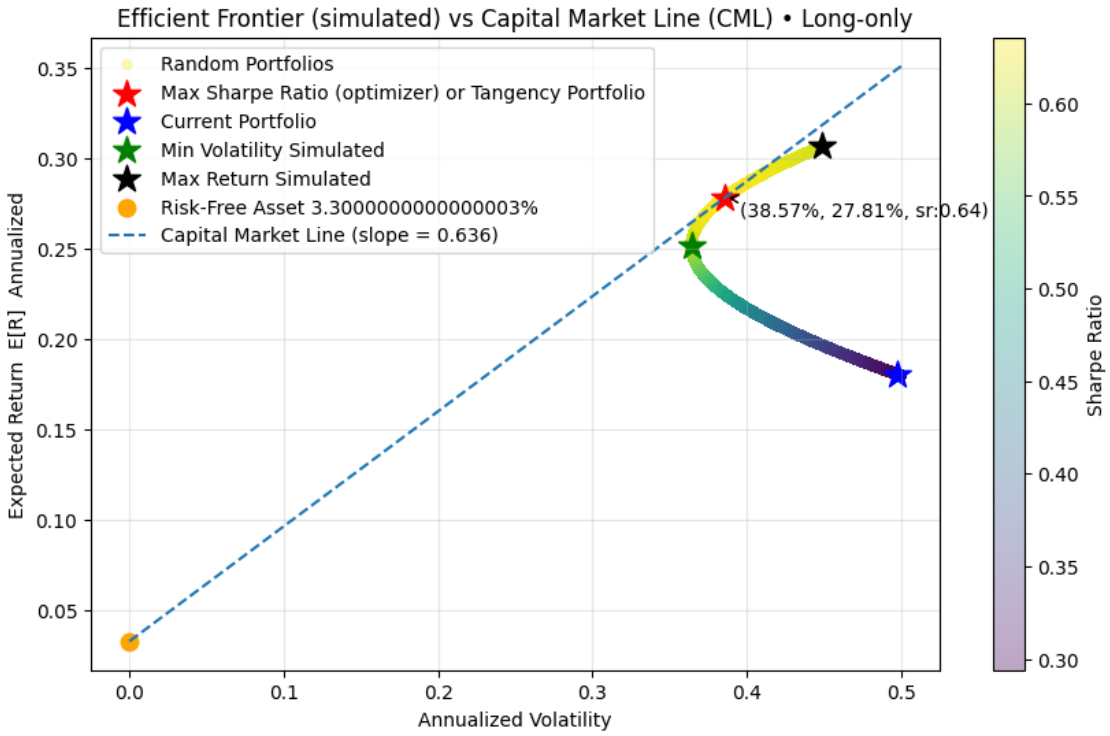
[110]:
```
# --- EFFICIENT FRONTIER & CAPITAL MARKET LINE ANALYSIS---

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, #70000
                           yields = yields/100
                           )
```

```
VBox(children=(FloatText(value=211300.0, description='Portfolio Value (V): $',␣
 ↪layout=Layout(width='350px'), s…
```

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

[111]:
```
no_starts = 25
no_simul = 2000000
```

[112]:
```
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
                )
```



15.1.1 Positions from current to optimal

[113]:
```
# 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
GMBXF           100        11.05    1105.000    0.005            0.776
MELI            100      2101.95  210194.995    0.995            0.224

        New Investment usd    New QTY  Position_to_Optimal
Ticker
GMBXF           163884.276  14831.156            14731.156
MELI             47415.719     22.558              -77.442
```

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:

