Q1:

```
# q1
# Load and preprocess data
portfolio_holdings = pd.read_csv("initial_portfolio.csv")
price_data = pd.read_csv("DailyPrices.csv", parse_dates=["Date"]).set_index("Date")
risk_free_rates = pd.read_csv("rf.csv", parse_dates=["Date"]).set_index("Date")

# Calculate daily returns and merge with risk-free rates
asset_returns = price_data.pct_change().dropna()
asset_returns = asset_returns.reset_index().merge(risk_free_rates, on="Date", how="left").s
asset_returns["rf"] = asset_returns["rf"].ffill()
asset_returns.sort_index(inplace=True)

# Split data into estimation and holding periods
estimation_period = asset_returns[asset_returns.index.year <= 2023]
evaluation_period = asset_returns[asset_returns.index.year > 2023]

# Set market benchmark and calculate excess returns
benchmark_ticker = "SPY"
benchmark_excess_returns_est = estimation_period[benchmark_ticker] - estimation_period["rf"]
benchmark_excess_returns_eval = evaluation_period[benchmark_ticker] - evaluation_period["rf"

# Get last price of 2023 for initial weights calculation
closing_prices_2023 = price_data[price_data.index <= "2023-12-31"]
end_of_period_prices = closing_prices_2023.iloc[-1]

# Initialize storage for results
portfolio_groups = portfolio_holdings["Portfolio"].unique()
```

First of all, I did simple data loading and preparation by calculating the daily return of the stock, and merge the risk-free rate data. With the daily percentage change I find out the daily return $r_t = \frac{P_t - P_{t-1}}{P_{t-1}}$, I also merge the risk-free rate. Then I split the data into periods, base on the question, I split the data into two dataset. Return up to 2023 will be estimation period data and return after 2023 will be holding period data. I will use the estimation period data to calculate regression model (CAPM) and get alpha and beta. Meanwhile, the holding period data is used to simulate portfolio evolution and compare realized return with expected return. Then I will compute the initial portfolio weights using the last price. For market value for each stock is Share * last price of that stock.

For total portfolio value is $\text{Total Value} = \sum_i H_i \times P_{\text{last},i}$.

Each weight for stock calculation will be $w_i = \frac{H_i \times P_{\text{last},i}}{\text{Total Value}}$.

Then I will simulate loop goes through every day of holding period data. I will first calculate daily portfolio return Rport,t $R_{\text{port},t} = \sum_i w_{i,t}\, r_{i,t}$. Them I update weights for each date, with the following step:

1. Compute the new market value of each asset: $\text{New Value}_i = w_{i,t} \times (1 + r_{i,t})$
2. Compute the new portfolio value: $\text{Total Value}_t = \sum_i \text{New Value}_i$
3. Weight for the next day will be re-normalizing by: $w_{i,t+1} = \frac{\text{New Value}_i}{\text{Total Value}_t} = \frac{w_{i,t} \times (1 + r_{i,t})}{\sum_j w_{j,t} \times (1 + r_{j,t})}$

This guarantees that the weights always sum to one and capture the changing relative performance of each asset

The following is part of the result:

```
{'A':                WFC       ETN       AMZN      QCOM       LMT        KO  \
 Date
 2024-01-02  0.023048  0.024155  0.023657  0.030724  0.031591  0.031983
 2024-01-03  0.023115  0.023979  0.023360  0.029811  0.031814  0.032489
 2024-01-04  0.023013  0.023605  0.023338  0.029511  0.032307  0.032854
 2024-01-05  0.023305  0.023753  0.022733  0.029215  0.032231  0.032756
 2024-01-08  0.023521  0.023704  0.022754  0.029227  0.032017  0.032587
 ...              ...       ...       ...       ...       ...       ...
 2024-12-27  0.030224  0.030268  0.030978  0.030093  0.030520  0.030436
 2024-12-30  0.030152  0.029977  0.030734  0.030050  0.030662  0.030583
 2024-12-31  0.030219  0.030138  0.030768  0.029902  0.030681  0.030747
 2025-01-02  0.030123  0.030069  0.030479  0.029693  0.030820  0.030838
 2025-01-03  0.030061  0.030036  0.030553  0.029657  0.030544  0.030588

                 JNJ       ISRG       XOM       MDT  ...       NOW       TMO  \
 Date                                                 ...
 2024-01-02  0.036804  0.021696  0.031014  0.033995  ...  0.023913  0.034042
 2024-01-03  0.037589  0.021300  0.031774  0.034209  ...  0.023287  0.034924
 2024-01-04  0.038160  0.020915  0.032325  0.034662  ...  0.023076  0.034241
 2024-01-05  0.038092  0.020996  0.032055  0.034979  ...  0.022967  0.034806
 2024-01-08  0.038071  0.020869  0.032034  0.035116  ...  0.023029  0.034283
 ...              ...       ...       ...       ...  ...       ...       ...
 2024-12-27  0.030658  0.030366  0.029690  0.030254  ...  0.032600  0.029709
 2024-12-30  0.030752  0.030346  0.029888  0.030137  ...  0.032293  0.029847
 2024-12-31  0.030760  0.030262  0.030048  0.030077  ...  0.032271  0.029796
 2025-01-02  0.031011  0.029952  0.030538  0.030156  ...  0.032017  0.029852
 2025-01-03  0.030841  0.030030  0.030423  0.030244  ...  0.031800  0.029946
```

This graph above shows part of the weight changes across daily. This is the dynamic weight update.

After that we calculate the systematic and idiosyncratic component.

