Team_XX_Assignment (3)

November 28, 2025

[22]:
```
#These are the libraries you can use.  You may add any libraries directy␣
 ↪related to threading if this is a direction
#you wish to go (this is not from the course, so it's entirely on you if you␣
 ↪wish to use threading).  Any
#further libraries you wish to use you must email me, james@uwaterloo.ca, for␣
 ↪permission.

from IPython.display import display, Math, Latex

import pandas as pd
import numpy as np
import numpy_financial as npf
import yfinance as yf
import matplotlib.pyplot as plt
import random
from datetime import datetime

import asyncio
import httpx
from scipy.optimize import minimize
import seaborn as sns
```

0.1 Group Assignment

0.1.1 Team Number: 10

0.1.2 Team Member Names: David, Tanvi, Johan

0.1.3 Team Strategy Chosen: Market Meet

0.2 Contribution Declaration

The following team members made a meaningful contribution to this assignment:

David, Tanvi, and Johan

Markdown Introduction:
Goal: Try to have the 25-stock portfolio's returns match the average of the TSX composite + S&P 500's return (~0.370%) as accurately as possible, over the 5-day period (Nov. 24 to Nov. 28).
- **S&P/TSX Composite Index** (`^GSPTSE`) - **S&P 500 Index** (`^GSPC`)

| Index | YTD Return | Daily Return (÷ 252)| 5-Day Estimated Return |
|:——|:———:|:———————:|:———————-:| | TSX Composite | 21.74% | 0.0863% | 0.43% | | S&P 500 | 15.55% | 0.0617% | 0.31% | | **Average (50/50)** | — | — | **0.370%** |
Use historical data from `yfinance` to calculate: - Mean returns - Volatility (standard deviation) - Beta and alpha values - Correlation with the benchmark - Idiosyncratic (residual) risk
Select 10–25 stocks (we aim for 25 if possible) that: - Have **beta** **1** and high correlation with the benchmark - Are liquid (average daily volume 5,000 shares) - Have sector diversification (no sector >40% of total value) - Include at least one large-cap (> 10B CAD) and one small-cap (< $2B CAD)
Weights are between (100/(2n))% and 15% (for 25 stocks, between 2% and 15%) - Spend approximately **$1,000,000 CAD**, net of trading fees: - Fees = min(2.15 USD, 0.001 USD * shares) per trade, applied to all stock purchases

[23]:
```
## === Load Stock Tickers from the TA's CSV File ===

tickers_df = pd.read_csv("Tickers.csv", header=None)
tickers_list = tickers_df[0].dropna().astype(str).tolist()

# Clean tickers BEFORE download
tickers_list = [t.strip().upper() for t in tickers_list if isinstance(t, str)␣
 ↪and t.strip() != ""]
```

[24]:
```
# === 1. Original Setup & Data Download ===

sp500 = "^GSPC"
tsx = "^GSPTSE"
start = "2024-01-01"
end = "2025-9-30"

min_vol = 5_000
corr_min  = 0.20
small_cap = 2_000_000_000
large_cap = 10_000_000_000
earn_low  = datetime(2025, 11, 24).date()
earn_high = datetime(2025, 11, 28).date()

# Download price + volume data
tickers_all = tickers_list + [sp500, tsx]
print("Starting synchronous download of price/volume data...")
data = yf.download(tickers_all, start=start, end=end, auto_adjust=False,␣
 ↪progress=False)
close = data["Close"]
vol = data["Volume"]
print("Price/Volume download complete.")

# Remove any tickers that did not download successfully
```

```
valid_tickers = [t for t in tickers_list if t in close.columns]

missing = set(tickers_list) - set(valid_tickers)
if missing:
    print("  WARNING: These tickers were not found on Yahoo Finance and were␣
 ↪removed:")
    print(missing)

tickers_list = valid_tickers

# Benchmark calculation
bench = ((close[sp500] + close[tsx]) / 2).dropna()
bench_ret = bench.pct_change().dropna()
stocks = close[tickers_list].loc[bench_ret.index].dropna(how="all", axis=1)
rets = stocks.pct_change().dropna()

# FX rate (Keep synchronous as it's a single, fast call)
fx_raw = yf.Ticker("CADUSD=X").history(period="1d")["Close"]
usd_to_cad = fx_raw.iloc[-1] if len(fx_raw) > 0 else 0.73  # fallback rate

async def async_get_meta(ticker):
    """Asynchronously fetches metadata for a single ticker."""
    t = ticker
    try:
        ticker_obj = yf.Ticker(t)

        # 1. Retrieve stock info
        # Use a try/except for info specifically to avoid crashing the whole␣
 ↪ticker
        try:
            info = await asyncio.to_thread(lambda: ticker_obj.info)
        except:
            info = {}

        sector = info.get("sector")
        industry = info.get("industry")
        mc_raw = info.get("marketCap", np.nan)

        # 2. Market cap. conversion
        if isinstance(mc_raw, (int, float)) and not pd.isna(mc_raw):
            mc = mc_raw if t.endswith(".TO") else mc_raw / usd_to_cad
        else:
            mc = np.nan

        # Define booleans safely
        small = mc < small_cap if not pd.isna(mc) else False
        large = mc > large_cap if not pd.isna(mc) else False
```

```
        # 3. Fetch Earnings Date
        earn = None

        # Attempt #1: Calendar (Fast)
        try:
            cal = await asyncio.to_thread(lambda: ticker_obj.calendar)
            if cal and "Earnings Date" in cal:
                dates = cal["Earnings Date"]
                if dates:
                    earn = dates[0]
        except Exception:
            pass

        # Attempt 2: Full History (Backup)
        if earn is None:
            try:
                e_df = await asyncio.to_thread(lambda: ticker_obj.
↪get_earnings_dates(limit=12))
                if e_df is not None and not e_df.empty:
                    future = [d.date() for d in e_df.index.to_pydatetime() if d.
↪date() >= datetime.now().date()]
                    if future:
                        earn = min(future)
            except Exception:
                pass

        return {
            "Ticker": t,
            "Sector": sector,
            "Industry": industry,
            "MarketCap": mc,
            "Small-cap": small,
            "Large-cap": large,
            "Earnings Date": earn
        }

    except Exception as e:
        # Fallback in case of total code failure
        return {
            "Ticker": t,
            "Sector": np.nan, "Industry": np.nan, "MarketCap": np.nan,
            "Small-cap": False,
            "Large-cap": False,
            "Earnings Date": None
        }
```

```python
async def fetch_all_metadata(tickers):
    print("Starting asynchronous metadata fetching with delay...")
    tasks = []
    for i, t in enumerate(tickers):
        # Get the metadata
        task = async_get_meta(t)
        tasks.append(task)

        # Pause execution for a short time every few requests to reduce the␣
 ↪risk of timeout
        if i > 0 and i % 5 == 0: # Example: pause every 5 tickers
            await asyncio.sleep(0.25)

    results = await asyncio.gather(*tasks)
    print("Metadata retrieval complete.")
    return results

metadata_results = await fetch_all_metadata(stocks.columns)

# Convert results to DataFrame
meta = pd.DataFrame(metadata_results).set_index("Ticker")
meta = meta.loc[stocks.columns] # Re-align index order
meta["Small-cap"] = meta["Small-cap"].astype(bool)
meta["Large-cap"] = meta["Large-cap"].astype(bool)

# Helper functions
def avg_volume(series):
    """Average daily volume after removing months with <18 trading days."""
    s = series.dropna()
    if s.empty: return np.nan
    month = s.index.to_period("M")
    valid = month.value_counts()[lambda x: x >= 18].index
    return s[month.isin(valid)].mean()

def weekly_vol(r):
    """Weekly volatility computed from daily returns."""
    w = (1 + r).resample("W-FRI").prod() - 1
    w = w.dropna()
    return w.std() if not w.empty else np.nan
```

```
Starting synchronous download of price/volume data…


6 Failed downloads:
['ASDFA.TO', 'INVALIDTIC', 'CELG', 'HDFC.NS', 'SQ']: YFTzMissingError('possibly
delisted; no timezone found')
['XZO']: YFPricesMissingError('possibly delisted; no price data found  (1d
2024-01-01 -> 2025-9-30) (Yahoo error = "Data doesn\'t exist for startDate =
```

```
1704085200, endDate = 1759204800")')

Price/Volume download complete.
Starting asynchronous metadata fetching with delay…

HTTP Error 404: {"quoteSummary":{"result":null,"error":{"code":"Not
Found","description":"No fundamentals data found for symbol: BTC-USD"}}}
HTTP Error 404: {"quoteSummary":{"result":null,"error":{"code":"Not
Found","description":"No fundamentals data found for symbol: VSP.TO"}}}
HTTP Error 404: {"quoteSummary":{"result":null,"error":{"code":"Not
Found","description":"No fundamentals data found for symbol: GC=F"}}}
GC=F: No earnings dates found, symbol may be delisted
VSP.TO: No earnings dates found, symbol may be delisted
BTC-USD: No earnings dates found, symbol may be delisted

Metadata retrieval complete.
```

