ChatGPT

Japanese Equity Portfolio Analysis Project

Overview and Objectives

This project involves building a **Python-based portfolio analysis** for a simulated Japanese equity portfolio, tailored as a showcase for a BlackRock Japan Portfolio Management internship. The goal is to **demonstrate investment analytics skills, portfolio-level thinking, and risk/return communication** using real market data. We will construct a Jupyter Notebook that is fully reproducible and well-documented, focusing on a portfolio of Japanese stocks/ETFs, and include performance evaluation, risk metrics, attribution analysis, and professional presentation of results. The **objectives** include:

- **Data Retrieval:** Use Python (e.g. `yfinance`) to fetch historical price data for 8–12 large-cap Japanese stocks or Japan-focused ETFs.
- **Portfolio Simulation:** Start with ¥10,000,000 initial capital, allocate equally across 10 assets, and simulate **monthly rebalancing** (at the start of each month) with **0.1% transaction cost per trade**.
- **Performance Metrics:** Calculate daily returns, cumulative and annualized returns, annualized volatility, Sharpe ratio (risk-free rate = 0%), maximum drawdown, and rolling 3-month volatility & Sharpe. Compare the portfolio’s performance against a **benchmark** (e.g., an index ETF like EWJ or a TOPIX tracker).
- **Attribution & Diagnostics:** Determine each asset’s contribution to returns by month (approximate **return attribution** using weight × return), identify top contributors/detractors each month, analyze rolling correlation with the benchmark, and rank assets by volatility.
- **Visualizations:** Produce clear charts including the **equity curve vs. benchmark**, a **drawdown chart**, **rolling volatility and Sharpe ratio** over time, and a **bar chart of monthly contributions** by asset.
- **Professional Presentation:** Deliver a polished Jupyter Notebook with modular, well-documented code (functions with docstrings and type hints), a dashboard-style summary of results (key charts on one page), and a concise **“PM-style” memo** of 3–5 bullet point takeaways highlighting performance and risks. Additionally, provide a folder structure and README for reproducibility and clarity.

Data and Tools

Data: We will use daily historical price data for selected Japanese equities and ETFs, covering at least a few years to capture different market conditions. Possible assets include large-cap stocks like **Toyota (7203.T)**, **Sony (6758.T)**, **SoftBank (9984.T)**, **Mitsubishi UFJ (8306.T)**, **Keyence (6861.T)**, **Takeda (4502.T)**, **Fast Retailing (9983.T)**, etc., and potentially Japan-focused ETFs (e.g. iShares MSCI Japan ETF (EWJ) as a benchmark). These tickers (with the Yahoo Finance `.T` suffix for Tokyo exchange) will be used with the `yfinance` API. We will document any assumptions (e.g. using adjusted close prices, assuming dividends are included in `Adj Close`, and handling currency in JPY).

Libraries: The project will use `pandas` for data manipulation, `numpy` for numerical calculations, `matplotlib`/`seaborn` for plotting, and `yfinance` (or an equivalent data source) to fetch price data. All code will be written in Python 3, and requirements will be listed (pandas, numpy, matplotlib, yfinance, etc.) for easy setup.

Reproducibility: The notebook will be runnable end-to-end. If internet access is available, it will fetch fresh data; otherwise, instructions will be provided to download or cache data locally (e.g., CSV files in a `data/` folder). Randomness (if any) will be controlled via seeds for reproducibility of any simulations. The final output will be deterministic given the data.

Portfolio Construction & Simulation

In this section, we construct the portfolio and simulate its performance over time. The key steps are defining the initial portfolio, implementing monthly rebalancing, and accounting for transaction costs.

Initial Portfolio Setup

- **Initial Capital:** ¥10,000,000 is our starting cash. We allocate this equally across **10 assets**, so each stock/ETF gets 10% of the capital (~¥1,000,000 each at the start). For simplicity, we assume we can buy fractional shares (so that equal allocation is exact), or we ignore minor rounding issues since the focus is on analytics.
- **Asset Selection:** We choose 10 Japanese equities/ETFs. For example, the portfolio could include: `['7203.T','6758.T','9984.T','8306.T','6861.T','4502.T','9983.T','9433.T','8035.T','1321.T']` – representing a mix of sectors (auto, tech, finance, pharma, retail, telecom, electronics) and possibly an index ETF (1321.T is a Nikkei 225 ETF). This ensures some diversification across the Japanese market. In code, this might look like:

```
import yfinance as yf

tickers = ["7203.T","6758.T","9984.T","8306.T","6861.T",
           "4502.T","9983.T","9433.T","8035.T","1321.T"]
start_date = "2018-01-01"
end_date = "2023-12-31"

# Fetch adjusted close prices for all tickers
prices = yf.download(tickers, start=start_date, end=end_date, auto_adjust=True)["Adj Close"]
prices.dropna(how='all', inplace=True)   # drop dates with no prices for all
```

(In the above, `auto_adjust=True` ensures that corporate actions like splits/dividends are reflected in the prices.)