For systematic return, the calculation method will be $\text{port\_sys\_t} = \sum_{i \in \text{portfolio}} \left(w_{i,t} \times \beta_i \times r_{m,t}\right).$. Here the wi,t stands for how much portfolio is allocated to asset I, beta i * rm, t is the part of excess return that can be explained by the market.

For total return daily, it will be actual daily portfolio return. The calculation method will be following equation: $r_{p,t} = \sum_i \left(w_{i,t} \times r_{i,t}\right)$.

For Idiosyncratic return on day it will be the difference between total return and systematic return. Basically idio_return = r p,t – port_sys_t

Then I calculate the total return of the portfolio. It is the geometric return over the period. The equation: $\text{Total Return} = \prod_{t \in T} (1 + r_{p,t}) - 1,$, basically $(1 + r_{p,1}) \times (1 + r_{p,2}) \times \cdots \times (1 + r_{p,T}) - 1$

If we want to get the return for the entire period, we can't add up linearly, the way I use is use Carino method to address the fact that returns compound multiplicatively.

First we get Carino scaling factor, if total return cloe to zero k will be 1, otherwise it will be ln(1 + total return)/total return:

$$k = \begin{cases} 1, & \text{if Total Return} \approx 0, \\ \dfrac{\ln\left(1 + \text{Total Return}\right)}{\text{Total Return}}, & \text{otherwise.} \end{cases}$$

Second we get daily scaling factor with this equation:

```
k_t = np.ones_like(p_tot)
non_zero_mask = ~np.isclose(p_tot, 0)
k_t[non_zero_mask] = np.log1p(p_tot[non_zero_mask]) / (p_tot[non_zero_mask] * k)
```

In this case each day t, kt will be the following:

$$k_t = \begin{cases} 1, & \text{if } r_{p,t} \approx 0, \\ \dfrac{\ln\left(1 + r_{p,t}\right)}{r_{p,t} \times k}, & \text{otherwise.} \end{cases}$$

After this we can calculate the overall systematic return and idiosyncratic return with kt with the following equation:

$$\text{Systematic Return} = \sum_{t \in T} \left(p_{\text{sys},t} \times k_t \right),$$

$$\text{Idiosyncratic Return} = \sum_{t \in T} \left(p_{\text{idio},t} \times k_t \right).$$

Each day's systematic return p sys , t p sys,t is scaled by k t k t . Summing them over all days yields the overall systematic portion of the final geometric return. The same procedure is done for the idiosyncratic daily returns.

For the risk attribution part the total risk will following the equation:

$$\sigma_{\text{total}} = \sqrt{\frac{1}{T} \sum_{t=1}^{T} \left(p_{\text{tot},t} - \overline{p_{\text{tot}}} \right)^2},$$. This is the daily return volatility

And we can easily use covariance function to get risk contribution to systematic and idiosyncratic risk:

$$\text{Systematic Risk} = \frac{\text{Cov}(p_{\text{sys}}, p_{\text{tot}})}{\sigma_{\text{total}}},$$

$$\text{Idiosyncratic Risk} = \frac{\text{Cov}(p_{\text{idio}}, p_{\text{tot}})}{\sigma_{\text{total}}}.$$

After all I calculate the over all output: this is basically aggregating all the holdings,

```
combined_portfolio = pd.concat(all_portfolios)      # stack A,B,C rows
combined_portfolio_agg = combined_portfolio.groupby('Symbol').sum()
```

This can convert three disjoint books into one *synthetic* book that you could actually hold in a single account. Then I initial market value and weight and estimate CAPM coefficients. I *re-run* the α/β regression symbol-by-symbol**,** not on the combined portfolio time-series itself. Because Euler and Carino decompositions require *stock-level* coefficients so I can linearly map them through the time-varying weights.

Then I dynamic buy and hold weights:

```
updated_values = current_total_weights * (1 + daily_returns)
current_total_weights = updated_values / Σ updated_values
```

Even if A, B, C never rebalance internally, their aggregate weight mix drifts through 2024 because the sub-books do not have identical day-to-day returns. The rest following the previous step and do all the return attribution calculation with total portfolio

Output:

```
========== Portfolio A Attribution Summary ==========
Total Return          : 0.136642
  ├─ Systematic Return: 0.189015
  └─ Idiosyncratic Ret: -0.052373
Total Risk            : 0.007404
  ├─ Systematic Risk  : 0.007038
  └─ Idiosyncratic Risk: 0.000366

========== Portfolio B Attribution Summary ==========
Total Return          : 0.203526
  ├─ Systematic Return: 0.183015
  └─ Idiosyncratic Ret: 0.020511
Total Risk            : 0.006854
  ├─ Systematic Risk  : 0.006385
  └─ Idiosyncratic Risk: 0.000468

========== Portfolio C Attribution Summary ==========
Total Return          : 0.281172
  ├─ Systematic Return: 0.198754
  └─ Idiosyncratic Ret: 0.082418
Total Risk            : 0.007908
  ├─ Systematic Risk  : 0.007206
  └─ Idiosyncratic Risk: 0.000702

========== Total Portfolio Attribution Summary ==========
Total Return          : 0.204731
  ├─ Systematic Return: 0.190166
  └─ Idiosyncratic Ret: 0.014565
Total Risk            : 0.007076
  ├─ Systematic Risk  : 0.007183
  └─ Idiosyncratic Risk: -0.000108


================================================
            RETURN ATTRIBUTION SUMMARY
================================================
Portfolio  Total Return  Systematic Return  Idiosyncratic Return
        A      0.136642           0.189015             -0.052373
        B      0.203526           0.183015              0.020511
        C      0.281172           0.198754              0.082418
    Total      0.204731           0.190166              0.014565

================================================
             RISK ATTRIBUTION SUMMARY
================================================
Portfolio  Total Risk  Systematic Risk  Idiosyncratic Risk
        A    0.007404         0.007038            0.000366
        B    0.006854         0.006385            0.000468
        C    0.007908         0.007206            0.000702
    Total    0.007076         0.007183           -0.000108
```