[25]:
```python
# === 2. Calculate the Stock & Benchmark Metrics ===

# --- New Filter: Ensure only one copy of duplicated tickers (e.g., COST) is␣
 ↪kept ---
stocks = stocks.loc[:, ~stocks.columns.duplicated(keep="first")]
rets = rets.loc[:, ~rets.columns.duplicated(keep="first")]
vol = vol.loc[:, ~vol.columns.duplicated(keep="first")]
meta = meta[~meta.index.duplicated(keep="first")]
tickers_list = list(stocks.columns)

# --- New Filter: Remove tickers missing Sector or Industry ---
meta = meta.dropna(subset=["Sector", "Industry"], how="any")
stocks = stocks.loc[:, stocks.columns.isin(meta.index)]
rets = rets.loc[:, rets.columns.isin(meta.index)]
vol = vol.loc[:, vol.columns.isin(meta.index)]

tickers_list = list(stocks.columns)

metrics = pd.DataFrame(index=stocks.columns,
                       columns=["AvgVol", "StdDev (%)", "Covariance", "Beta",␣
 ↪"Correlation", "WeeklyVol %","IdioVol %"],
                       dtype=float)

for t in stocks.columns:
    r = rets[t].dropna()
    b = bench_ret.reindex(r.index).dropna()

    # Align the indexes
    idx = r.index.intersection(b.index)
    if isinstance(r, pd.DataFrame):
        r = r.iloc[:, 0]
```

```
    if isinstance(b, pd.DataFrame):
        b = b.iloc[:, 0]

    # Compute each of the metrics via their formulas
    av = avg_volume(vol[t].loc[start:end])
    sd = r.std() * 100

    # --- FORCE Series objects to avoid covariance ambiguity ---
    r = pd.Series(r, dtype=float)
    b = pd.Series(b, dtype=float)

    # If insufficient data, assign NaN to all dependent metrics
    if len(r) < 2 or len(b) < 2:
        cv = np.nan
        beta = np.nan
        corr = np.nan
        wv = np.nan
        iv = np.nan
    else:
        cv = float(r.cov(b)) * (100**2)
        var_b = float(b.var())
        beta = cv / (var_b * (100**2)) if var_b > 0 else np.nan
        corr = r.corr(b)
        wv = weekly_vol(r) * 100

# --- NEW CALCULATION: Idiosyncratic Volatility ---
# Formula: Std Dev of (Actual Return - Expected Return)
# Expected Return = Alpha + (Beta * Benchmark Return)
    if not np.isnan(beta):
        # Calculate Alpha
        alpha = r.mean() - beta * b.mean()

        # Calculate the residual value (not explained by the benchmark)
        residuals = r - (alpha + beta * b)

        # Calculate idiosyncratic volatility (Std Dev of residuals)
        iv = residuals.std() * 100
    else:
        iv = np.nan

    metrics.loc[t] = [av, sd, cv, beta, corr, wv, iv]

# --- Apply assignment filters & requirements ---

keep_vol = metrics["AvgVol"] >= min_vol
keep_corr = metrics["Correlation"] >= corr_min
```

```
keep_earn = pd.Series(True, index=metrics.index)
for t in metrics.index:
    e = meta.loc[t, "Earnings Date"]
    if meta.loc[t, "Small-cap"].any() and isinstance(e, datetime):
        if earn_low <= e <= earn_high:
            keep_earn[t] = False

mask = keep_vol & keep_corr & keep_earn

filtered = metrics[mask].join(meta, how="left")
filtered_tickers = list(filtered.index)

print("# of Original Stocks:", len(tickers_list))
print("# of Filtered Stocks:", len(filtered_tickers))
display(filtered.head(len(filtered_tickers)))

# --- Reference data for the benchmark ---

bench_std = bench_ret.std() * 100
bench_week = weekly_vol(bench_ret) * 100

print("Benchmark Std Dev:", bench_std)
print("Benchmark Volatility:", bench_week)
```

```
# of Original Stocks: 30
# of Filtered Stocks: 28
```

Ticker	AvgVol	StdDev (%)	Covariance	Beta	Correlation	\
SU.TO	8.174526e+06	1.769958	0.895912	1.056839	0.549761	
ABBV	5.935053e+06	1.836539	0.504833	0.595512	0.298551	
LOW	2.560915e+06	1.570290	0.723441	0.853388	0.500374	
AUST	1.182165e+05	6.007690	1.446165	1.705929	0.261446	
TD	2.421870e+06	1.115628	0.527291	0.622004	0.513337	
AMZN	4.233414e+07	2.182404	1.300011	1.533522	0.646969	
AXP	2.862413e+06	2.023984	1.469645	1.733626	0.788636	
GCT	1.631175e+06	4.679426	1.838083	2.168245	0.426623	
BK	3.920481e+06	1.520568	1.004159	1.184528	0.717246	
EXE.TO	1.559795e+05	1.653738	0.358953	0.423429	0.235745	
CMCSA	2.157687e+07	1.800610	0.657594	0.775712	0.396652	
SHOP	9.355884e+06	4.003860	2.500498	2.949644	0.678296	
COST	2.067537e+06	1.383979	0.563520	0.664741	0.442233	
CSCO	2.000644e+07	1.437717	0.932496	1.099993	0.704441	
GM	1.307922e+07	2.373532	0.921515	1.087040	0.421676	
GOOG	2.146409e+07	2.064775	0.969633	1.143800	0.510042	
AIM.TO	6.049795e+04	1.828755	0.397134	0.468468	0.235859	
FTG.TO	3.111731e+04	2.341370	0.542133	0.639513	0.251482	
ORCL	1.100925e+07	3.785642	1.688055	1.991268	0.484305	

SAP.TO	5.777861e+05	1.437835	0.588285	0.693954	0.444376
DUOL	8.839092e+05	4.010836	1.618575	1.909308	0.438298
PEP	6.795452e+06	1.427536	0.264448	0.311949	0.201199
SLB	1.341993e+07	2.334673	1.320076	1.557192	0.614107
SPG	1.541920e+06	1.725003	1.112535	1.312371	0.700479
SHOP.TO	2.354477e+06	4.005269	2.459096	2.900805	0.666830
RY.TO	4.475770e+06	1.075878	0.706950	0.833934	0.713669
TD.TO	7.340803e+06	1.087858	0.470795	0.555360	0.470036
AW.TO	1.303553e+04	1.553043	0.317903	0.375006	0.222322

Ticker	WeeklyVol %	IdioVol %	Sector
SU.TO	3.946258	1.478485	Energy
ABBV	4.403813	1.752781	Healthcare
LOW	3.274357	1.359572	Consumer Cyclical
AUST	12.542651	5.798732	Basic Materials
TD	2.410529	0.957418	Financial Services
AMZN	4.192573	1.664119	Consumer Cyclical
AXP	4.022134	1.244467	Financial Services
GCT	11.371530	4.232212	Technology
BK	3.243326	1.059563	Financial Services
EXE.TO	3.543898	1.607128	Healthcare
CMCSA	3.273467	1.652905	Communication Services
SHOP	8.892075	2.941992	Technology
COST	3.083456	1.241291	Consumer Defensive
CSCO	3.286296	1.020437	Technology
GM	3.965573	2.152192	Consumer Cyclical
GOOG	4.247712	1.776015	Communication Services
AIM.TO	4.050123	1.777160	Basic Materials
FTG.TO	4.554208	2.266123	Industrials
ORCL	7.333941	3.312056	Technology
SAP.TO	2.739298	1.288071	Consumer Defensive
DUOL	8.946400	3.605058	Technology
PEP	2.683947	1.398343	Consumer Defensive
SLB	5.042704	1.842577	Energy
SPG	3.315603	1.231087	Real Estate
SHOP.TO	8.945796	2.984766	Technology
RY.TO	1.823441	0.753634	Financial Services
TD.TO	2.198719	0.960195	Financial Services
AW.TO	2.835227	1.514176	Consumer Cyclical

Ticker	Industry	MarketCap	Small-cap	Large-cap
SU.TO	Oil & Gas Integrated	7.624041e+10	False	True
ABBV	Drug Manufacturers - General	5.623203e+11	False	True
LOW	Home Improvement Retail	1.900601e+11	False	True
AUST	Gold	3.310049e+07	True	False
TD	Banks - Diversified	2.023041e+11	False	True

AMZN	Internet Retail	3.483709e+12	False	True
AXP	Credit Services	3.551726e+11	False	True
GCT	Software - Infrastructure	1.952602e+09	True	False
BK	Banks - Diversified	1.104670e+11	False	True
EXE.TO	Medical Care Facilities	1.722458e+09	True	False
CMCSA	Telecom Services	1.376969e+11	False	True
SHOP	Software - Application	2.885765e+11	False	True
COST	Discount Stores	5.661248e+11	False	True
CSCO	Communication Equipment	4.247753e+11	False	True
GM	Auto Manufacturers	9.780647e+10	False	True
GOOG	Internet Content & Information	5.399775e+12	False	True
AIM.TO	Specialty Chemicals	2.519520e+08	True	False
FTG.TO	Aerospace & Defense	2.857180e+08	True	False
ORCL	Software - Infrastructure	8.044502e+11	False	True
SAP.TO	Packaged Foods	1.624666e+10	False	True
DUOL	Software - Application	1.236384e+10	False	True
PEP	Beverages - Non-Alcoholic	2.845414e+11	False	True
SLB	Oil & Gas Equipment & Services	7.564953e+10	False	True
SPG	REIT - Retail	9.820394e+10	False	True
SHOP.TO	Software - Application	2.905974e+11	False	True
RY.TO	Banks - Diversified	3.048237e+11	False	True
TD.TO	Banks - Diversified	2.026495e+11	False	True
AW.TO	Restaurants	8.977570e+08	True	False

Ticker	Earnings Date
SU.TO	2026-02-04
ABBV	2025-10-31
LOW	2025-11-19
AUST	None
TD	2025-12-04
AMZN	2025-10-30
AXP	2026-01-23
GCT	2025-11-06
BK	2026-01-13
EXE.TO	2025-11-11
CMCSA	2026-01-29
SHOP	2025-11-04
COST	2025-12-11
CSCO	2026-02-11
GM	2026-01-27
GOOG	2025-10-29
AIM.TO	2025-11-12
FTG.TO	2025-10-08
ORCL	2025-12-08
SAP.TO	2026-02-05
DUOL	2025-11-05
PEP	2025-10-09

```
SLB         2026-01-16
SPG         2025-11-03
SHOP.TO     2025-11-04
RY.TO       2025-12-03
TD.TO       2025-12-04
AW.TO       2026-03-19

Benchmark Std Dev: 0.8034836577141855
Benchmark Volatility: 1.48885560758792
```