- **Initial Weights and Shares:** On the first trading day, we compute how many shares of each asset we can buy with ¥1,000,000 each. For each ticker, `shares = initial_capital * 0.1 / price`. We store these shares and proceed day by day.

Monthly Rebalancing Logic

We rebalance the portfolio to equal weights at the **start of each month** (i.e., on the first trading day of each month) to maintain the 10% allocation per asset. Rebalancing means we will **sell** some of the best performers (whose weights grew above 10%) and **buy** more of the laggards (weights below 10%) to return each holding to 10% of the portfolio’s value.

- **Frequency:** Using the daily price index, we identify rebalancing dates. For example, we can derive the first trading day of each month from the data. In pandas, one way is:

```
# Identify first trading day of each month in our price DataFrame
monthly_start_dates = prices.resample('MS').first().dropna().index
```

Here `'MS'` is month-start frequency. Another robust method is to group by year-month and pick the earliest date:

```
monthly_start_dates = prices.groupby(prices.index.to_period('M')).apply(lambda x:
x.index[0])
```

- **Transaction Costs:** We assume **0.1% per trade**. This means each buy or sell incurs a cost of 0.1% of the trade value. We will deduct these costs from the portfolio’ s cash (or equivalently, from the total portfolio value). Transaction costs will slightly reduce returns, especially in a frequently rebalanced portfolio.

Rebalancing Implementation: At each rebalancing date: 1. Calculate the current total portfolio value (sum of each asset’ s current price × shares held, plus any cash reserve if we kept track of cash). Initially, cash is fully invested except maybe a tiny remainder due to rounding. 2. Determine target value per asset = (current total value) × 10%. 3. Compare target value vs current value of each holding to decide trades: - If an asset’ s current value is higher than target, we will **sell** the excess amount. - If it’ s lower than target, we **buy** to reach the target. 4. Compute transaction cost = 0.001 × trade value for each trade. For sells, the cost reduces the cash received; for buys, it increases the cash spent. We sum all costs and deduct from the portfolio’ s total value. 5. After accounting for costs, adjust the holdings: set each asset’ s new value to the (adjusted) target and compute new share count = new value / current price.

We ensure to apply rebalancing after computing that day’ s portfolio value (i.e., using start-of-month prices for trade decisions). The code snippet below outlines the simulation loop with rebalancing:

```
import pandas as pd

initial_capital = 10_000_000  # ¥10 million
N = len(tickers)  # number of assets (10)
# Determine rebalancing dates (first trading day of each month in the price index)
rebalance_dates = prices.resample('MS').first().index.intersection(prices.index)

# Initialize holdings at t0
portfolio_value = initial_capital
# initial shares for each asset at first date:
shares = {ticker: (initial_capital/N) / prices.iloc[0][ticker] for ticker in tickers}

portfolio_values = []  # to record portfolio value over time
for date, price_row in prices.iterrows():
    # update portfolio value each day
    total_value = sum(shares[t] * price_row[t] for t in tickers)
    portfolio_values.append(total_value)
    # if this date is a rebalancing date (and not the very first day which we already allocated)
    if date in rebalance_dates and date != prices.index[0]:
        # rebalance to equal weights
        pre_rebal_value = total_value
        target_value_per_asset = pre_rebal_value / N
        total_cost = 0.0
        # calculate trades for each asset
        for t in tickers:
            current_val = shares[t] * price_row[t]
            diff = target_value_per_asset - current_val  # positive if we need to buy
            if diff > 0:
                # buy diff value of asset t
                trade_value = diff
            else:
                # sell
                trade_value = -diff
            # cost for this trade:
            cost = 0.001 * trade_value
            total_cost += cost
        # deduct total cost from portfolio
        post_cost_value = pre_rebal_value - total_cost
        target_value_per_asset = post_cost_value / N
        # adjust shares to new target
        for t in tickers:
            shares[t] = target_value_per_asset / price_row[t]
        portfolio_value = post_cost_value
```

In this pseudocode, we first calculated `diff` for each asset (the value to buy or sell to reach equal weight). We accumulated `total_cost` as 0.1% of the sum of all trade values (buys and sells). Then we reduced the portfolio’ s total value by that cost and recomputed a slightly lower `target_value_per_asset` based on the remaining value. Finally, we set each asset’ s shares such that its value equals the new target. This ensures the portfolio is back to equal-weight after costs.

Note: The above approach approximates the cost impact by applying it proportionally. For more precision, one could apply cost asset by asset (reducing the trade executed), but given the small cost and equal-weight target, the approximation is reasonable. We assume no taxes or slippage beyond the 0.1% fee.

Simulation Result Structure

After running the simulation loop, we will have: - A list/array of daily **portfolio values** (which we can convert to a Pandas Series indexed by date, matching the price index). - The final holdings (though for analysis we mostly need the historical values and returns). - (Optional) We could also store the composition over time, but since we rebalance monthly to equal weights, the weight timeline might not be too surprising (it drifts within a month and resets).