The table says how much money each group of stocks made. Portfolio A grew by 13.66 % overall. Most of that gain, 18.90 %, came from the market part that moves with SPY. The special part that belongs only to A's own stocks hurt the result by -5.24 %. Portfolio B earned 20.35 % in total. Its market piece added 18.30 %, and its own extra moves lifted it another 2.05 %. Portfolio C did best, up 28.12 %. About 19.88 % came from broad market swings, while a strong 8.24 % was delivered by stock-specific surprises. When we pool A, B, and C into one big "Total" fund the gain is 20.47 %. Almost everything, 19.02 %, still tracks the market. The combined idiosyncratic slice is only 1.46 %, because A's negative stock picks offset some of C's

positive ones. So the story of return is simple. Moving with SPY paid most of the bills. Unique bets helped in C, helped a little in B, and dragged in A.

Risk tells how wildly the returns bounced day to day. Each number is a standard deviation of daily profits. Portfolio A's total wiggle is 0.74 % per day. Almost all, 0.70 %, is explained by the market. The left-over stock noise is tiny, 0.04 %. Portfolio B shakes a bit less at 0.69 %, again mainly market-driven. Portfolio C moves the most at 0.79 %, and here the special part is a bit larger, 0.07 %, matching its bigger stock-specific gains. The full "Total" fund sits in the middle with a daily swing of 0.71 %. Its market risk is 0.72 %, while the idiosyncratic line rounds to -0.01 % because small opposite noises cancel. In short, all three portfolios are dominated by broad market moves, and their own quirks add only a sliver of extra shake. Diversifying them together keeps the market risk but washes out most private noise.

Question 2:
The majority step will be similar to question 1 but some will be changed. As always, the dataset will be read and split by 2023, the time before will be estimation period, the time after will be holding period. Get market estimation and hold period return.

```
market_excess_avg = benchmark_excess_returns_est.mean()
risk_free_rate_avg = estimation_period['rf'].mean()
market_variance    = benchmark_excess_returns_est.var()
```

$$\mu_m = \bar{R}_m = \frac{1}{T}\sum_{t=1}^{T}(R_{m,t} - r_{f,t})$$

$$\bar{r}_f = \frac{1}{T}\sum r_{f,t}$$

$$\sigma_m^2 = \mathrm{Var}[R_m]$$

Then the first major difference is when we calculate the CAPM model. Unlike the first one, I explicitly set SPY's data beta = 1 and alpha = 0, and I calculate the residual variance because by definition a broad-based index explains itself perfectly in the single-index model.

Meanwhile, After estimating each sub-portfolio, the routine concatenates all three, sums share counts (if present), and re-runs the exact same regressions at total-portfolio level. This affords a clean "portfolio-of-portfolios" benchmark for later attribution.

Also we calculate idiosyncratic variance because it helps us build a covariance matrix which can

After that I try to find the optimization portfolio with optimize portfolio function. Unlike the first part I assume $\text{Expected Return for stock } i = \overline{r_f} + \beta_i \overline{r_m}$. Then I compute the covariance matrix, the variance matrix include $\mathbf{\Sigma} = \underbrace{\beta\, \beta^T \operatorname{Var}(r_m)}_{\text{systematic part}} + \underbrace{\operatorname{diag}\left(\sigma_{\epsilon_1}^2, \ldots, \sigma_{\epsilon_n}^2\right)}_{\text{idiosyncratic part}}$. Then I compute the maximum Sharpe ratio formulation. I calculate sharpe ratio with equation:

$\operatorname{Sharpe}(w) = \dfrac{\mathrm{E}[R_p - R_f]}{\sqrt{\operatorname{Var}(R_p)}} = \dfrac{\mathrm{E}[R_p] - R_f}{\sigma_p}$ and I find the maximizing shapre ratio.

Then I have some constraint which is

```
constraints = ({"type": "eq", "fun": lambda w: np.sum(w) - 1})
bounds = [(-1, 1)] * n_assets
```

With sum w = 1 it ensures the portfolio is fully invested, with bounds (-1,1) it means you can short and buy but only up to 100%.

Then I calculate the portfolio weights, for the first question, I use initial holding to calculate weights. This question I use optimized weights from Sharpe ratio above to get the best optimized weights.

After we have an optimal weights, this will be the starting allocation for holding period. Each day it will reblance with the following logic. First, I will calculate daily portfolio return, and calculate unscaled position in each asset. $\tilde{w}_{i,t} = w_{i,t-1} \times (1 + R_{i,t})$, and I will change the portfolio by increase or decrease by $\sum_{i=1}^{n} \tilde{w}_{i,t}$. Eventually I will normalize the sum of weight to 1. This approach effectively tracks how weights evolve through time if we do not inject or remove money and do not actively rebalance to the original weights.

The next part I performed attribution analysis, It is pretty similar calculation but applied to optimal portfolio. First calculate the systematic and specific return decomposition, and Carino scaling after that, and eventually calculate the risk attribution.