[26]:
```python
## === 3. Historical Stock Returns vs TSX/S&P 500 Benchmark ===

periods = [5, 21, 63, 252]

# Prices aligned with the benchmark dates
px = stocks[filtered_tickers].loc[stocks.index.intersection(bench.index)].
 ↪dropna(how="all", axis=1)

def trailing_ret(series, d):
    s = series.dropna()
    if len(s) <= 1: return np.nan
    if len(s) <= d: return (s.iloc[-1] / s.iloc[0] - 1) * 100
    return (s.iloc[-1] / s.iloc[-(d+1)] - 1) * 100

def calc_avg_weekly(series):
    """Calculates the mean weekly return (%) over the available period."""
    # Resample to weekly (Friday close)
    w_prices = series.dropna().resample("W-FRI").last()
    # Calculate % change
    w_rets = w_prices.pct_change().dropna()
    return w_rets.mean() * 100

# Add "Avg Weekly (%)" returns to the column list
stock_returns = pd.DataFrame(index=px.columns,
                             columns=[f"{d}d" for d in periods] + ["Avg Weekly␣
 ↪(%)"])

for t in px.columns:
    for d in periods:
        stock_returns.loc[t, f"{d}d"] = trailing_ret(px[t], d)

    # Calculate and store the average weekly return for the stock
    stock_returns.loc[t, "Avg Weekly (%)"] = calc_avg_weekly(px[t])

# Returns table for the benchmark
bench_returns = pd.DataFrame(index=[f"{d}d" for d in periods],
                             columns=["TSX/S&P Benchmark (%)"])
```

```python
for d in periods:
    bench_returns.loc[f"{d}d"] = trailing_ret(bench, d)

bench_returns.loc["Avg Weekly (%)"] = calc_avg_weekly(bench)

# Formatting (round to 2 decimal places)
stock_returns = stock_returns.astype(float).round(2)
bench_returns = bench_returns.astype(float).round(2)

display(stock_returns)
display(bench_returns)
```

Ticker	5d	21d	63d	252d	Avg Weekly (%)
SU.TO	1.26	2.18	12.96	16.10	0.35
ABBV	0.26	7.33	19.47	15.66	0.42
LOW	-2.72	-1.64	14.51	-1.29	0.25
AUST	-8.21	8.81	37.69	79.00	1.95
TD	0.62	9.50	9.48	27.45	0.27
AMZN	-2.40	-4.07	2.33	19.17	0.54
AXP	0.39	4.69	9.92	30.56	0.72
GCT	-2.23	12.71	45.60	69.68	1.16
BK	0.26	3.44	21.89	54.49	0.86
EXE.TO	7.46	16.22	9.49	61.37	0.87
CMCSA	0.00	-6.20	-10.57	-20.92	-0.30
SHOP	-5.17	5.27	31.78	98.09	1.09
COST	-2.80	-2.97	-6.72	2.73	0.41
CSCO	-0.04	-2.46	-1.78	33.91	0.38
GM	3.80	4.50	24.33	25.61	0.66
GOOG	-3.37	15.06	40.09	51.96	0.72
AIM.TO	-2.80	1.30	16.42	19.08	0.07
FTG.TO	-8.75	-2.61	-2.69	102.80	1.27
ORCL	-13.83	17.66	32.86	71.86	1.29
SAP.TO	2.21	-1.66	24.94	16.74	0.29
DUOL	6.28	1.86	-17.53	31.39	0.90
PEP	-0.61	-4.63	9.32	-20.00	-0.17
SLB	3.51	-3.76	3.78	-15.23	-0.32
SPG	2.60	3.62	15.72	13.08	0.35
SHOP.TO	-4.52	6.51	34.53	102.40	1.13
RY.TO	-0.39	1.96	15.25	22.71	0.48
TD.TO	1.41	10.81	11.71	30.32	0.31
AW.TO	1.95	-1.15	3.31	-6.38	-0.09

	TSX/S&P Benchmark (%)
5d	-0.05
21d	4.86
63d	11.37
252d	25.41

```
Avg Weekly (%)                  0.40
```

[27]:
```python
# === 4. Scoring & Normalization ===

# 1. Merge Dataframes
# We need 'Avg Weekly (%)' from stock_returns combined with Beta/Idio/Cap from filtered
df_score = filtered.join(stock_returns["Avg Weekly (%)"], how="inner")

# 2. Get the target return for the benchmark
target_return = bench_returns.loc["Avg Weekly (%)", "TSX/S&P Benchmark (%)"]
print(f"Target Benchmark Weekly Return: {target_return}%")

# A. Beta Score (40%)
# Logic: Minimize the difference (gap) between each Stock's Beta and 1.0
df_score["Beta_Gap"] = (df_score["Beta"] - 1.0).abs()
# Rank: Ascending = False, means the Smallest Gap gets the Highest Score
df_score["Rank_Beta"] = df_score["Beta_Gap"].rank(ascending=False)

# B. Return Score (20%)
# Logic: Minimize the distance (gap) between Stock Return and Benchmark Return
df_score["Return_Gap"] = (df_score["Avg Weekly (%)"] - target_return).abs()
# Rank: Ascending = False means the Smallest Gap gets the Highest Score
df_score["Rank_Return"] = df_score["Return_Gap"].rank(ascending=False)

# C. Idiosyncratic Volatility Score (20%)
# Logic: Lower is better
# Rank: Ascending = False means the Lowest Volatility gets the Highest Score
df_score["Rank_Idio"] = df_score["IdioVol %"].rank(ascending=False)

# D. Market Cap Score (20%)
# Logic: Higher is better (liquidity & index drivers)
# Rank: Ascending = True means the Highest Cap gets the Highest Score
df_score["Rank_Cap"] = df_score["MarketCap"].rank(ascending=True)

# Normalize ranks to a 0-100 scale relative to the number of stocks
n_stocks = len(df_score)

# Weights: Beta = 4, Others = 2 (Total = 10 parts)
# Formula: (4*Beta + 2*Cap + 2*Return + 2*Idio) / 10
df_score["Final_Score"] = (
    (0.40 * df_score["Rank_Beta"]) +
    (0.20 * df_score["Rank_Return"]) +
    (0.20 * df_score["Rank_Idio"]) +
    (0.20 * df_score["Rank_Cap"])
)
```

```
# Sort by the Final Score
df_final = df_score.sort_values("Final_Score", ascending=False)

# --- Selection & Constraints ---

# 1. Initial Attempt: Pick Top 25 Stocks
top_25 = df_final.head(25).copy()

# 2. Check if there is a Small Cap
has_small = top_25["Small-cap"].any()

if not has_small:
    #No Small Cap in Top 25. So, we swap the lowest-ranked large cap with the␣
 ↪best-scoring small cap

    # Identify candidates: Stocks NOT in the top 25 that are small caps
    # We use iloc[25:] to look at stocks ranked 26th and below
    remaining_stocks = df_final.iloc[25:]

    small_cap_candidates = remaining_stocks[remaining_stocks["Small-cap"] ==␣
 ↪True]

    if not small_cap_candidates.empty:
        # A. Find the best available small cap (i.e. the first one, since␣
 ↪df_final is sorted)
        best_small_cap = small_cap_candidates.iloc[[0]]

        # B. Drop the lowest-ranked stock from our top 25 list (i.e. the last␣
 ↪one)
        top_25 = top_25.iloc[:-1]

        # C. Add the best small cap
        top_25 = pd.concat([top_25, best_small_cap])

        print(f"Replaced rank 25 with Small Cap '{best_small_cap.index[0]}'.")

# 3. Finalized formatting
cols_to_show = ["Beta", "Beta_Gap", "Avg Weekly (%)", "Return_Gap", "IdioVol␣
 ↪%", "MarketCap", "Final_Score", "Small-cap"]

display(top_25[cols_to_show])

print(f"# of Stocks: {len(top_25)}")
print(f"Large Cap?: {top_25['Large-cap'].any()}")
print(f"Small Cap?: {top_25['Small-cap'].any()}")
```

```
Target Benchmark Weekly Return: 0.4%
```

Ticker	Beta	Beta_Gap	Avg Weekly (%)	Return_Gap	IdioVol %	\
CSCO	1.099993	0.099993	0.38	0.02	1.020437	
RY.TO	0.833934	0.166066	0.48	0.08	0.753634	
COST	0.664741	0.335259	0.41	0.01	1.241291	
SU.TO	1.056839	0.056839	0.35	0.05	1.478485	
GOOG	1.143800	0.143800	0.72	0.32	1.776015	
LOW	0.853388	0.146612	0.25	0.15	1.359572	
SPG	1.312371	0.312371	0.35	0.05	1.231087	
TD	0.622004	0.377996	0.27	0.13	0.957418	
BK	1.184528	0.184528	0.86	0.46	1.059563	
TD.TO	0.555360	0.444640	0.31	0.09	0.960195	
ABBV	0.595512	0.404488	0.42	0.02	1.752781	
GM	1.087040	0.087040	0.66	0.26	2.152192	
SAP.TO	0.693954	0.306046	0.29	0.11	1.288071	
AMZN	1.533522	0.533522	0.54	0.14	1.664119	
CMCSA	0.775712	0.224288	-0.30	0.70	1.652905	
AXP	1.733626	0.733626	0.72	0.32	1.244467	
PEP	0.311949	0.688051	-0.17	0.57	1.398343	
EXE.TO	0.423429	0.576571	0.87	0.47	1.607128	
AIM.TO	0.468468	0.531532	0.07	0.33	1.777160	
AW.TO	0.375006	0.624994	-0.09	0.49	1.514176	
FTG.TO	0.639513	0.360487	1.27	0.87	2.266123	
SLB	1.557192	0.557192	-0.32	0.72	1.842577	
ORCL	1.991268	0.991268	1.29	0.89	3.312056	
SHOP	2.949644	1.949644	1.09	0.69	2.941992	
SHOP.TO	2.900805	1.900805	1.13	0.73	2.984766	