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

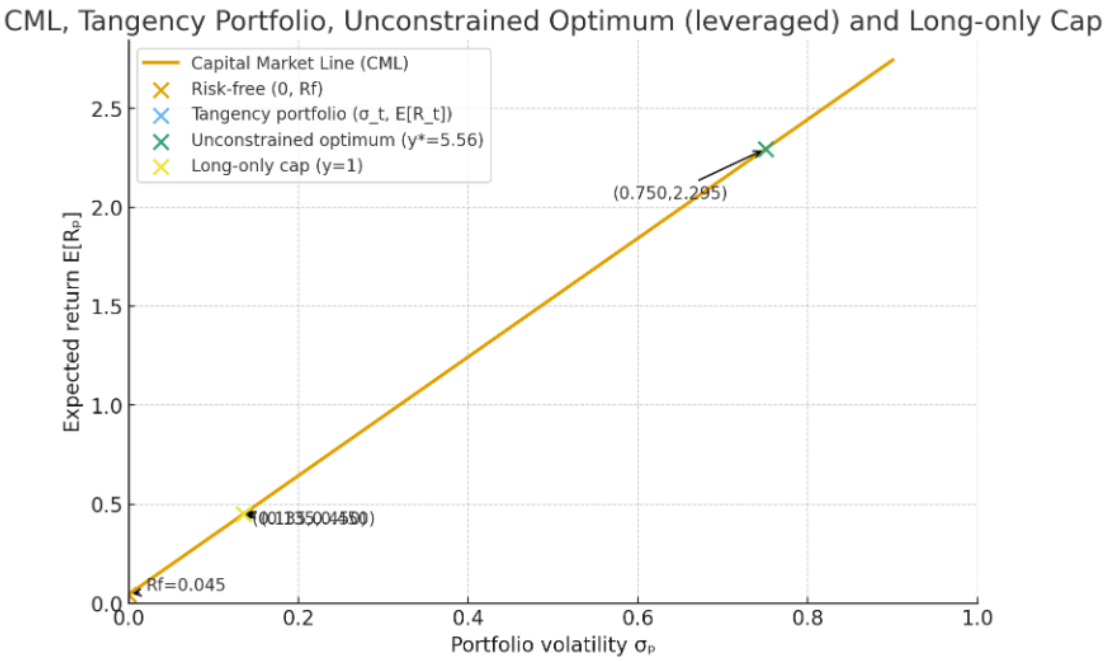
- Objective 3. Mean-Variance Utility (Expected Utility Funcion):

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

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

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.

[114]:
```
# 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} ")
```

[115]:
```
# 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}")
```

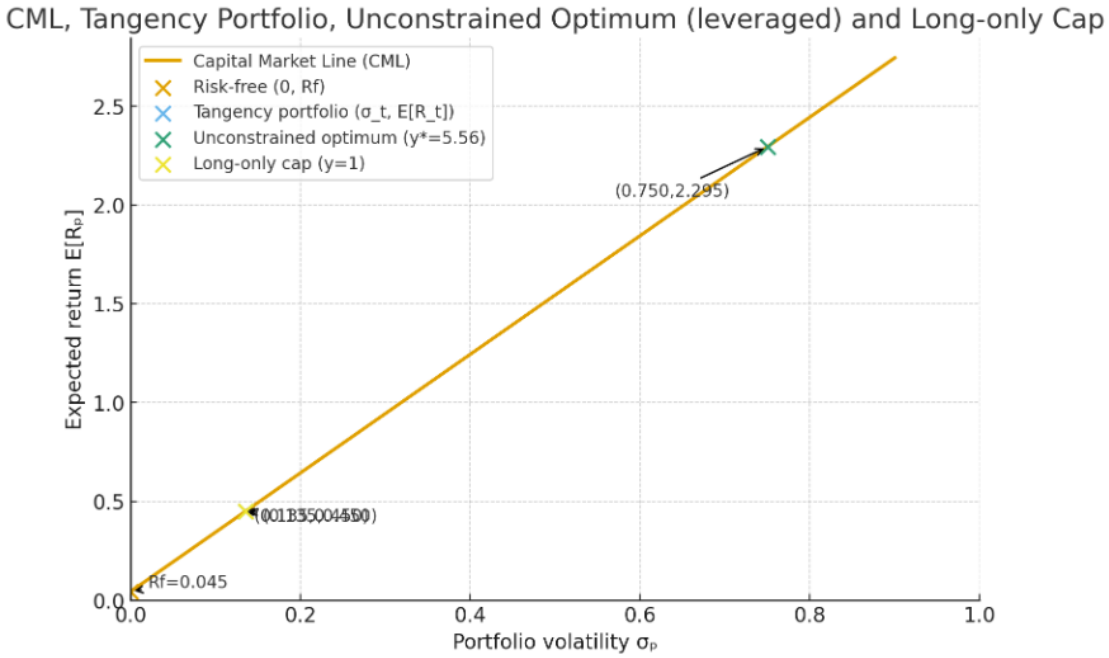
[116]:
```
# Objective 3. You maximize mean-variance Utility (risk aversion )

# 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}")
```