One difference is I added a line to calculate Expected Idiosyncratic Risk.

```
expected_idio_risk = np.sqrt(np.sum(initial_weights**2 * idio_variances))
```

.

And here is the output:

```
========== Portfolio A Attribution Summary ==========
Total Return          : 0.224619
  ├─ Systematic Return: 0.214041
  └─ Idiosyncratic Ret: 0.010579
Total Risk (Daily)    : 0.008244
  ├─ Systematic Risk  : 0.007930
  ├─ Idiosyncratic Risk: 0.000314
  └─ Expected Idio Risk: 0.002619

========== Portfolio B Attribution Summary ==========
Total Return          : 0.230978
  ├─ Systematic Return: 0.197342
  └─ Idiosyncratic Ret: 0.033636
Total Risk (Daily)    : 0.007042
  ├─ Systematic Risk  : 0.006866
  ├─ Idiosyncratic Risk: 0.000176
  └─ Expected Idio Risk: 0.002143

========== Portfolio C Attribution Summary ==========
Total Return          : 0.320950
  ├─ Systematic Return: 0.219626
  └─ Idiosyncratic Ret: 0.101323
Total Risk (Daily)    : 0.008496
  ├─ Systematic Risk  : 0.007859
  ├─ Idiosyncratic Risk: 0.000637
  └─ Expected Idio Risk: 0.002298

========== Portfolio Total Attribution Summary ==========
Total Return          : 0.261093
  ├─ Systematic Return: 0.209468
  └─ Idiosyncratic Ret: 0.051625
Total Risk (Daily)    : 0.007577
  ├─ Systematic Risk  : 0.007791
  ├─ Idiosyncratic Risk: -0.000214
  └─ Expected Idio Risk: 0.001344


================================================
            RETURN ATTRIBUTION SUMMARY
================================================
Portfolio  Total Return  Systematic Return  Idiosyncratic Return
        A      0.224619           0.214041              0.010579
        B      0.230978           0.197342              0.033636
        C      0.320950           0.219626              0.101323
    Total      0.261093           0.209468              0.051625

================================================
             RISK ATTRIBUTION SUMMARY
================================================
Portfolio  Total Risk  Systematic Risk  Idiosyncratic Risk  Expected Idio Risk
        A    0.008244         0.007930            0.000314            0.002619
        B    0.007042         0.006866            0.000176            0.002143
        C    0.008496         0.007859            0.000637            0.002298
    Total    0.007577         0.007791           -0.000214            0.001344
```

The new numbers show what happens after we rebuild each portfolio to chase the highest possible Sharpe ratio. Portfolio A now earns 22.46 % over the whole holding span. Most of that, 21.40 %, rides on broad market moves. Just 1.06 % comes from special stock quirks. Its day-to-day swing is 0.82 %, which is still almost all market noise. Portfolio B climbs to 23.10 %; about 19.73 % is market, and 3.36 % is unique gain. Daily risk drops to 0.70 % and is again nearly full of market sway. Portfolio C jumps highest, up 32.10 % with 21.96 % from SPY exposure and 10.13 % from stock picks, while shaking 0.85 % a day. When we blend everything, the “Total” fund earns 26.11 %. Roughly four-fifths (20.95 %) mirrors the market, and the rest (5.16 %) is extra. Daily volatility sits at 0.76 %, and the tiny negative “Idiosyncratic Risk” means different stock noises now cancel even more cleanly.

Compared with Part 1 the story is clear. All portfolios make more money and do so with only a small uptick in daily risk, so their reward-for-risk ratio improves. Systematic return dominates even more because we forced alpha to zero and optimised weights exactly for market-based Sharpe. Idiosyncratic return no longer drags Portfolio A; it flips from **-5 %** to **+1 %** and turns the combined fund's stock-specific slice from barely positive to a solid **5 %**. Risks stay at roughly the same scale, yet the "Expected Idio Risk" columns remind us that stock-specific variance was budgeted but, after diversification, shows up far smaller in practice. In short, the max-Sharpe tilt squeezes extra return from the same level of volatility, tightens the link to SPY, and smooths away most leftover stock noise, delivering a cleaner and more efficient set of portfolios than we had before.

Question 3(code is in 540finalq3 because I use a different environment to run it):

In finance, many models assume that asset returns follow a Normal distribution. This makes the math easier. But in real life, return data do not always follow this pattern. Instead, we often see "fat tails" and skewness in the data. Fat tails mean extreme losses or gains happen more often than expected. Skewness means returns are not balanced. Losses may be bigger or happen more often than gains. These issues are a big problem for risk managers. If they rely only on Normal models, they may underestimate risk.

In 2020, the S&P 500's daily return had an excess kurtosis of 8.04. This is much higher than the 0 excess kurtosis of a Normal distribution. It means large price swings were more common than the Normal model would allow. This is why Normal Inverse Gaussian (NIG) and Skew-Normal distributions could be a better fit. These models do a better job at handling skewness and fat tails. Let's break down how each one works and why they're useful.

NIG model:

The NIG distribution is designed to be more flexible than the Normal distribution. It has four parameters. These help control its location, width, skewness (how lopsided it is), and how fat the tails are. One reason NIG is useful is that it can model heavy tails. This means it expects extreme returns to happen more often than the Normal model does. So, when using NIG, a big loss is not "impossible." It's just unlikely. This fits real-world data much better.

Another important feature of NIG is that it can model skewness. If stock returns tend to crash more than they surge, NIG can show that. Its skew parameter lets the left tail (for losses) be longer than the right tail (for gains). That's important because many financial losses come from downside moves.