Ticker	MarketCap	Final_Score	Small-cap
CSCO	4.247753e+11	25.2	False
RY.TO	3.048237e+11	23.6	False
COST	5.661248e+11	22.2	False
SU.TO	7.624041e+10	21.5	False
GOOG	5.399775e+12	20.9	False
LOW	1.900601e+11	20.0	False
SPG	9.820394e+10	19.5	False
TD	2.023041e+11	19.0	False
BK	1.104670e+11	18.8	False
TD.TO	2.026495e+11	18.6	False
ABBV	5.623203e+11	18.6	False
GM	9.780647e+10	18.0	False
SAP.TO	1.624666e+10	17.8	False
AMZN	3.483709e+12	16.6	False
CMCSA	1.376969e+11	15.4	False
AXP	3.551726e+11	14.1	False
PEP	2.845414e+11	12.2	False
EXE.TO	1.722458e+09	10.4	True

```
AIM.TO    2.519520e+08          10.4       True
AW.TO     8.977570e+08           9.8       True
FTG.TO    2.857180e+08           9.4       True
SLB       7.564953e+10           9.2      False
ORCL      8.044502e+11           8.0      False
SHOP      2.885765e+11           7.0      False
SHOP.TO   2.905974e+11           6.8      False

# of Stocks: 25
Large Cap?: True
Small Cap?: True
```

[28]:
```python
# === 5. Portfolio Optimization ===

df_score = df_score[~df_score.index.duplicated(keep="first")]
stock_returns = stock_returns[~stock_returns.index.duplicated(keep="first")]
meta = meta[~meta.index.duplicated(keep="first")]

def optimize_weights_for_top25(rets, stock_returns, bench_returns, top25, meta,␣
 ↪df_score, lam=0.01):

    # get list of the selected tickers
    tickers = list(top25.index)

    # get final score values for these tickers
    score_vec = df_score.loc[tickers, "Final_Score"].astype(float)

    # normalize scores so they sum to one
    score_norm = score_vec / score_vec.sum()

    # get each stock's weekly return
    weekly_stock_ret = stock_returns.loc[tickers, "Avg Weekly (%)"].
 ↪astype(float) / 100

    # get benchmark weekly return
    weekly_bench_ret = float(bench_returns.loc["Avg Weekly (%)", "TSX/S&P␣
 ↪Benchmark (%)"]) / 100

    # objective combines strong return matching and score alignment
    def objective(weights):
        #compute portfolio weekly return
        port_ret = np.dot(weights, weekly_stock_ret)

        # we square the difference between portfolio return and benchmark return
        # then multiply by 1,000,000 so that even tiny differences become␣
 ↪extremely important to the optimizer
        # which forecs the portfolio return to match the benchmark as close as␣
 ↪it can
```

```
        track_err = 1_000_000 * (port_ret - weekly_bench_ret)**2

        # the score alignment encourages higher scoring stocks to have higher␣
↪weights
        score_err = np.sum((weights - score_norm)**2)
        return track_err + lam * score_err

    # weights must sum to one
    def sum_to_one(weights):
        return np.sum(weights) - 1

    # any one sector must not exceed 40%
    def sector_check(weights):
        sector_totals = {}
        for i, t in enumerate(tickers):
            sec = meta.loc[t, "Sector"]
            if sec not in sector_totals:
                sector_totals[sec] = 0
            sector_totals[sec] += weights[i]
        return min(0.40 - w for w in sector_totals.values())

    # ensure at least 1 large cap
    def large_cap_check(weights):
        total = 0
        for i, t in enumerate(tickers):
            if meta.loc[t, "MarketCap"] > 10_000_000_000:
                total += weights[i]
        return total - (1 / len(tickers))

    # ensure at least 1 small cap
    def small_cap_check(weights):
        total = 0
        for i, t in enumerate(tickers):
            if meta.loc[t, "MarketCap"] < 2_000_000_000:
                total += weights[i]
        return total - (1 / len(tickers))

    # list of constraints
    constraints = [
        {'type': 'eq',  'fun': sum_to_one},
        {'type': 'ineq','fun': sector_check},
        {'type': 'ineq','fun': large_cap_check},
        {'type': 'ineq','fun': small_cap_check},
    ]

    # weight bounds from the assignment
    min_weight = 1 / (2 * len(tickers))
```

```
    bounds = [(min_weight, 0.15)] * len(tickers)

    # start the near score distribution
    starting_guess = np.array(score_norm)

    # run the portfolio optimizer
    result = minimize(
        fun = objective,
        x0 = starting_guess,
        method = "SLSQP",
        bounds = bounds,
        constraints = constraints,
        options = {"maxiter": 5000, "disp": True}
    )

    # normalize weights
    weights = result.x
    weights = weights / np.sum(weights)

    # compute the final return and difference
    final_port_ret = np.dot(weights, weekly_stock_ret)
    difference = abs(final_port_ret - weekly_bench_ret)

    print("\nOptimization:")
    print("Benchmark Weekly Return:", weekly_bench_ret)
    print("Portfolio Weekly Return:", final_port_ret)
    print("Absolute Difference:", difference)
    print("Sum of Weights:", np.sum(weights))

    # display with 4 decimals
    return pd.Series(np.round(weights, 4), index=tickers)


opt_weights = optimize_weights_for_top25(
    rets = rets,
    stock_returns = stock_returns,
    bench_returns = bench_returns,
    top25 = top_25,
    meta = meta,
    df_score = df_score
)

display(opt_weights)
```

```
Optimization terminated successfully    (Exit mode 0)
            Current function value: 1.1220892192413767e-05
            Iterations: 3
            Function evaluations: 79
```

```
            Gradient evaluations: 3

Optimization:
Benchmark Weekly Return: 0.004
Portfolio Weekly Return: 0.004000027165186366
Absolute Difference: 2.7165186365987748e-08
Sum of Weights: 1.0

/opt/anaconda3/lib/python3.13/site-packages/scipy/optimize/_slsqp_py.py:435:
RuntimeWarning: Values in x were outside bounds during a minimize step, clipping
to bounds
  fx = wrapped_fun(x)

CSCO       0.0584
RY.TO      0.0548
COST       0.0517
SU.TO      0.0502
GOOG       0.0486
LOW        0.0469
SPG        0.0457
TD         0.0447
BK         0.0439
TD.TO      0.0438
ABBV       0.0437
GM         0.0422
SAP.TO     0.0420
AMZN       0.0392
CMCSA      0.0533
AXP        0.0335
PEP        0.0461
EXE.TO     0.0252
AIM.TO     0.0257
AW.TO      0.0382
FTG.TO     0.0227
SLB        0.0395
ORCL       0.0200
SHOP       0.0200
SHOP.TO    0.0200
dtype: float64
```

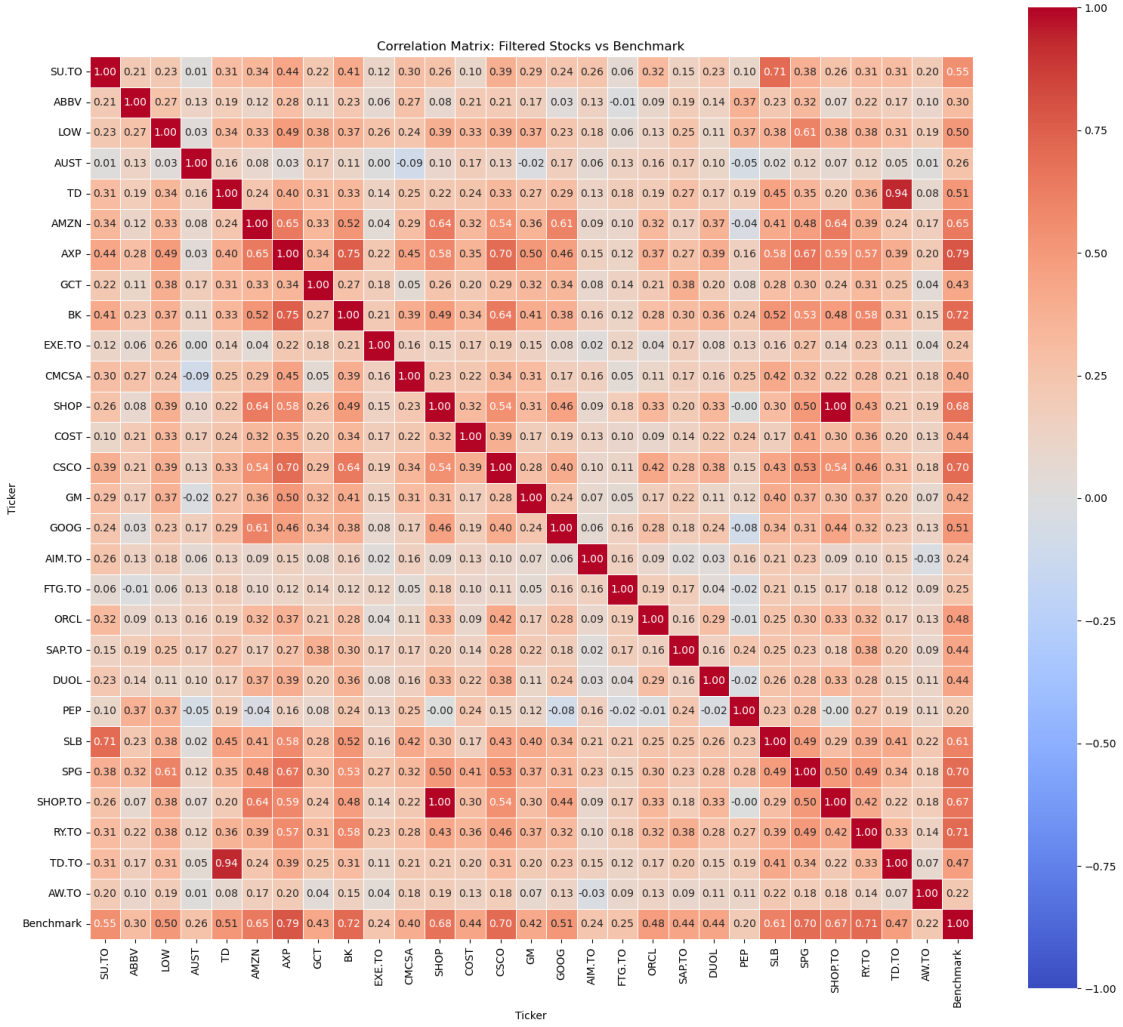
[29]:
```python
# === 6. Visualizing the Data ===

# Temporary placeholder portfolio for Section 6
portfolio_df = pd.DataFrame({
    "Ticker": opt_weights.index,
})
portfolio_df["Value (CAD)"] = 1.0
portfolio_df["Weight (%)"] = (1 / len(portfolio_df)) * 100
```

```
# ================================
# Correlation & Covariance Heatmaps
# ================================

# 1. Prepare data values
heatmap_data = rets[filtered_tickers].copy()
heatmap_data["Benchmark"] = bench_ret # Add benchmark for comparison

# 2. Compute the matrices
corr_matrix = heatmap_data.corr()
cov_matrix = heatmap_data.cov() * (100**2) # Scale up covariance for␣
 ↪readability (like in metrics)

# 3. Plot the Correlation Heatmap
plt.figure(figsize=(16, 14))
sns.heatmap(corr_matrix, annot=True, fmt=".2f", cmap="coolwarm", vmin=-1,␣
 ↪vmax=1, square=True, linewidths=0.5)
plt.title("Correlation Matrix: Filtered Stocks vs Benchmark")
plt.tight_layout()
plt.show()

# --- Explanation of Heatmap #1 ---
print("""The Correlation heatmap ensures diversification by revealing if stocks␣
 ↪are redundant (moving identically) or independent.
It is normalized on a scale of -1 to +1. By removing volatility, it tells us if␣
 ↪stocks move in sync.
If two stocks have a high correlation (close to 1), they are mathematically␣
 ↪redundant. This information helps diversify a portfolio""")

# 4. Plot the Covariance Heatmap
plt.figure(figsize=(16, 14))
sns.heatmap(cov_matrix, annot=True, fmt=".2f", cmap="viridis", square=True,␣
 ↪linewidths=0.5)
plt.title("Covariance Matrix (Scaled)")
plt.tight_layout()
plt.show()

# --- Explanation of Heatmap #2 ---
print("""The Covariance heatmap measures actual risk magnitude, identifying how␣
 ↪volatile individual stocks are and how much they swing together.
It combines correlation plus volatility, which tells us how much actual risk is␣
 ↪being generated.
A high covariance means two stocks not only move together but do so with␣
 ↪violent swings""")
```