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

[117]:
```
# ANNUAL DIVIDEND YIELDS
yields_merge = functions.plot_annual_dividnd_yields(merge_daily_returns)
```

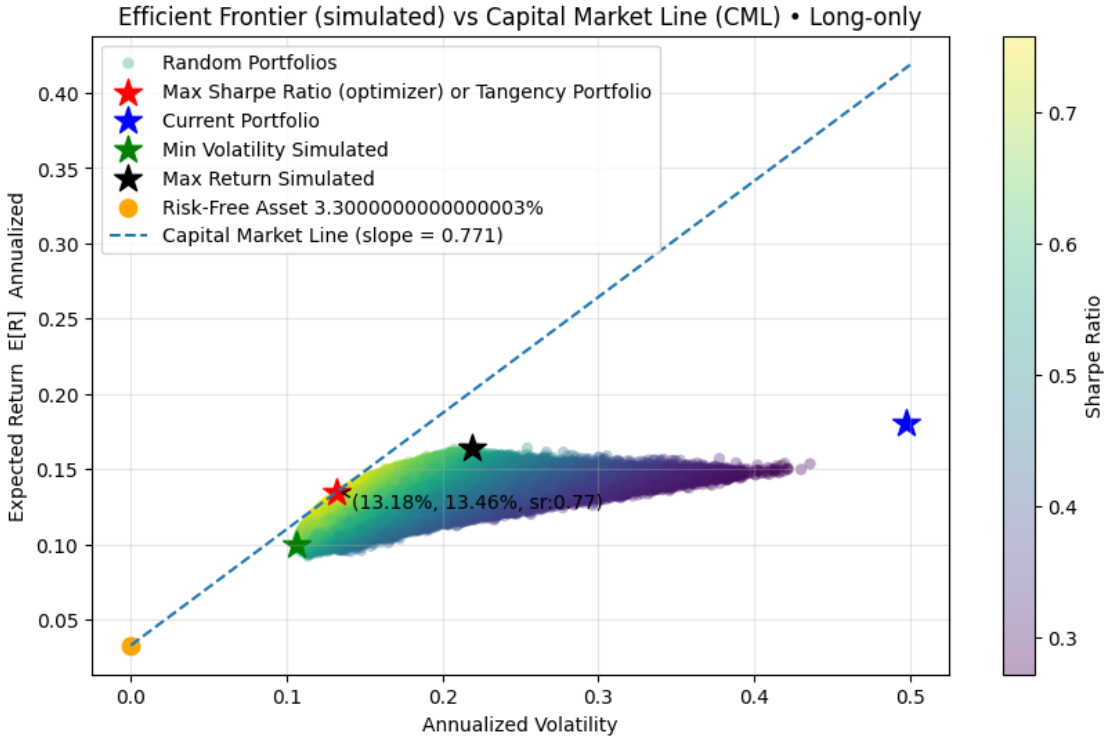
```
HTTP Error 404:
HTTP Error 404:
HTTP Error 404:
HTTP Error 404:
HTTP Error 404:
HTTP Error 404:
HTTP Error 404:
HTTP Error 404:
HTTP Error 404:
```

```
EE.UU. (S&P 500)          0
EE.UU. (NASDAQ)           0
EE.UU. (DJIA)             0
EE.UU. (Russell 100)      0
México (IPC)              0
Japón (Nikkei 225)        0
Alemania (DAX)            0
Reino Unido (FTSE 100)    0
Initial Portfolio         0
Name: dividendYield, dtype: int64
```