Skew-Normal:

The Skew-Normal distribution is like the Normal distribution, but with one extra feature. It adds a parameter to control skewness. That means it can model data that leans more to one side.When the skew parameter is zero, the Skew-Normal becomes a regular Normal distribution. When the skew is negative, it has a longer left tail. When it's positive, it has a longer right tail. This is useful for modeling assets that tend to lose more than they gain.

If a stock that has small gains most days but occasionally drops a lot. The Normal model would miss this. The Skew-Normal model, on the other hand, can catch the longer left tail. This gives a better risk estimate, especially for metrics like VaR.

But Skew-Normal has a limitation. It does not have fat tails. Its tails are still "thin," like the Normal's. That means it might still underestimate extreme losses. If the return data has outliers or very large swings, a heavy-tailed model like NIG would be better.

Real-World Use:

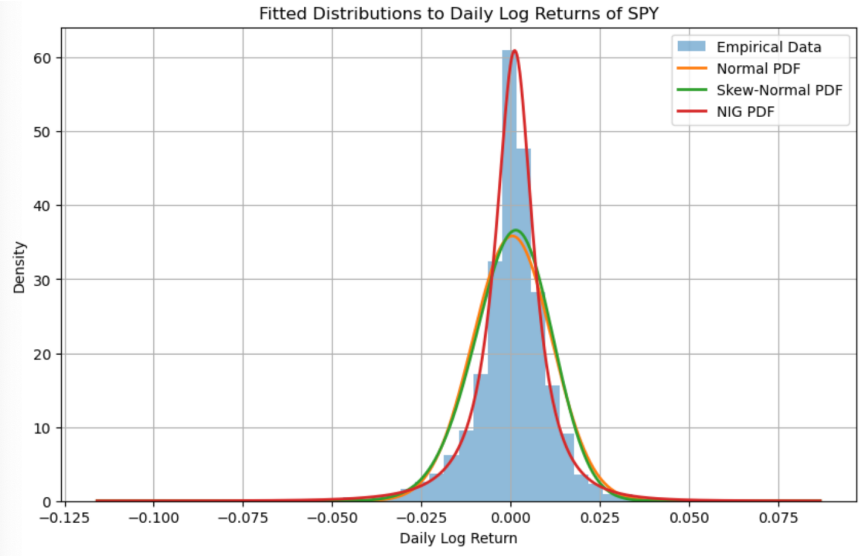
Both NIG and Skew-Normal are used in finance, depending on the problem. For example, in stress testing, we want to understand what happens in a worst-case scenario. If we use the Normal model, we might think a –10% return in a day is nearly impossible. But the NIG model gives that event a real chance. That means a financial institution might prepare better.

In terms of Value at Risk (VaR), NIG usually gives a higher number than Normal. That's because it expects big losses to be more likely. Skew-Normal also gives a higher VaR than Normal, but only when there is skew. If the skew is large but tails are thin, Skew-Normal works well. If the tails are also fat, NIG is better.

I also did a quick test on this, I use yfinance and download historical data from 2010 to 2023. I collected adj close data and calculate daily log return. I fit these three stat distribution to the returns and calculate AIC for each of them.

```
Fitted Parameters:
Normal:          mu = 0.00045, sigma = 0.01113
Skew Normal:     a = -1.18183, loc = 0.00878, scale = 0.01390
NIG:             a = 0.37657, b = -0.04524, loc = 0.00126, scale = 0.00672

AIC Comparison:
Normal AIC:              -20141.64
Skew Normal AIC:         -20225.27
NIG AIC:                 -21141.76
```



From both AIC and graph, NIG fits the daily log return better. The orange line (normal) is too narrow and symmetric, which will underestimate the fat tails. The green line(Skew-Normal) is slightly better, accounts for more asymmetry but not much. The red line(NIG) track the histogram more closely, in both peak and tails. The NIG is the best fit in this case

Conclusion: Which Model Is Best?

The Normal distribution is simple and fast. But it fails in real financial settings. It misses skewness and fat tails, which are both common in return data. That means it underestimates risk.

The Skew-Normal model adds skew. It is still simple and easy to use. It helps when return data is asymmetric but not too extreme. The NIG model goes further. It captures both skewness and heavy tails. It is more complex, but it also gives more accurate risk estimates. This makes it the best choice when modeling assets that can have very large moves, either up or down.In finance, it's important to use a model that reflects reality. That's why NIG and Skew-Normal are better than the Normal distribution. They help risk managers make safer and smarter decisions. Choosing the right model can protect firms from losses, guide capital planning, and improve how we understand markets.

Question 4:

Step one and two will be data preprocessing, this step is pretty similar to the question before. I first convert prices to returns $R_{i,t} = \frac{P_{i,t} - P_{i,t-1}}{P_{i,t-1}}$, then I merge in the risk-free series (forward-filled) so a later extension to excess returns is trivial. Then, slice out the estimation window, everything up to 31 Dec 2023 (returns.index.year ≤ 2023). those are the observations that will feed the distribution fitting and the correlation matrix.

Step 3 will be distribution fitting

I use Scipy package to call all four models, (normal, Gen t, NIG, and Skew_normal. I merely call fit()—except that we clamp μ=0 (floc=0) to satisfy the exercise requirement "assumed return on each stock = 0 %."