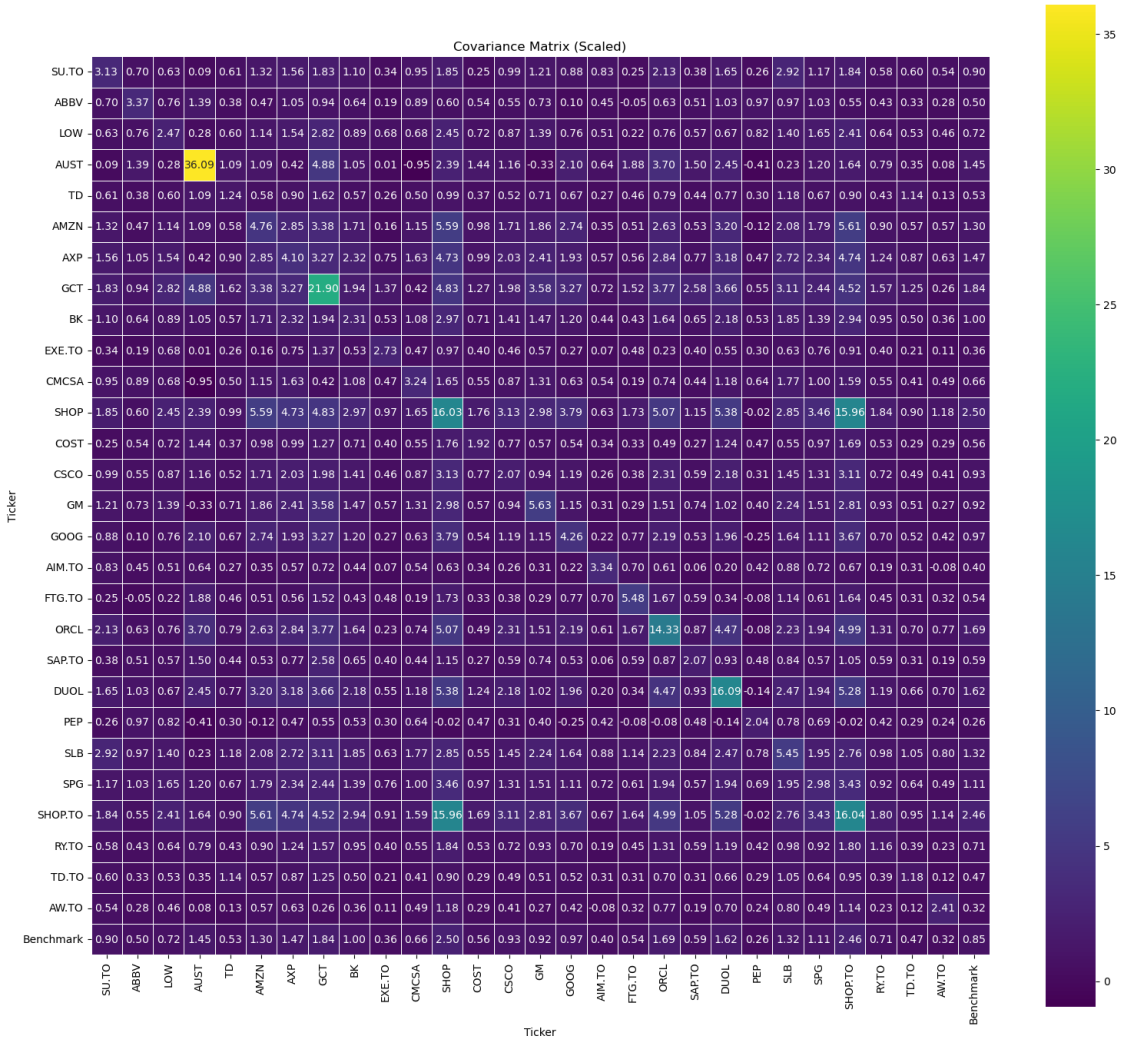
```
# ==================================
# Time-Series Graphs: Beta & Std Dev
# ==================================

beta_series = metrics["Beta"]
std_series = metrics["StdDev (%)"]
benchmark_beta = 1.0
benchmark_std = bench_std

# --- Plot the Beta Graph ---
plt.figure(figsize=(10, 5))
plt.plot(beta_series.index, beta_series.values, marker='o', color='black')
plt.axhline(y=benchmark_beta, color='red', linestyle='--', label='Benchmark␣
 ↪Beta')
plt.title("Stocks' Beta Values Relative to Benchmark")
plt.xlabel("Ticker")
plt.ylabel("Beta")
plt.xticks(rotation=90)
plt.grid(True)
plt.tight_layout()
plt.show()

# --- Explanation of the Time-Series Graph #1 ---
print("""Through the beta chart, we can see a wide range of market sensitivity.␣
 ↪While certain stocks sit well above the benchmark beta of 1.00, others fall␣
 ↪noticeably below it. The main benefit of this graph is that it showcases the␣
 ↪assets' differing market movements, and it helps identify positions that␣
 ↪carry higher systematic risk versus those that behave more defensively.

Because these values are spread out, the portfolio ends up holding stocks with␣
 ↪different levels of market exposure, and, as a result, the overall risk is␣
 ↪dispersed rather than concentrated in one category. This naturally supports␣
 ↪diversification across both high-beta and low-beta positions.""")

# --- Plot the Standard Deviation Graph ---
plt.figure(figsize=(10, 5))
plt.plot(std_series.index, std_series.values, marker='o', color='black')
plt.axhline(y=benchmark_std, color='red', linestyle='--', label='Benchmark Std␣
 ↪Dev')
plt.title("Average Daily Std Dev (%) by Stock")
plt.xlabel("Ticker")
plt.ylabel("Standard Deviation (%)")
plt.xticks(rotation=90)
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
# --- Explanation of the Time-Series Graph #2 ---
print("""In the standard deviation plot, an important benefit comes from seeing␣
 ↪how the overall volatility of each stock compares relative to the rest of␣
 ↪the portfolio. As a result, the graph helps us identify which positions␣
 ↪contribute to more short-term risk and which ones are more stable.

Furthermore, by viewing all volatility levels together, the chart makes it␣
 ↪easier to understand how spread out the risk is across the holdings. This␣
 ↪allows us to judge whether the portfolio relies too heavily on a single␣
 ↪volatility profile or whether it gains diversification by balancing varying␣
 ↪levels of fluctuation.""")

# ========================================
# Pie Charts: Portfolio Sectors & Weightings
# ========================================

# uses final portfolio and meta sector info
# temporary placeholder for Section 6 (full Portfolio_Final created in Section␣
 ↪7)
portfolio_df = pd.DataFrame({
    "Ticker": opt_weights.index,
    "Price": stocks[opt_weights.index].iloc[-1].values,
    "Currency": ["CAD" if t.endswith(".TO") else "USD" for t in opt_weights.
 ↪index],
})
portfolio_df["Value (CAD)"] = 1.0
portfolio_df["Weight (%)"] = opt_weights.values * 100

sector_series = meta["Sector"]

# sector breakdown for the portfolio
sectors = portfolio_df["Ticker"].map(sector_series)
sector_totals = portfolio_df.groupby(sectors)["Value (CAD)"].sum()

# --- Plot the Portfolio's Sectors Chart ---
plt.figure(figsize=(8, 8))
plt.pie(sector_totals, labels=sector_totals.index, autopct="%1.1f%%")
plt.title("Portfolio's Sector Allocations")
plt.tight_layout()
plt.show()

# --- Plot the Stock Weightings Chart ---
plt.figure(figsize=(10, 10))
plt.pie(portfolio_df["Weight (%)"], labels=portfolio_df["Ticker"], autopct="%1.
 ↪1f%%")
plt.title("Stock Weighting Distribution")
plt.tight_layout()
```