[118]:
```
# --- EFFICIENT FRONTIER & CAPITAL MARKET LINE ANALYSIS---

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

functions.run_full_frontier_analysis(rets = merge_daily_returns,
                          curr_port_weights = weights_df['Weights'],
                          curr_port_vol = portfolio_annualized_volatility,
                          curr_port_ret = portfolio_annualized_return,
                          mean_ann = merge_daily_returns.mean() * 252,
                          cov_ann = merge_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_merge/100
                          )
```

```
VBox(children=(FloatText(value=211300.0, description='Portfolio Value (V): $',␣
 ↪layout=Layout(width='350px'), s…
```

[119]:
```
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
                )
```



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.

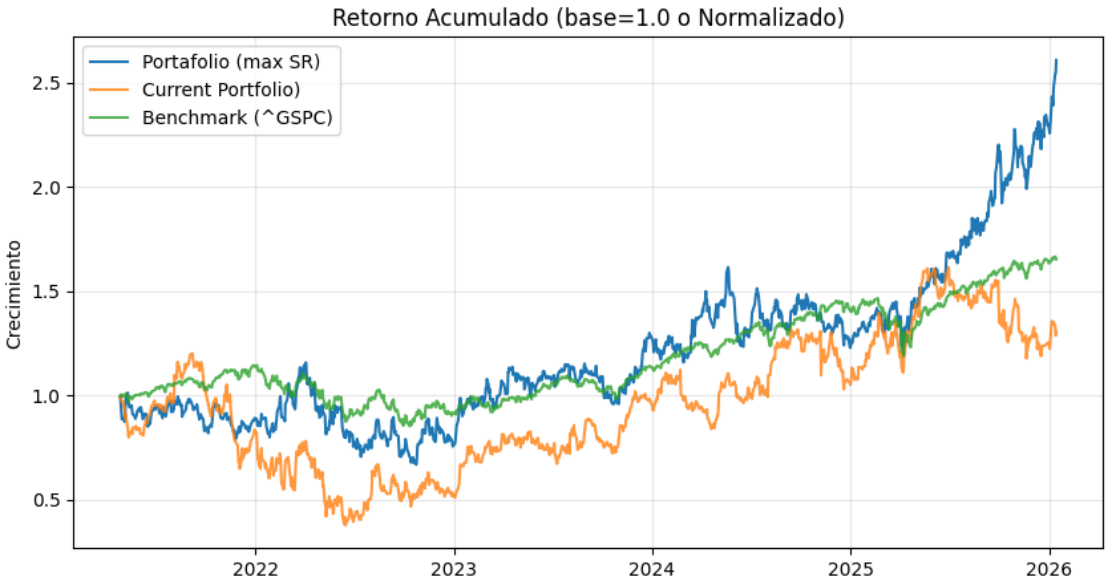
Métrica	Valor Óptimo / Bueno	Interpretación
(Volatilidad anual)	10%–20% moderado, <10% defensivo, >25% muy riesgoso	Mide el riesgo total (desviación estándar).
Sharpe Ratio	> 1 bueno, > 1.5 muy bueno, > 2 excelente	Retorno ajustado por riesgo total.
Sortino Ratio	> 2 excelente	Retorno ajustado por riesgo 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).

[120]:
```
#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'}
```

[121]:
```
# 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)
```

```
                       anual    anual  Sharpe  Sortino
Current Portfolio     0.1806   0.4974  0.2967   0.4864
Portafolio (max SR)   0.2781   0.3857  0.6355   1.0914
Benchmark [^GSPC]     0.1214   0.1703  0.5191      NaN

                    Alpha (anual)    Beta      R²  VaR 1d  CVaR 1d
Current Portfolio          0.0012  1.6550  0.3212  0.0502   0.0728
Portafolio (max SR)        0.1618  0.9422  0.1732  0.0382   0.0518
```



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:

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

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

20 PDF EXPORT