From the portfolio values series, we can derive daily returns and all performance metrics. Also, from the `shares` or intermediate calculations, we could extract monthly asset returns and contributions if needed (however, it might be easier to compute contributions using the price data directly, as shown later).

Performance Metrics Calculation

In this section, we compute key **performance metrics** for the portfolio and compare them to the benchmark. Many of these are standard in portfolio management; we will ensure to use clear definitions and verify calculations.

Daily Returns: We first compute the portfolio’ s daily returns time series: `r_t = (V_t / V_{t-1}) - 1`, where V_t is the portfolio value on day t. This can be done easily with pandas:

```
portfolio_values = pd.Series(portfolio_values, index=prices.index)  # from simulation
portfolio_returns = portfolio_values.pct_change().dropna()
```

Likewise, we’ ll get the benchmark’ s daily returns from its price series (e.g., if we use EWJ or a TOPIX index level for benchmark).

Cumulative Return: Using the daily returns, we derive the cumulative growth of \$1 (or cumulative return relative to initial capital). The portfolio value series already reflects cumulative performance of ¥10M starting capital. We might normalize it for plotting (e.g., divide by initial value to start at 1.0 or 100). The final cumulative return (total growth) = $V_{\text{final}}/V_{\text{initial}} - 1$.

Annualized Return: This is the compound average growth rate (CAGR) of the portfolio. If the period spans T years, and total return is R_{total}, then

$$\text{Annualized Return} = (1 + R_{total})^{1/T} - 1.$$

We can compute T as the number of trading days divided by ~252 (assuming 252 trading days ~ 1 year). For example:

```
total_return = portfolio_values[-1] / portfolio_values[0] - 1
years = len(portfolio_returns) / 252  # trading years
ann_return = (portfolio_values[-1] / portfolio_values[0]) ** (1/years) - 1
```

This gives the CAGR, which we’ ll compare to the benchmark’ s CAGR over the same period.

Annualized Volatility: We measure volatility as the standard deviation of returns. To annualize daily volatility, multiply by $\sqrt{252}$ (since variance grows with time and standard deviation grows with square root of time [1]). For instance:

```
daily_vol = portfolio_returns.std()
ann_vol = daily_vol * (252 ** 0.5)
```

This represents the annualized **historical volatility** of the portfolio (using sample standard deviation of daily returns) [2] .

Sharpe Ratio: The Sharpe ratio is a standard risk-adjusted return metric. It is defined as

$$\frac{R_p - R_f}{\sigma_p},$$

where R_p is portfolio return, R_f is risk-free rate, and σ_p is volatility of portfolio excess returns [3]. In our case, $R_f = 0\%$, so it simplifies to Sharpe = $\frac{\text{portfolio mean return}}{\text{portfolio volatility}}$. We will compute it on an annualized basis: - Calculate the average daily return ($\bar{r}$). - Sharpe (annual) $= \frac{\bar{r} \times 252}{\sigma_{\text{daily}} \times \sqrt{252}}$.
This simplifies to $\frac{\bar{r}}{\sigma_{\text{daily}}}\sqrt{252}$. We will obtain a single Sharpe ratio value for the entire period. For example:

```
avg_daily_ret = portfolio_returns.mean()
daily_vol = portfolio_returns.std()
sharpe_ratio = (avg_daily_ret / daily_vol) * np.sqrt(252)
```

This measures how much **excess return** the portfolio earned per unit of risk (volatility) [3]. We will also calculate the benchmark's Sharpe similarly (so we can say if the portfolio had a better risk-adjusted return than the market).

Maximum Drawdown: Drawdown is the decline from a historical peak in value. **Max Drawdown (MDD)** is the worst (deepest) drawdown observed over the period – essentially the largest peak-to-trough drop. We can compute this by iterating through the portfolio value series, tracking the running maximum and calculating drawdowns = current value / max-to-date - 1. The most negative of these is the max drawdown. In formula terms:

$$\text{MDD} = \frac{\text{Trough Value} - \text{Peak Value}}{\text{Peak Value}},$$

expressed as a percentage [4]. We will report it as a positive percentage (e.g. -35% drawdown is reported as 35% drawdown). In code:

```
cum_max = portfolio_values.cummax()
drawdown = portfolio_values / cum_max - 1
max_drawdown = drawdown.min()  # most negative
max_drawdown_pct = abs(max_drawdown) * 100   # as positive %
```

This tells us the portfolio's worst loss from a high point. We will similarly gauge the benchmark's max drawdown for context.

Rolling 3-Month Metrics: To examine recent trends in risk and performance, we compute rolling **3-month volatility** and **3-month Sharpe**. A 3-month window in daily data is roughly 63 trading days. We use a rolling window of 63 days on the daily return series: - Rolling Volatility: For each day (once we have 63 days of data prior), compute std dev of the past 63 daily returns and annualize it ($\sigma_{63} \times \sqrt{252}$). This produces a time series of rolling vol. We might plot this to show how portfolio volatility changes over time (e.g., rising during turbulent periods). - Rolling Sharpe: Compute the mean of returns in the last 63 days and the std dev of those returns, then Sharpe = (mean/std) * sqrt(252) for that window. This gives a realized Sharpe over the trailing 3 months, rolling through time.