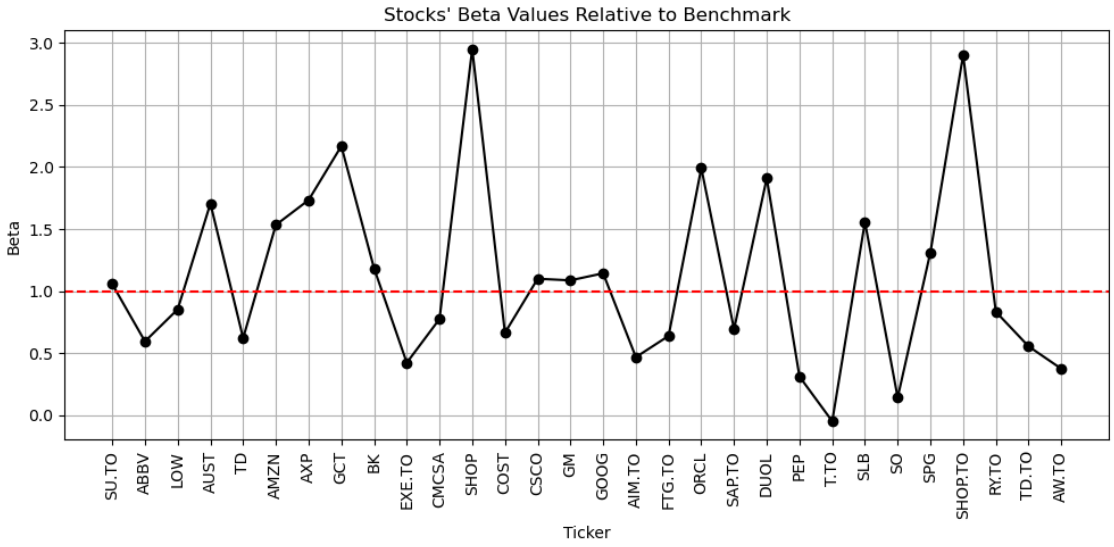
```
plt.show()

# get sector totals from the final selected stocks
benchmark_totals = portfolio_df.groupby(sectors)["Value (CAD)"].sum()

# convert totals to percentages for the pie chart
benchmark_sectors = (benchmark_totals / benchmark_totals.sum()).to_dict()

# --- Plot the Benchmark's Sectors Chart ---
plt.figure(figsize=(10, 10))
plt.pie(
    list(benchmark_sectors.values()),
    labels=list(benchmark_sectors.keys()),
    autopct="%1.1f%%"
)
plt.title("Average % per Sector in the S&P 500/TSX Benchmark")
plt.tight_layout()
plt.show()

# --- Explanation of the Pie Charts ---
print("""The sector allocation chart focuses on how the portfolio's value is␣
 ↪spread across different industries. By looking at the distribution, we can␣
 ↪see which areas the portfolio naturally leans toward and whether any sector␣
 ↪carries a noticeably larger share. This makes it easier to understand where␣
 ↪the portfolio is most exposed and where diversification is helping reduce␣
 ↪concentration risk.

The weight distribution chart focuses on the individual tickers rather than the␣
 ↪sectors. It shows how much each position contributes to the whole portfolio,␣
 ↪allowing us to confirm that no single stock dominates the allocation. This␣
 ↪is especially useful for checking that the final weights line up with the␣
 ↪optimization output. The benchmark sector mix is generated using the same␣
 ↪set of selected stocks.

Since it shows the underlying composition of the portfolio, it provides a␣
 ↪simple reference point to compare against the sector allocation chart.␣
 ↪Together, these visuals help clarify both the top-down (sector) and␣
 ↪bottom-up (ticker) structure of the portfolio.""")
```



The Correlation heatmap ensures diversification by revealing if stocks are redundant (moving identically) or independent.
It is normalized on a scale of -1 to +1. By removing volatility, it tells us if stocks move in sync.
If two stocks have a high correlation (close to 1), they are mathematically redundant. This information helps diversify a portfolio

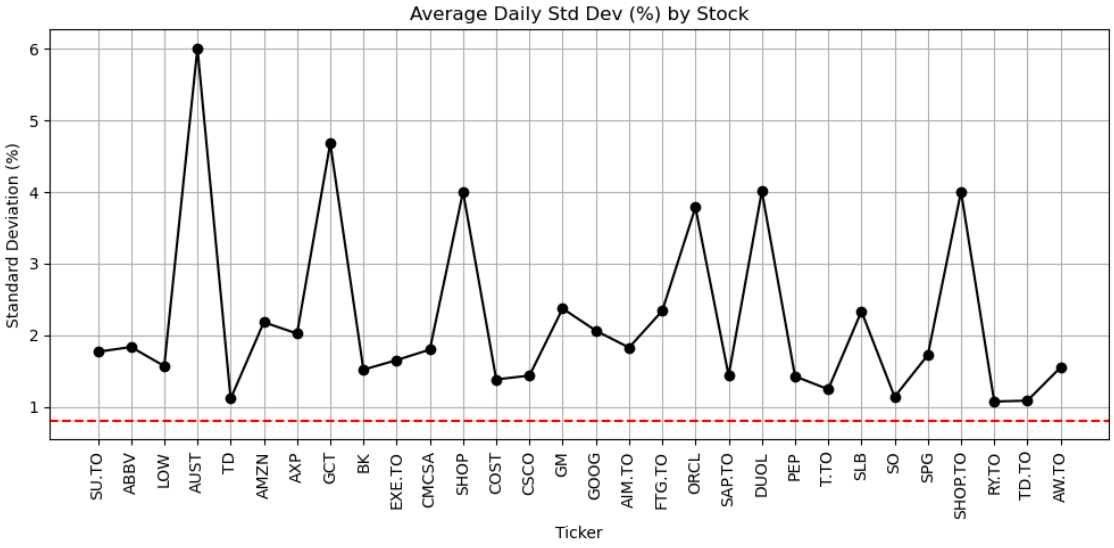


The Covariance heatmap measures actual risk magnitude, identifying how volatile individual stocks are and how much they swing together.
It combines correlation plus volatility, which tells us how much actual risk is being generated.
A high covariance means two stocks not only move together but do so with violent swings



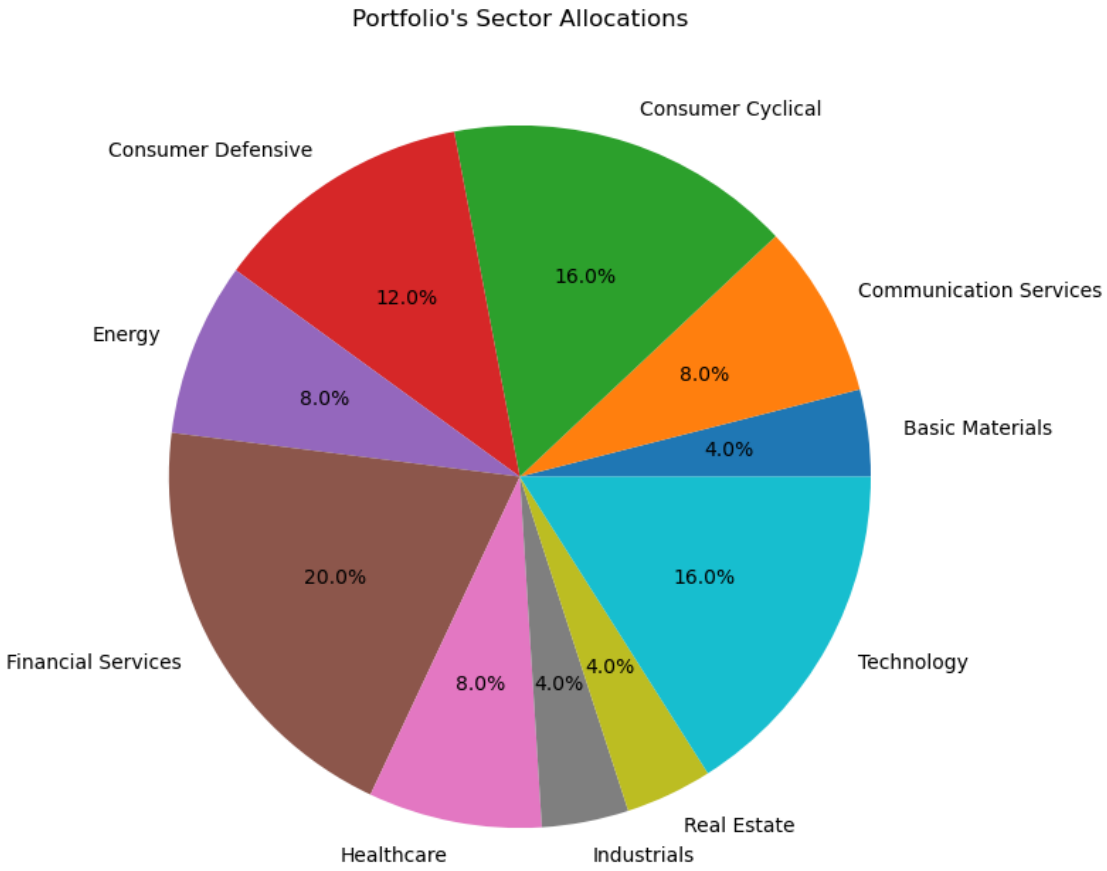
Through the beta chart, we can see a wide range of market sensitivity. While certain stocks sit well above the benchmark beta of 1.00, others fall noticeably below it. The main benefit of this graph is that it showcases the assets' differing market movements, and it helps identify positions that carry higher systematic risk versus those that behave more defensively.