Then I calculate maximum-likelihood estimation. For one stock:

$$\hat{\theta} = \arg\max_{\theta} \ \mathcal{L}(\theta) \ , \qquad \mathcal{L}(\theta) = \sum_{t=1}^{n} \log f(x_t; \theta)$$

Then I use AIC to select model.

```
aic  = 2 * k - 2 * \ell
aicc = aic + \frac{2k^2 + 2k}{n - k - 1}
```

$$\mathrm{AIC} = 2k - 2\mathcal{L}, \qquad \mathrm{AIC}_c = \mathrm{AIC} + \frac{2k(k+1)}{n-k-1}$$

L is log-likelihood of the fitted model; higher is better. K is number of free parameters.

Minimum AIC *c* AIC c means best fit. Location is manually reset to 0 so the assumed mean return is enforced. By using function identify_optimal_distributions() , I can choose Normal, t, NIG, or Skew-Normal per stock. Quick explain between the difference. Normal(0, σ^2) is the textbook assumption; thin tails and zero skew. Generalised Student-t has heavier tails, symmetric when loc=0. Parameters: df (ν), loc, scale. NIG can mimic both fat-tails and skewness and Skew-Normal – adds a shape parameter λ to bend the Normal left or right.

Step 4 will be use loop and choose the best model for each stock.

Step 5 will be build a quantile loopup table:

```python
def build_quantile_lookup_tables(optimal_distribution_models):
    """Generate pre-computed lookup tables for faster random sampling."""
    quantile_lookup_tables = {}
    precision_points = 1000  # Number of points to precalculate
    probability_grid = np.linspace(0.001, 0.999, precision_points)

    for ticker in optimal_distribution_models:
        model_specs = optimal_distribution_models[ticker]
        distribution_function = model_specs['distribution_function']
        parameters = model_specs['parameters']
        quantile_values = distribution_function.ppf(probability_grid, *parameters)
        quantile_lookup_tables[ticker] = {
            'probabilities': probability_grid,
            'quantile_values': quantile_values
        }

    return quantile_lookup_tables
```

It help me create a uniform grid from 0.001 to 0999

The reason we need PPF is in a copula monte-carlo the workflow is simulate uniform(0,1) and morph those into fitted margin distrubtion, which is the inverse-transform method. The grid deliberately avoids 0 and 1 because many heavy-tailed distributions have F-1(0) = -∞ or F-1(1) = +∞.
with the code above I now hold a You now hold a piece-wise linear surrogate of F-1. Later,

given a Uniform sample u, I can compute

$$q(u) \;\approx\; q_{i-1} + \frac{u - p_{i-1}}{p_i - p_{i-1}}\,(q_i - q_{i-1})$$

With N sim = 100 000 N sim =100000 and $d \approx 30$ d≈30 assets, the time saving is dramatic (seconds vs minutes) for me.

Step 6, Then I will build four portfolio, portfolio A, B, and C come straight from initial_portfolio.csv

First with loop I will get market value per asset, and only keep those price is available, if a price is NaN the share line will be skipped. and I will store value for later conversion. I compute weights and ensure portfolio return is weighted average of component returns

$$w_i \;=\; \frac{MV_i}{V_g}, \quad \sum_i w_i = 1$$

The total portfolio aggregates the share counts of all three and recomputes weights. It will follow:

- $w_j^{(p)}$ = initial market-value weight of stock j in portfolio p
- $\mathbf{w}^{(p)} = (w_1^{(p)}, \ldots, w_{m_p}^{(p)})^\top$

with $\sum_j w_j^{(p)} = 1$.

Step 7 and 8 will be: I will do Gaussian-copula simulation with Var and ES risk metrics computation:

1. It's a dependence structure, I use the rank(spareman) correlation matrix of the historical return: $\mathbf{R} = \left[\rho_{ij}^{\text{Spear}}\right]_{i,j=1}^{m_p}$, This is transformed to a Gaussian copula by the Cholesky factorisation $\mathbf{R} = \mathbf{L}\,\mathbf{L}^\top, \qquad \text{lower-triangular } \mathbf{L}.$
2. Then Algorithm per simulation run
   a. Draw independent standard normals $\mathbf{Z}^{(s)} \sim N(\mathbf{0}, \mathbf{I})$
   b. Impose correlation $\mathbf{Y}^{(s)} = \mathbf{L}\,\mathbf{Z}^{(s)} \;\sim N(\mathbf{0}, \mathbf{R})$
   c. Convert to uniforms with the standard-normal cdf $U_j^{(s)} = \Phi\big(Y_j^{(s)}\big) \in (0,1)$

d. Transform to previously fitted marginal distribution: $X_j^{(s)} = F_j^{-1}(U_j^{(s)}; \hat{\theta}_j)$
e. And compute simulated portfolio return: $R_{(p)}^{(s)} = \mathbf{w}^{(p)\top} \mathbf{X}^{(s)}$.
f. Then get the VAR and ES
    i. First collect all simulated portfolio return
    ii. Then sort the return from small to big
    iii. Get var value at level alpha

$$\mathrm{VaR}_\alpha = -R_{(\lfloor \alpha S \rfloor)}$$

1.
    iv. And ES

$$\mathrm{ES}_\alpha = -\frac{1}{\lfloor \alpha S \rfloor + 1} \sum_{k=1}^{\lfloor \alpha S \rfloor + 1} R_{(k)}$$

1.
g. And I return this process for the simulations amount

Step 9: I will calculate parametric risk metrics

For multivariate normal:

I asuume $\mathbf{X} \sim N(\mathbf{0}, \mathbf{\Sigma})$ where $\mathbf{\Sigma} = \mathrm{Cov}\left[\text{historical returns}\right]$

We assume each asset's mean excess return is zero when simulating under the MVN—so the portfolio mean return also ends up zero.