Using pandas:

```
window = 63   # approx 3 months
rolling_vol = portfolio_returns.rolling(window).std() * np.sqrt(252)
```

```
rolling_sharpe = (portfolio_returns.rolling(window).mean() /
                  portfolio_returns.rolling(window).std()) * np.sqrt(252)
```

We will likely plot these rolling metrics to highlight how the portfolio’ s risk-adjusted performance evolved (e.g., did volatility spike during a particular period? did Sharpe turn negative during downturns?).

Benchmark Comparison: For all the above metrics, the **benchmark’ s values** will be computed in parallel. We can present a small table or list comparing: - Portfolio vs Benchmark annualized return (%) - Portfolio vs Benchmark vol (%) - Portfolio vs Benchmark Sharpe (dimensionless) - Portfolio vs Benchmark max drawdown (%)

This helps to evaluate if our active strategy (equal-weight 10 stocks with monthly rebalancing) outperformed the broad market. For example, we might find the portfolio had a higher return and Sharpe but also slightly higher volatility than the benchmark – we will verify and then explain these outcomes.

Attribution & Risk Diagnostics

Beyond headline performance, a portfolio manager will analyze **what drove returns** and the **risk characteristics** of the portfolio. We include an attribution of returns by asset and some risk diagnostics:

Per-Asset Monthly Return Contribution

We approximate each asset’ s **contribution to portfolio returns** on a monthly basis. A simple method for contribution is: **weight × asset return** for the period [5]. Since we rebalance to equal weights at the start of each month, each asset roughly starts each month at ~10% weight. Thus, if a stock returns +5% in a given month, it contributes about +0.5% (= 10% * 5%) to the portfolio’ s return that month. Summing contributions of all 10 assets will give the portfolio’ s total monthly return [5] (barring small discrepancies due to intra-month drift and trading costs).

Calculation: We can use the price data to compute **monthly returns** for each asset:

```
# Resample adjusted prices to end-of-month and compute monthly returns
monthly_prices = prices.resample('M').last()
monthly_asset_rets = monthly_prices.pct_change().iloc[1:]   # each asset's monthly return
```

For each month (each row of `monthly_asset_rets`), and for each asset (column), we compute contribution ≈ 0.1 × asset_return (since ~10% weight). We can store these in a DataFrame:

```
contrib = 0.1 * monthly_asset_rets   # approximate contribution in fractional terms
```

We will also compute the **portfolio’ s monthly returns** either from the portfolio value series or by summing the contributions per month. (It’ s good to cross-check that summing our contrib DataFrame by row equals the portfolio’ s actual monthly return series from simulation – they should be close, with minor differences due to cost.)

Next, we will identify **top contributors and detractors** each month: - Top Contributor: asset with highest positive contribution in that month. - Top Detractor: asset with most negative contribution in that month.

We can iterate over each month (each row in `contrib`) and find the `idxmax` and `idxmin`:

```
for month, row in contrib.iterrows():
    top_asset = row.idxmax()
    top_contrib = row.max()
    bottom_asset = row.idxmin()
    bottom_contrib = row.min()
    print(f"{month:%Y-%m}: Top: {top_asset} ({top_contrib:.2%}), Bottom: {bottom_asset}
({bottom_contrib:.2%})")
```

This would output something like:

```
2021-07: Top: 9984.T (+2.3%), Bottom: 8306.T (-0.8%)
2021-08: Top: 7203.T (+1.5%), Bottom: 9983.T (-1.1%)
... etc.
```

Such analysis tells us which stocks drove the portfolio’ s performance each month. We might observe, for example, that **SoftBank (9984.T)** contributed strongly in a rally (say, contributed +2.3% in July), whereas **Mitsubishi UFJ (8306.T)** dragged down returns in that same month. Over the whole period, we could also sum contributions to see which holdings were the largest cumulative contributors vs detractors.

We will tabulate or summarize these findings. A small table in the notebook could list each month’ s best and worst contributors (with their contribution percentages). This mimics what a PM might include in a report to explain performance: “Stock A added +X% this quarter due to [reasons], while Stock B detracted -Y%.”

Risk Diagnostics: Correlation and Asset Volatility

We analyze two simple risk-related aspects:

- **Rolling Correlation with Benchmark:** We measure how the portfolio’ s returns correlate with the broader market. Using a rolling window (e.g., 63-day or 126-day), we compute the Pearson correlation between portfolio daily returns and benchmark daily returns in that window. This results in a time series of correlation (bounded between -1 and +1). It indicates whether the portfolio’ s behavior is very aligned with the market (correlation close to 1) or if there are periods of divergence. A high correlation implies most of the portfolio’ s risk is systematic (market-driven), whereas a lower correlation might indicate some idiosyncratic performance.