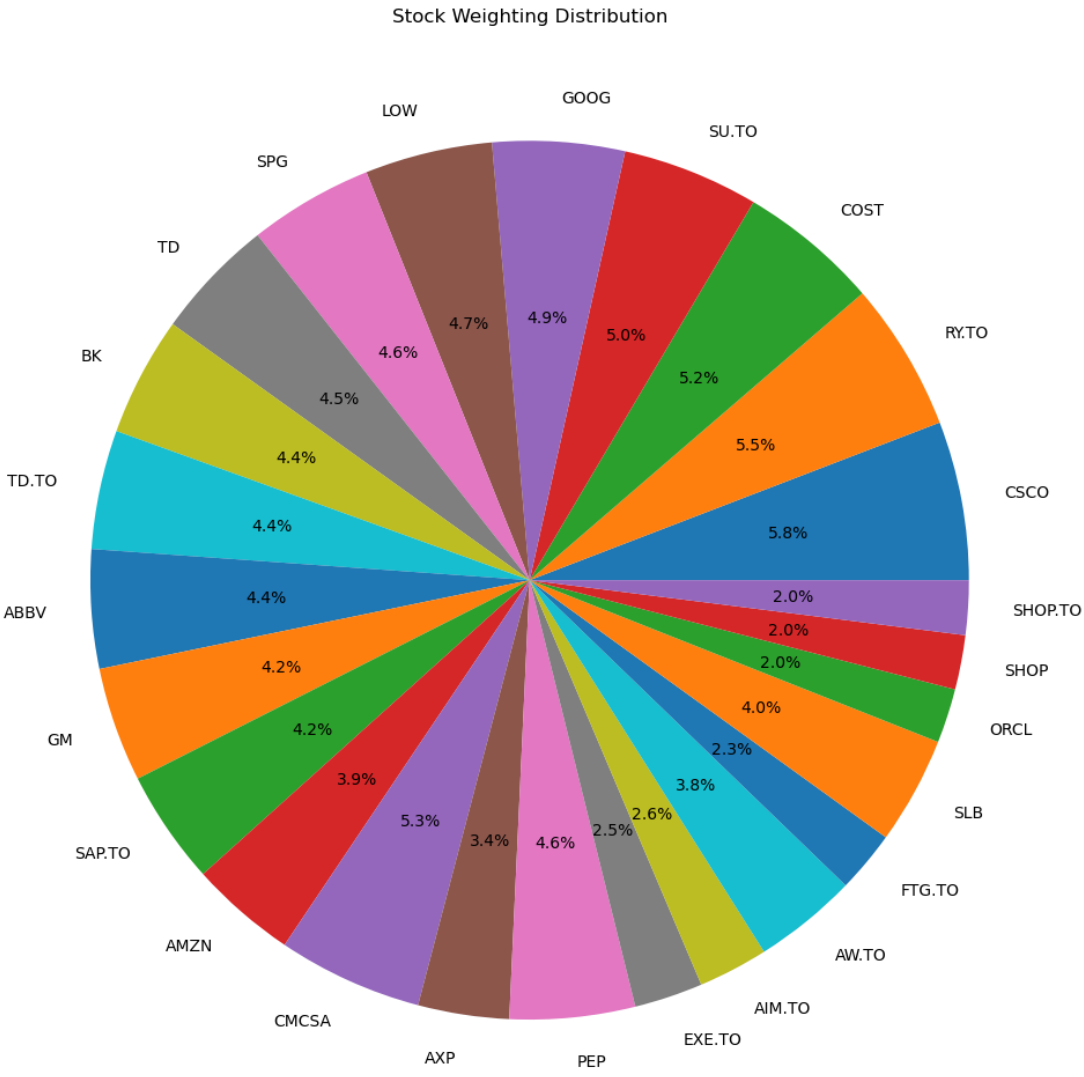
Because these values are spread out, the portfolio ends up holding stocks with different levels of market exposure, and, as a result, the overall risk is dispersed rather than concentrated in one category. This naturally supports diversification across both high-beta and low-beta positions.

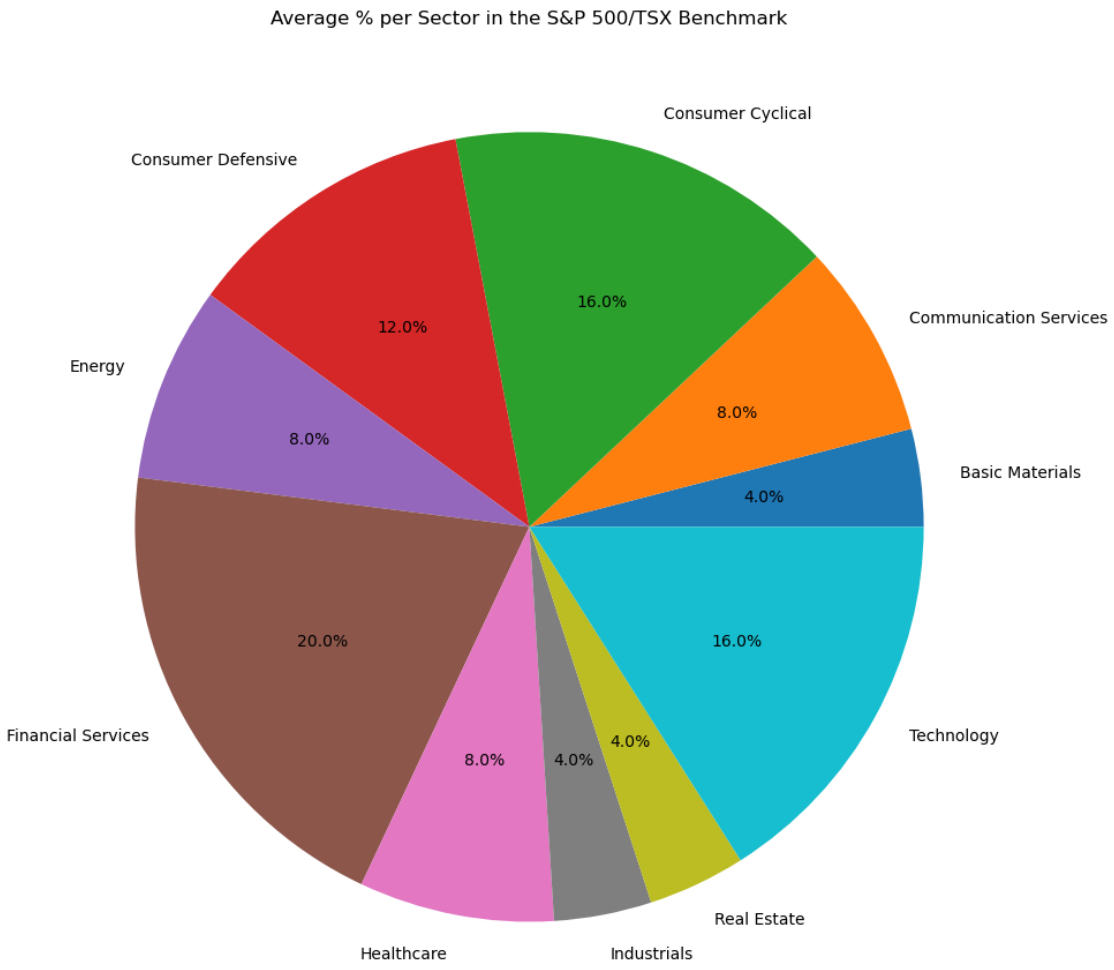


In the standard deviation plot, an important benefit comes from seeing how the overall volatility of each stock compares relative to the rest of the portfolio. As a result, the graph helps us identify which positions contribute to more short-term risk and which ones are more stable.

Furthermore, by viewing all volatility levels together, the chart makes it easier to understand how spread out the risk is across the holdings. This allows us to judge whether the portfolio relies too heavily on a single volatility profile or whether it gains diversification by balancing varying levels of fluctuation.




Stock Weighting Distribution
GOOG
SU.TO
COST
RY.TO
CSCO
SHOP.TO
SHOP
ORCL
SLB
FTG.TO
AW.TO
AIM.TO
EXE.TO
PEP
AXP
CMCSA
AMZN
SAP.TO
GM
ABBV
TD.TO
BK
TD
SPG
LOW
4.9%
5.0%
5.2%
5.5%
5.8%
2.0%
2.0%
2.0%
4.0%
2.3%
3.8%
2.6%
2.5%
4.6%
3.4%
5.3%
3.9%
4.2%
4.2%
4.4%
4.4%
4.4%
4.5%
4.6%
4.7%



The sector allocation chart focuses on how the portfolio's value is spread across different industries. By looking at the distribution, we can see which areas the portfolio naturally leans toward and whether any sector carries a noticeably larger share. This makes it easier to understand where the portfolio is most exposed and where diversification is helping reduce concentration risk.

The weight distribution chart focuses on the individual tickers rather than the sectors. It shows how much each position contributes to the whole portfolio, allowing us to confirm that no single stock dominates the allocation. This is especially useful for checking that the final weights line up with the optimization output. The benchmark sector mix is generated using the same set of selected stocks.

Since it shows the underlying composition of the portfolio, it provides a simple reference point to compare against the sector allocation chart. Together, these visuals help clarify both the top-down (sector) and bottom-up (ticker) structure

of the portfolio.