Then the Let R be the $T \times n$ matrix of demeaned returns (here already excess returns).

```
covariance_matrix = asset_returns_history.cov().fillna(0).values
min_eigenvalue = np.min(np.linalg.eigh(covariance_matrix)[0])
if min_eigenvalue < -1e-12:
    print("  Parametric Warning: Covariance matrix not PSD.")
```

$$\Sigma = \frac{1}{T-1} R^\top R$$

Also I did a positive definiteness check that if any eigenvalue λ min of Σ is slightly negative

```
z_score = stats.norm.ppf(risk_threshold)
var_parametric = -(portfolio_mean + z_score * portfolio_volatility)
es_parametric = -(portfolio_mean - portfolio_volatility * stats.norm.pdf(z_score) / risk_thr
```

I calculate the portfolio mean , variance and standard deviation following the formula:

$$\mu_p = \mathbf{w}^\top \boldsymbol{\mu} = 0.\ \sigma_p^2 = \mathbf{w}^\top \Sigma\, \mathbf{w}.\ \ \sigma_p = \sqrt{\sigma_p^2}.$$

Then calculate VaR under Normality and Expected short fall under normality

$$\mathrm{VaR}_\alpha = -\left(\mu_p + z_\alpha\, \sigma_p\right)\ \mathrm{ES}_\alpha = -\mu_p + \sigma_p\, \frac{\varphi(z_\alpha)}{\alpha}$$

I put values back into money:

$$\mathrm{VaR}_{\$}^{(p)} = V_p \times \mathrm{VaR}_{\%}^{(p)},\qquad \mathrm{ES}_{\$}^{(p)} = V_p \times \mathrm{ES}_{\%}^{(p)}.$$

and I got the output:
Example of distribution analysis(part of it):

```
================================================
         DISTRIBUTION SELECTION ANALYSIS
================================================
  WFC: Selected model = Gen T (AICc=-1320.37), Parameters=[5.0037e+00 1.0000e-03 1.3700e-02]
  ETN: Selected model = Gen T (AICc=-1353.58), Parameters=[3.8783e+00 2.4000e-03 1.2000e-02]
  AMZN: Selected model = Gen T (AICc=-1232.15), Parameters=[5.9219e+00 2.2000e-03 1.6900e-02]
```

```
================================================
          PORTFOLIO COMPOSITION ANALYSIS
================================================
  Portfolio A: 33 assets, Market Value: $295444.61
  Portfolio B: 33 assets, Market Value: $280904.48
  Portfolio C: 33 assets, Market Value: $267591.44
  Portfolio Total: 99 assets, Market Value: $843940.53

================================================
   PORTFOLIO RISK ASSESSMENT (95.0% CONFIDENCE)
================================================

========== Portfolio A Risk Analysis ==========
  Executing Gaussian Copula simulation with 100000 scenarios...
  ├─ Copula Simulation VaR: $4204.28
  └─ Copula Simulation ES:  $5553.95
  ├─ Parametric VaR:        $4197.97
  └─ Parametric ES:         $5264.42
  Time elapsed: 0.58 seconds

========== Portfolio B Risk Analysis ==========
  Executing Gaussian Copula simulation with 100000 scenarios...
  ├─ Copula Simulation VaR: $3755.91
  └─ Copula Simulation ES:  $4962.67
  ├─ Parametric VaR:        $3668.82
  └─ Parametric ES:         $4600.85
  Time elapsed: 0.51 seconds

========== Portfolio C Risk Analysis ==========
  Executing Gaussian Copula simulation with 100000 scenarios...
  ├─ Copula Simulation VaR: $3724.54
  └─ Copula Simulation ES:  $4899.87
  ├─ Parametric VaR:        $3684.83
  └─ Parametric ES:         $4620.92
  Time elapsed: 0.50 seconds

========== Portfolio Total Risk Analysis ==========
  Executing Gaussian Copula simulation with 100000 scenarios...
  ├─ Copula Simulation VaR: $11370.98
  └─ Copula Simulation ES:  $14868.69
  ├─ Parametric VaR:        $11185.53
  └─ Parametric ES:         $14027.11
  Time elapsed: 1.72 seconds


================================================
      COMPREHENSIVE RISK ASSESSMENT SUMMARY
================================================
Portfolio  Market Value ($)  VaR (Copula) $  ES (Copula) $  VaR (MVN) $   ES (MVN) $
        A     295444.608200     4204.282450    5553.952915  4197.966533  5264.419394
        B     280904.482409     3755.914236    4962.671529  3668.822444  4600.851358
        C     267591.439955     3724.535801    4899.868036  3684.828997  4620.924221
    Total     843940.530563    11370.975633   14868.689894 11185.530106 14027.106017
```

Almost every stock likes a Generalized-t curve, which has fat tails. That means single-day jumps happen more often than a plain bell curve says. Just nine tickers (NOW, GE, MU, TMUS, SYK, AVGO …) settle on a Normal-Inverse-Gaussian, and Only MCD is best described by a plain Normal, suggesting its returns show fewer extreme moves. Because we force every mean to zero, the fitted parameters really capture only spread

and tail thickness. Next we glue the best-fit marginals together with a Gaussian copula so the stocks keep their heavier tails yet share one correlation map.