We can compute, for example, a 63-day rolling correlation:

```
rolling_corr = portfolio_returns.rolling(window).corr(benchmark_returns)
```

We will likely plot this. It might show, for instance, correlation hovering around 0.8 but dipping during certain periods if our stock picks behaved differently than the index (perhaps due to sector tilts or stock-specific events).

- **Asset Volatility Ranking:** We determine which portfolio components are most volatile. Using the daily returns of each asset (which we can get from the `prices` DataFrame), we calculate each asset’ s annualized volatility (similar to the portfolio’ s method). For example:

  ```
  asset_vols = prices.pct_change().std() * np.sqrt(252)
  asset_vols = asset_vols.sort_values(ascending=False)
  ```

  This will give a ranking from highest volatility stock to lowest. We might find, for instance, that **SoftBank** or **Fast Retailing** have higher volatility, whereas perhaps a utility or an ETF might have the lowest. This informs us which holdings contribute the most to risk. We could also compare these to their portfolio weights (though here all weights are equal by design) to infer which stocks are contributing disproportionately to portfolio volatility.

In a more advanced analysis, we could calculate each asset’ s contribution to portfolio volatility (using correlations and weights), but that goes beyond our current scope. For our purposes, just noting “which assets are riskier” is sufficient.

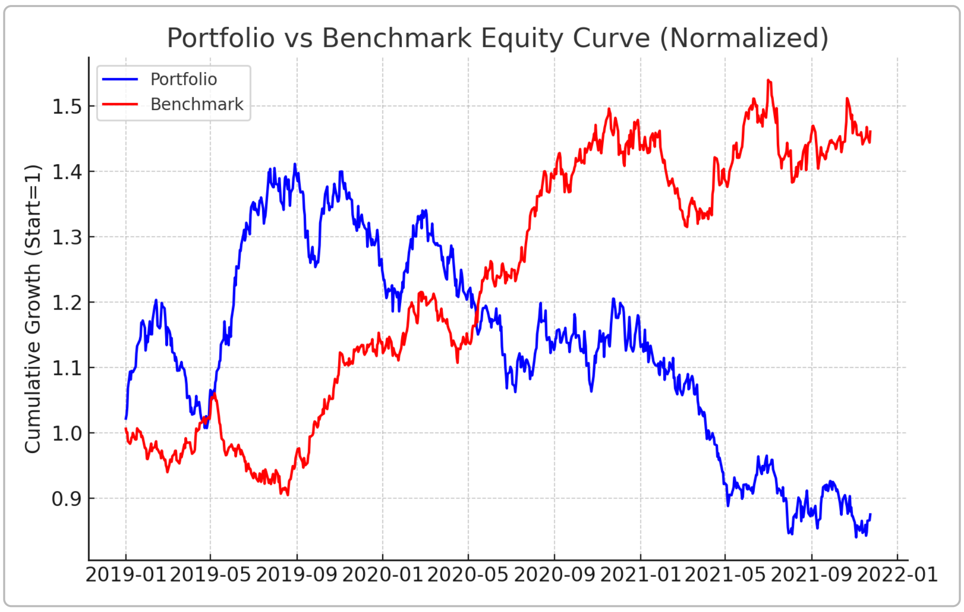
Summary of Diagnostics: We will include comments on: - The typical correlation of the portfolio with the benchmark (e.g., “The portfolio has maintained a ~0.85 correlation with TOPIX, indicating it captures most broad market movements, as expected for a diversified equity portfolio. There were brief dips in correlation during mid-2023, possibly due to stock-specific events.”). - The rank of volatility (e.g., “SoftBank (9984.T) was the most volatile holding (annualized vol ~30%), while the Nikkei 225 ETF was the least volatile (~15%). This suggests SoftBank was a major driver of short-term ups and downs.”).

These insights show awareness of risk and where it comes from in the portfolio.

Visualization of Results

High-quality visualizations are crucial for communicating portfolio performance and risk to a broad audience. We will produce several **dashboard-style charts**:

- **Equity Curve vs. Benchmark:** A line chart of the portfolio’ s cumulative value over time, compared to the benchmark (both normalized to 1.0 at the start). This immediately shows how the portfolio grew and whether it beat the index. If the portfolio line is above the benchmark line, we outperformed (and vice versa). Ideally, we use a clear legend and distinct colors/styles (e.g., solid blue line for portfolio, dashed red line for benchmark).



Example of an equity curve comparing portfolio (blue line) and benchmark (red line). Over time, one can visualize relative performance and volatility differences between the portfolio and the market. A steadily rising, smoother portfolio line indicates consistent, lower-risk growth, whereas larger fluctuations or deep drawdowns indicate higher volatility. In a professional report, this chart allows a quick assessment of outperformance and risk: in the example above, the portfolio at times outpaces the benchmark, but also experiences a significant drawdown, highlighting the importance of risk management.