[30]:
```
# === 7. Output the Portfolio's Data Frame & Export the .csv file ===

portfolio_value = 1_000_000  # CAD budget before fees

tickers_sel = opt_weights.index
prices_local = stocks[tickers_sel].iloc[-1]

currency = pd.Series(
    ["CAD" if t.endswith(".TO") else "USD" for t in tickers_sel],
    index = tickers_sel
)

# Convert local prices to CAD (used for value & share calculations)
prices_cad = pd.Series(
    [p if currency[t] == "CAD" else p / usd_to_cad for t, p in prices_local.
 ↪items()],
    index = tickers_sel
)

# --- STEP #1: Allocate the full CAD budget based on stock weightings BEFORE␣
 ↪fees ---
pre_fee_allocation_cad = opt_weights * portfolio_value

# --- STEP #2: Convert allocations into fractional shares ---
shares = pre_fee_allocation_cad / prices_cad

# --- STEP #3: Compute exact fees for each stock ---
# Fee rule: min(2.15 USD, 0.001 USD * shares)
fees_usd = np.minimum(2.15, 0.001 * shares)
fees_cad = fees_usd / usd_to_cad
total_fees_cad = fees_cad.sum()

# --- STEP #4: Actual investable CAD after fees ---
portfolio_net_cad = portfolio_value - total_fees_cad

# --- STEP #5: Recalculate the final # of shares using NET capital ---
final_allocation_cad = opt_weights * portfolio_net_cad
final_shares = final_allocation_cad / prices_cad

# --- STEP #6: Final CAD values and weights ---
values_cad = final_shares * prices_cad
total_value_cad = portfolio_net_cad

final_weights = round(((values_cad / total_value_cad) * 100),2)
```

```python
# --- Build Portfolio_Final (index 1..n) ---
Portfolio_Final = pd.DataFrame({
    "Ticker": tickers_sel,
    "Price": prices_local.values,
    "Currency": currency.values,
    "Shares": final_shares.values,
    "Value (CAD)": values_cad.values,
    "Weight (%)": final_weights.values
})
Portfolio_Final.index = range(1, len(Portfolio_Final) + 1)

print(f"Net Portfolio Value (CAD): ${total_value_cad:,.2f}")
print(f"Total Fees Deducted (CAD): ${total_fees_cad:,.2f}")
print(f"Total # of Shares: {final_shares.sum():,.2f}")
print(f"Sum of Stock Weightings: {final_weights.sum():.1f}%")

display(Portfolio_Final)
```

```
Net Portfolio Value (CAD): $999,979.49
Total Fees Deducted (CAD): $20.51
Total # of Shares: 20,761.87
Sum of Stock Weightings: 100.0%
```

	Ticker	Price	Currency	Shares	Value (CAD)	Weight (%)
1	CSCO	67.720001	USD	617.159376	58398.802476	5.84
2	RY.TO	204.699997	CAD	267.703357	54798.876296	5.48
3	COST	916.869995	USD	40.353780	51698.939863	5.17
4	SU.TO	58.470001	CAD	858.542322	50198.970622	5.02
5	GOOG	244.360001	USD	142.333652	48599.003431	4.86
6	LOW	253.320007	USD	132.496618	46899.038290	4.69
7	SPG	185.850006	USD	175.976656	45699.062897	4.57
8	TD	79.769997	USD	401.023101	44699.083402	4.47
9	BK	109.750000	USD	286.260505	43899.099807	4.39
10	TD.TO	111.110001	CAD	394.195857	43799.101857	4.38
11	ABBV	223.160004	USD	140.141421	43699.103908	4.37
12	GM	61.119999	USD	494.117863	42199.134666	4.22
13	SAP.TO	33.759998	CAD	1244.050381	41999.138767	4.20
14	AMZN	222.169998	USD	126.270557	39199.196183	3.92
15	CMCSA	31.490000	USD	1211.311925	53298.907055	5.33
16	AXP	342.309998	USD	70.036860	33499.313064	3.35
17	PEP	140.169998	USD	235.367936	46099.054694	4.61
18	EXE.TO	15.120000	CAD	1666.632503	25199.483260	2.52
19	AIM.TO	3.120000	CAD	8237.010881	25699.473008	2.57
20	AW.TO	37.730000	CAD	1012.436182	38199.216688	3.82
21	FTG.TO	11.580000	CAD	1960.236155	22699.534524	2.27
22	SLB	35.110001	USD	805.133134	39499.190031	3.95
23	ORCL	282.760010	USD	50.618987	19999.589889	2.00
24	SHOP	149.000000	USD	96.060572	19999.589889	2.00

```
25  SHOP.TO  207.460007      CAD    96.402146  19999.589889        2.00
```

[31]:
```
# --- Export CSV file (Ticker + Shares only) ---
Stocks_Final = Portfolio_Final[["Ticker", "Shares"]]
Stocks_Final.to_csv("Stocks_Group_10.csv", index=False, header=True)

from IPython.display import FileLink, HTML

print("Click below to download our Stocks_Final CSV:")
display(FileLink("Stocks_Group_10.csv"))
```

```
Click below to download our Stocks_Final CSV:
```

/Users/dhirkachroo/anaconda_projects/CFM 101/Stocks_Group_10.csv

---

Markdown Conclusion: According to our chosen strategy (Market Meet), our goal was to generate a portfolio return as close as possible to the benchmark average (above or below) of the S&P/TSX Composite and the S&P 500 indices.
Here is a step-by-step overview of the main code sections, as well as an explanation of our understanding:
1. Parse the Tickers.csv file and extract ticker data (market capitalization, sector, industry, daily volume) from Yahoo Finance (yfinance).
In this section, the code reads the TA-provided Tickers.csv file and downloads the year-to-date (YTD) daily price and volume data (between 2025-01-01 and 2025-11-21) for every ticker, as well as the S&P 500 and TSX Composite benchmarks. The script extracts each company's sector, industry, market capitalization, and upcoming earnings date using yfinance.Ticker().info. Market caps are normalized into CAD, ensuring U.S. tickers (such as AAPL at approximately USD 3.0T) are directly comparable to Canadian ones under the assignment thresholds ($<$ CAD 2B small-cap, $>$ CAD 10B large-cap). The average daily volume is calculated after removing any month with fewer than 18 trading days. This generates a complete and consistent dataset that later filtering and calculations rely on.
2. Calculate metrics: Beta, Relative Standard Deviation (StdDev), Weekly Volatility (Weekly Vol), Covariance
In this stage, the code computes all statistical metrics using the return series aligned to the benchmark. Daily standard deviation is converted into percentage units (e.g., many large-caps fall in the 1–3% daily volatility range, while higher-risk names may exceed 4–5%). Covariance values are scaled into percent-squared units, often ranging from 0.01 to 0.15 depending on the stock's co-movement with the benchmark. Beta is calculated as Cov/Var, typically producing values between 0.6–1.8 depending on the stock's sensitivity. Weekly volatility is computed by compounding daily returns into weekly returns, then taking the standard deviation of that series; stable names often show 3–6% weekly volatility, while higher-risk ones can exceed 10–12%. Correlation values usually cluster between 0.20 and 0.90.
3. Calculate historical stock & benchmark returns (5 days, 21d, 63d, 252d)

This section computes trailing percentage returns for each stock and the benchmark across 5-day, 21-day, 63-day, and 252-day windows. When the available price history is shorter than the required window, the script uses the earliest available price to compute a valid return. These returns typically range widely: 5-day values often fall between −10% and +12% for volatile tickers, while 63-day values often lie between −20% and +25%. The benchmark's returns are usually narrower, such as −3% to +4% over 21 days and −8% to +12% over 252 days. The resulting DataFrames present all values in percentage format for clear interpretation during subsequent scoring and selection steps.

4. Assign a weighted point value score per stock, for every category based on importance [Beta (40%), Market Cap (20%), Returns (20%), & Volatility (20%)]

In this part of the code, each stock receives a numerical score derived from its beta, market cap, volatility, and return on investment (ROI), each normalized and then weighted by the importance levels selected by the group. The 40% beta component favours values near 1.00, the 20% market cap component increases the score of larger firms (which may range from CAD 50B to CAD 3T), and the 20% volatility component lowers the score of stocks with higher daily or weekly variability (such as weekly volatility exceeding 8–10%). The 20% returns column focuses on how accurate a stock's weekly returns are relative to the 50/50 benchmark. The sum of these points is tallied into a single total score used to rank each filtered stock.

5. Optimize the large-cap portfolio (# of valid filtered stocks - 1), replace the lowest scoring large-cap stock with the highest scoring small-cap stock. Note: Check if there is a small-cap in the filtered list of valid stocks.

After all stocks are scored, the code selects the top-ranked large-cap stocks (market cap > CAD 10B) and checks whether any small-cap stocks (< CAD 2B) remain in the filtered set. If at least one small-cap is present, the script replaces the lowest-ranked large-cap with the highest-ranked small-cap to satisfy the assignment requirement. The optimizer then computes position weights subject to the constraints: no position above 15%, a minimum weight of 100/(2n)%, sector exposure below 40%, and total allocated amount near $1,000,000 CAD after applying the fee rule (2.15 USD or 0.001/share). Share quantities are then calculated from these weights and the last available price data.

6. Visualize the data (Heatmaps, Time-series graphs, Pie charts)

***In this section, the code produces visual summaries that make the structure of the filtered stocks and the final portfolio easy to understand. A correlation heatmap shows how closely each stock's returns move together and how strongly they relate to the benchmark. A covariance heatmap presents the differences in joint variability between stock pairs. Time-series plots display each stock's beta and volatility values so they can be compared directly across the filtered universe. Sector pie charts show how the portfolio's exposure is distributed across industries, and a separate pie chart shows the weight assigned to each stock.

7. Output the final stock portfolio data frame and export the Tickers + # of shares to a .csv file

As the final stage, this creates the Portfolio_Final DataFrame, which contains each stock's tickers, last traded price, currency, number of shares, total CAD value, and final optimized portfolio weight. The script verifies that the portfolio value stays near the 1,000,000 CAD budget after fees and that the weights sum exactly to 100%. Note: The fee for each stock is the smaller of a $2.15 USD flat charge or 0.001 USD per share purchased; the total transaction fee (CAD) is the sum of these amounts across all stocks in the portfolio.

It then exports the required Stocks_Final CSV file using only the ticker and # of shares for submission. These outputs summarize the final optimized portfolio using only the results produced by the earlier filtering, scoring, and weighting steps.