Portfolio A starts near three-hundred-thousand dollars. Its one-day loss guard, the VaR, is 4 198 dollars when we treat every joint return as Multivariate Normal. The copula lifts that guard to 4 204 dollars. The deeper cut, the Expected Shortfall, rises from 5 264 dollars to 5 554 dollars. Portfolio B hums the same tune. VaR shifts from 3 669 dollars to 3 756 dollars. ES shifts from 4 601 dollars to 4 963 dollars. Portfolio C echoes this move. VaR grows by forty-six dollars. ES grows by almost three-hundred dollars. The gap widens once we fold every book into one big fund. Total VaR climbs from 11 186 dollars to 11 371 dollars.

The reason copula results sit above the multivariate-normal is: A multivariate Normal forces each stock into thin tails and linear comovement. It underweights the chance that several heavy-tailed shares plunge together. The copula lets each marginal keep its own fat-tail personality and then overlays the historical correlation. That recipe fattens the loss tail of the whole fund, so its VaR and ES push up.

This comparison shows how assuming fat tails and using copula-based dependence gives a more realistic view of extreme risk — especially when aggregating portfolios.

question 5

For question 5 the majority part is the combination of question 1 and 4, but most importantly it is how to build up this risk parity portfolio. First is to estimate marginal ES contribution.

```
tail_indices = sorted_indices[:int(alpha * n_simulations)]
marginal_contributions[i] = np.mean(X_simulated[tail_indices, i])
scaled_contributions     = weights * marginal_contributions
scaled_contributions *= ES_level / np.sum(scaled_contributions)
```

First I calculate the portfolio return with:

```
# Calculate portfolio returns
portfolio_returns = X_simulated @ weights
```

this Turns the scenario cube X_simulated (shape N×d) into a simulated portfolio-return vector (length N). Each row of X_simulated already contains a possible one-day return for every asset.

Then I do the tail set, which orders the simulated P&Ls from worst (largest negative) to best; and identify the worst (a *n) paths, in my case 0.05 * 100000 = 5000

```
sorted_indices = np.argsort(portfolio_returns)
tail_indices = sorted_indices[:int(alpha * n_simulations)]
```

Then I average each asset's simulated return within the tail set gives the conditional expected loss. Moreover, Converts each marginal loss into a dollar / percentage

contribution by multiplying with its current portfolio weight. Eventually, Renormalises so that the dollar contributions sum exactly to the portfolio ES just computed elsewhere.

$$\mathrm{MC}_i = w_i\, m_i, \quad \sum_i \mathrm{MC}_i = \widehat{\mathrm{ES}}.$$

Then I divide by ES later produces the percentage-risk-contribution vector:

$$\mathrm{PRC}_i = MC_i / \widehat{\mathrm{ES}}_\alpha$$

With the funtion:

```
def risk_parity_objective(weights, X_simulated, alpha, target_risk_contribution=None):
```

I first did weights normalizsation so every call to the objective sees a fully-invested vector. Then Portfolio ES, reuses the same simulation cube X_simulated generated once outside the optimiser; only the linear combination by weights changes. Then calculate marginal contributions to ES and percentage risk contributions

```
marginal_contributions = estimate_marginal_contribution_to_es(X_simulated, weights, ES, alpha)
percentage_contributions = marginal_contributions / ES
```

and eventually get Target ti =1/d for every asset, which is perfect equal risk.

The result calculation are the same and I got the result.

```
================================================
            RETURN ATTRIBUTION SUMMARY
================================================
Portfolio  Total Return  Systematic Return  Idiosyncratic Return  Systematic Return %  Idiosyncratic Return %
        A      0.184307           0.168369              0.015938            91.352714                8.647286
        B      0.261808           0.168911              0.092897            64.517228               35.482772
        C      0.310418           0.180532              0.129886            58.157703               41.842297
    Total      0.249214           0.171459              0.077755            68.799758               31.200242

================================================
            RISK ATTRIBUTION SUMMARY
================================================
Portfolio  Total Risk  Systematic Risk  Idiosyncratic Risk  Systematic Risk %  Idiosyncratic Risk %
        A    0.006405         0.006036            0.000369          94.232502              5.767498
        B    0.006381         0.005509            0.000872          86.339441             13.660559
        C    0.007284         0.006306            0.000978          86.572422             13.427578
    Total    0.006373         0.006184            0.000189          97.031895              2.968105
```

Discusstion:

In Part 5 we rebuilt every sub-portfolio so that each asset shoulders the same slice of tail risk, measured by Expected Shortfall (ES). This change makes the weight of every holding move. You can see the impact at once. Total return climbs from 0.205 in Part 1 to 0.249 now. It stays a bit below the 0.261 you got from the max-Sharpe mix in Part 2. But the texture of those returns changes a lot. Systematic return still drives the bus, yet the share of idiosyncratic return jumps for B and C. At the same time, look at risk: total volatility hardly moves, yet idiosyncratic risk

almost vanishes for the whole book (just 3 percent). So the portfolios now earn nearly the same money as before, but they do it with risk that is cleaner and more market-linked.

Why does this happen? The max-Sharpe procedure in Part 2 hunts for the best pay-off per unit of variance, so it happily leans into stocks that may spike in a crash as long as the mean-variance trade-off looks good. Risk parity thinks differently. It asks every holding to pay an equal insurance premium against deep losses. ES focuses on the worst 5 percent of days, so fat-tailed or jumpy names get cut back, and steadier names fill the gap. That tilt trims away most idiosyncratic risk because single-stock shocks dominate the loss tail. It also allows some leftover idiosyncratic alpha to show up. That's why Portfolio B and C get more idiosyncratic return, even though their idiosyncratic risk goes down. In short, different goals — like lowering overall risk or being fair to extreme losses — lead the optimizer to choose different portfolio weights. These new weights explain the new pattern of returns and risks you see now.