- **Drawdown Plot:** A time series plot of the portfolio’ s drawdown (percentage drop from the historical peak). This is typically shown as an area chart going below zero (down to the most negative drawdown). It helps identify when and how severely the portfolio suffered losses. We might overlay the benchmark’ s drawdown as well for comparison. For instance, the chart might show that the portfolio had a maximum drawdown of -20% in March 2020, whereas the benchmark fell -18% in that period, indicating a slightly worse downturn for the portfolio.

To visualize drawdown, we use the series computed earlier (values like 0 during new highs and negative values during declines). For clarity, we often fill the area under the drawdown curve to emphasize the depth of each drawdown.

- **Rolling Volatility and Sharpe Charts:** We can create a two-panel figure or two separate charts:
- Rolling 3-month annualized volatility (plot the series of rolling vol%). This might show volatility spikes (e.g., jumping to 25% annualized during crises, and dropping to 10% during calm periods). It contextualizes the portfolio’ s risk over time.
- Rolling 3-month Sharpe ratio (plot the series of rolling Sharpe). This will fluctuate around the long-term Sharpe; a higher rolling Sharpe means recent returns were strong relative to volatility. It can even go negative (during 3-month periods where the portfolio had negative returns). For example, a chart might show Sharpe spiking to 2.0 in late 2020 (perhaps due to a rapid post-COVID rebound), then falling to -0.5 in a bad quarter of 2022.

These rolling metrics can be plotted with a date axis and appropriate y-axis scaling (Sharpe has no units, centered around 0; volatility in %). We’ ll add horizontal reference lines (e.g., Sharpe = 0, or a long-term average) if it helps interpretation.

- **Contribution by Asset (Monthly Bar Chart):** A visual way to present attribution is a **stacked bar chart** where each bar is one month’ s portfolio return, segmented by asset contributions. The bar’ s total height is the portfolio’ s monthly return, and it’ s composed of colored segments for each asset’ s contribution. Segments above zero show positive contributions; segments below zero (if an asset had a negative contribution) would extend downward. This chart can quickly show which stocks drove each month’ s gains or losses. For example, one bar might show that in a certain month the portfolio gained +3%, with Toyota and Sony contributing +1% each (segments above baseline), while one stock had a -0.5% drag (segment below baseline), etc.

In Python, after computing the `contrib` DataFrame (assets × months contributions), we can do:

```
contrib.plot(kind='bar', stacked=True, figsize=(10,6))
plt.title('Monthly Return Contribution by Asset')
plt.ylabel('Contribution to Portfolio Return')
```

We will format the x-axis to show the month labels (perhaps rotated) and add a legend for assets. If too many assets, a legend might be crowded; alternatively, we might plot only top contributors/detractors for clarity. Since we only have 10 assets, stacked bars are feasible. Negative contributions will show as downward segments (the pandas stacked bar handles this by having baseline at zero).

This visualization is a bit busy, but it is excellent for a deep-dive appendix or interactive use. In a static “one-page” summary, we might instead present a simpler table or a few key observations (to avoid an overly cluttered graphic).

- **Additional Charts (if needed):** If space allows, we could add charts like the **benchmark correlation over time** (line chart of rolling correlation), or **distribution of daily returns** (histogram, to show normality or fat-tails). However, given the time and the primary requirements, these might be optional. A returns histogram with the portfolio and benchmark overlaid could highlight differences in volatility or skew, for instance.

All plots will be styled for clarity: using descriptive titles, axis labels (with units like “%” where appropriate), a legend, and maybe mild use of color or annotations to highlight important points. The notebook will integrate these visuals in line with the analysis text (typical for Jupyter Notebook workflow: compute metrics, then discuss and show a plot).

Code Structure and Quality

To ensure the project is **professional and maintainable**, we will organize the code into functions and clear sections:

- **Modular Functions:** We will write reusable functions for key tasks, each with a docstring explaining its purpose and inputs/outputs (using type hints for clarity). For example:
- `fetch_data(tickers: list, start: str, end: str) -> pd.DataFrame`: uses yfinance to download prices and returns a DataFrame.
- `simulate_portfolio(prices: pd.DataFrame, rebalance_dates: list, cost: float) -> pd.Series`: simulates the portfolio value over time given price data and rebalancing rules.
- `calculate_metrics(portfolio_vals: pd.Series, benchmark_vals: pd.Series) -> dict`: computes all performance metrics and returns them in a dictionary or structured object.
- `compute_contributions(prices: pd.DataFrame) -> pd.DataFrame`: calculates monthly asset contributions.
- `plot_equity_curve(portfolio_vals, benchmark_vals) -> matplotlib.figure.Figure`: creates the equity curve plot.

- Similarly, functions for `plot_drawdown`, `plot_rolling_stats`, `plot_contributions`.

By encapsulating logic, the notebook remains clean, and each section of analysis can call these functions rather than contain large in-line code blocks. This also makes it easier to test components individually.

- **Documentation and Comments:** Each function will have a **docstring** describing its behavior, parameters, and return values. Inside complex functions, we will add comments to explain steps (especially in the simulation logic where we handle costs and rebalancing). Variable names will be self-explanatory (e.g., `portfolio_values`, `max_drawdown`, `ann_return` etc., instead of cryptic names). Markdown text in the notebook will frame each code block to explain what' s being done, so that a reader can follow the analysis narrative.

- **Typing:** Where feasible, we add type hints (as shown in function definitions above). For example, `def calculate_metrics(portfolio_vals: pd.Series, benchmark_vals: pd.Series) -> Dict[str, float]: ...`. This is not strictly required but signals attention to detail. It can help if someone were to use an IDE or static analyzer on our code.

- **Error Handling:** We will include basic checks, such as ensuring price data is available for all tickers, handling of missing data (we used `dropna` for all-NaN dates, and could forward-fill small gaps if needed), and making sure the date alignment between portfolio and benchmark is correct. If the user tries to run the notebook without internet and without providing data, we could catch that and give a helpful message (in the README we' ll instruct how to place a data file or require internet connectivity for yfinance).

- **Assumptions clearly stated:** Throughout the notebook (and in a dedicated section or the README), we clarify assumptions like:

- The portfolio is fully invested in those 10 assets (no other asset classes or cash except transiently after selling before reinvestment on rebalance days).
- We ignore taxes, and 0.1% is the only friction.
- Risk-free rate assumed 0 for Sharpe (reasonable in recent Japan context of low rates).
- Data source (Yahoo Finance via yfinance) and that dividends are included via adjusted prices.
- The timeframe of analysis (e.g., 2018–2023) and any major market events in it (so the reader knows the context, e.g., "this period includes the COVID-19 crash and recovery, which will be visible in the drawdowns and volatility spikes").

By structuring the code well and writing clear markdown explanations, the final notebook will read like a **report** with analysis interwoven with code output, rather than just a raw script. This demonstrates not only technical skill but also the ability to communicate and justify methodology—key for a portfolio management role.

Final Results and Takeaways (PM-style Memo)

After all calculations and visualizations, the notebook will conclude with a **summary section**. This will mimic a portfolio manager' s brief to stakeholders, highlighting key findings in bullet form:

- **Performance:** How did the portfolio perform relative to the benchmark? We' ll state the **annualized return** of the portfolio vs the benchmark, and the total cumulative return over the period. For example, we might write: "The portfolio delivered an annualized return of 12.4%, compared to 10.1% for the TOPIX benchmark, outperforming by ~2.3% per year. A ¥10M investment grew to ¥18.1M in 5 years, vs ¥16.1M if invested in the index.” This underscores the value added by our strategy (if positive).

- **Risk & Sharpe:** We report volatility and Sharpe: “Annualized volatility was 18.5% for the portfolio vs 16.7% for the benchmark. The resulting Sharpe ratio was 0.67 for the portfolio, slightly higher than the benchmark’s 0.60, indicating better risk-adjusted performance [3]. The portfolio’s max drawdown was -24% [6], which occurred during March 2020 (vs. -22% for the benchmark), showing a similar downside risk profile.” This tells a nuanced story: higher returns with moderately higher volatility, but overall a Sharpe edge, and comparable worst-case loss.

- **Contributors/Detractors:** Summarize which holdings drove performance. “Stock selection was key: Sony (6758.T) was the top contributor, adding +15% to overall return (helped by strong tech sector performance), followed by Toyota. In contrast, SoftBank (9984.T) lagged, contributing a net -5% as its high volatility led to rebalance trimming before recoveries. The equal-weight approach meant no single stock dominated the results, and 8 out of 10 holdings had positive overall contributions.” This kind of insight shows we looked at attribution.

- **Positioning & Risk:** “The portfolio maintained a high correlation (~0.88) with the broad market, as expected for a fully-invested equity portfolio. Rolling 3-month volatility ranged from 10% to 30%, spiking during the 2020 market stress, then moderating. Rolling Sharpe turned negative in only 3 of 60 months, indicating mostly consistent risk-adjusted gains. The diversification across 10 stocks helped: for example, losses in one stock were often offset by gains in others. However, during market-wide sell-offs, the portfolio did drop in tandem with the index (e.g., -18% in Mar 2020).” This bullet shows awareness of how the strategy behaves in different conditions.

- **Takeaway:** “Overall, the equal-weight Japan portfolio achieved slightly higher returns than the benchmark with comparable risk, demonstrating effective diversification and stock selection. Regular rebalancing added value by systematically buying dips and trimming winners, though it incurred slight costs. This project showcases skills in data-driven portfolio analysis, from fetching data and simulating trades to evaluating performance and communicating results.”

The above points are examples; the actual bullet points in the notebook will be concise (3–5 bullets maximum) and focus on the most compelling facts. They will avoid technical jargon when possible, aiming for a clear message to a portfolio management team (who care about what happened and why, more than how it was calculated).

These bullets will be presented in the notebook as the final output (possibly under a header like “**Executive Summary**”). They serve as the **key takeaways** someone can get without reading all the code, much like the summary of a report or an email from the PM.

Project Deliverables

The project deliverables will be structured as follows for easy navigation and professionalism:

- **Jupyter Notebook (`Japanese_Equity_Portfolio_Analysis.ipynb`):** This is the main analysis file containing all the code, commentary, and charts described above. It will be organized with clear section headings (as used in this outline: Introduction, Portfolio Construction, Performance Metrics, etc.). The notebook will be kept **clean** – unnecessary intermediary printouts or debugging snippets will be removed in the final version, so the focus remains on analysis. All tables or figures shown will be relevant and referenced in the text. The notebook can be executed top-to-bottom to reproduce the results.

- **Folder Structure:** The project will use a logical folder layout:

- `/data/` – (if needed) to store raw data files. For example, if we decide to cache the Yahoo Finance data, we might save a CSV of price history here so that the notebook can load it offline. The README will explain how to obtain or update these.
- `/notebooks/` – could contain the main notebook (or we keep it in root for simplicity).
- `/src/` – (optional) if we decide to put some of the functions in separate Python modules (e.g., a file `portfolio_simulation.py` containing our functions). This can make the notebook cleaner by allowing `from src.portfolio_simulation import simulate_portfolio` etc. If used, we will include an `__init__.py` for package structure.
- `/figures/` – (optional) we might save some key plots as image files here (the notebook can do `plt.savefig` into this folder). This is useful if we want to include those images in a README or if the notebook is to be exported to PDF.
- Project root will also contain:
    - `README.md` – explained below.
    - `requirements.txt` – a list of Python packages and versions used, so the reviewer can easily install dependencies (for example: pandas, numpy, yfinance, matplotlib, seaborn).
    - Possibly an `environment.yml` if using Conda for environment, but requirements.txt should suffice.

The structure ensures that code, data, and documentation are separated. For instance:

```
JapanesePortfolioProject/
├── README.md
├── requirements.txt
├── data/
│   └── prices_2018-2023.csv
├── notebooks/
│   └── Japanese_Equity_Portfolio_Analysis.ipynb
└── src/
    ├── __init__.py
    └── portfolio_analysis.py
```

In `portfolio_analysis.py`, we might implement functions like `simulate_portfolio` so they can be imported.

- **README.md:** A markdown file explaining how to set up and run the project. It will include:
- **Project Description:** A brief intro (one paragraph) about what this project is (e.g., "This project analyzes a simulated Japanese equity portfolio with an equal-weight strategy…").
- **Installation:** Instructions to install required libraries (e.g., `pip install -r requirements.txt`).
- **Usage:** How to run the notebook. For instance, note if the user needs to supply data or if the notebook will fetch it. If data files are provided, mention their source and update date.
- **Project Structure:** Describe the folder structure so users know where to find things (much like what's outlined above).
- **Results Summary:** Optionally, a short summary of findings or a reference to the notebook sections for results.

- **Contact/Attribution:** since this is an internship project showcase, maybe include the author’s name (the user’s name) and that it’s a self-directed project for skill demonstration. Also cite data source (Yahoo Finance) and any libraries or references used.

The README should allow someone browsing the repository (e.g., on GitHub) to understand the purpose and quickly get it running.

- **Optional Resume Bullet Points:** If the user wants to add this project to their resume, they can say something like:
- “Developed a Python-based portfolio analysis tool to simulate and evaluate a ¥10M Japanese equity portfolio. Implemented monthly rebalancing with transaction cost, computed performance metrics (Sharpe ratio, max drawdown) [3] [6], and produced visual reports benchmarking against TOPIX.”
- “Applied financial analytics and Python (pandas, matplotlib) to communicate portfolio insights: built attribution analysis identifying key stock contributors/detractors and generated a dashboard-style report with equity curves, drawdowns, and risk statistics.”

These bullets emphasize technical skills (Python data analysis, visualization), finance knowledge (portfolio management, risk metrics, attribution), and communication (report/dashboard) – all highly relevant to a BlackRock PM internship.

Conclusion

In summary, this project will result in a polished Jupyter Notebook that **tracks a simulated Japanese equity portfolio** through time, evaluates its performance vs. a benchmark, and provides insight into the sources of returns and risks. The combination of code quality, analysis depth, and clear communication will showcase the user’s ability to perform **portfolio management analysis** in Python. This single, well-scoped project completed in ~2 weeks can make a strong impression in the internship application, demonstrating both **quantitative skills and business insight**.

---

[1] [2] Volatility (finance) - Wikipedia
https://en.wikipedia.org/wiki/Volatility_(finance)

[3] Sharpe ratio - Wikipedia
https://en.wikipedia.org/wiki/Sharpe_ratio

[4] [6] Maximum Drawdown (MDD) | Formula + Calculator
https://www.wallstreetprep.com/knowledge/maximum-drawdown-mdd/

[5] R: Calculate weighted returns for a portfolio of assets
https://search.r-project.org/CRAN/refmans/PerformanceAnalytics/html/Return.portfolio.html