DegreeLive

B.Com Honours | Semester V | Calicut University

Basics of Python for Finance

Course Code: COM5FS112 (3) • Module 4 Notes

MODULE IV: PYTHON FOR FINANCIAL DATA ANALYSIS

MODULE OVERVIEW & QUANTITATIVE ASSET PRICING

The ultimate objective of computational financial literacy is the empirical modeling of asset returns, portfolio risk, market sensitivity, and risk-adjusted performance. Modern investment banking, wealth management, and quantitative hedge funds rely on computational econometrics to price securities and optimize capital allocation. Module IV explores the core quantitative frameworks of modern finance implemented in Python. Students master the mathematical formulations and computational execution of: Simple versus Continuous Logarithmic Rates of Return for single assets and multi-security portfolios; Investment Risk Measurement via Variance, Volatility, Covariance, and Correlation matrices; Decomposition of total portfolio risk into Systematic (Market) and Idiosyncratic (Diversifiable) components; Econometric Ordinary Least Squares (OLS) Linear Regression for computing empirical Alpha, Beta, and R-Squared; the Capital Asset Pricing Model (CAPM) for estimating equilibrium required returns and Security Market Line (SML) pricing; and institutional Risk-Adjusted Performance Attribution metrics (Sharpe Ratio, Treynor Ratio, and Jensen's Alpha).

UNIT 1: CALCULATING & COMPARING RATES OF RETURN IN PYTHON

The **Rate of Return** measures the percentage gain or loss generated on an investment relative to its initial cost over a specified temporal horizon. In quantitative finance, distinguishing between discrete percentage returns and continuous logarithmic returns is fundamental to mathematical model design.

1. Discrete Simple Returns vs Continuous Logarithmic Returns

Financial analysts process price time-series using two distinct return formulations:

Return Framework	Mathematical Formulation	Properties & Computational Utility
Simple Return (Discrete)	`R_t = (P_t - P_{t-1}) / P_{t-1} = (P_t / P_{t-1}) - 1`	**Cross-Sectional Additivity:** The return of a multi-asset portfolio is exactly equal to the weighted sum of the simple returns of its constituent assets: `R_p = sum(w_i * R_i)`. Essential for accounting and client wealth reporting.
Logarithmic Return (Continuous)	`r_t = ln(P_t / P_{t-1}) = ln(P_t) - ln(P_{t-1})`	**Inter-Temporal Additivity:** The cumulative return over N trading sessions is simply the arithmetic sum of the individual daily log returns: `r_{0,N} = sum(r_t)`. Essential for derivative pricing, econometric regression, and algorithmic backtesting.

Symmetry Advantage of Logarithmic Returns:
• Suppose an equity asset priced at INR 100 increases by 50% on Day 1 to INR 150, and then drops back to INR 100 on Day 2 (a 33.33% decline).
– **Simple Returns:** Day 1 = +50.0%; Day 2 = -33.33%. Arithmetic mean = (+50% - 33.33%) / 2 = **+8.33%** (misleading positive distortion despite zero net capital gain).
– **Log Returns:** Day 1 = ln(150/100) = **+0.4055**; Day 2 = ln(100/150) = **-0.4055**. Arithmetic sum = **0.0000** (accurately reflecting capital preservation).

2. Calculating a Security's Rate of Return in Python

```
import numpy as np import pandas as pd # Historical Closing Prices for Tata Consultancy Services (TCS) dates = pd.date_range(start="2025-01-01", periods=6, freq="B") prices = pd.Series([3800.0, 3850.0, 3820.0, 3900.0, 3950.0, 3920.0], index=dates, name="TCS_Close") # 1. Computing Simple Percentage Returns via Pandas .pct_change() simple_returns = prices.pct_change().dropna() # 2. Computing Continuous Logarithmic Returns via NumPy np.log() log_returns = np.log(prices / prices.shift(1)).dropna() # Comparison DataFrame df_returns_comp = pd.DataFrame({"Price": prices.iloc[1:], "Simple_Return": simple_returns, "Log_Return": log_returns}) print("TCS RETURN COMPARISON: ", df_returns_comp) # Annualizing Average Daily Returns (assuming 252 trading sessions) mean_daily_return = simple_returns.mean() annualized_simple_return = mean_daily_return * 252 compound_annual_growth = (prices.iloc[-1] / prices.iloc[0]) ** (252 / len(prices)) - 1 print(f" Annualized Arithmetic Mean Return: {annualized_simple_return:.2%}") print(f"Compound Annual Growth Rate (CAGR): {compound_annual_growth:.2%}")
```

3. Geometric Mean vs Arithmetic Mean & Volatility Drag

In wealth compounding, the **Geometric Mean Return** is always strictly less than or equal to the **Arithmetic Mean Return** due to **volatility drag**. The mathematical approximation is defined as:
`Geometric Mean Return ≈ Arithmetic Mean Return - (0.5 × Volatility^2)`

```
# Demonstrating Volatility Drag in Python initial_capital = 1000000.0 # INR 10 Lakhs # High-volatility sequence: Alternating +30% and -25% annual_returns = np.array([0.30, -0.25, 0.30, -0.25, 0.30, -0.25]) arithmetic_mean = np.mean(annual_returns) # Compound wealth growth wealth_path = initial_capital * np.prod(1 + annual_returns) geometric_cagr = (wealth_path / initial_capital) ** (1 / len(annual_returns)) - 1 approx_cagr = arithmetic_mean - 0.5 * (np.std(annual_returns, ddof=1)**2) print(f"Arithmetic Mean Return: {arithmetic_mean:.2%}") print(f"Actual Geometric Compound Growth (CAGR): {geometric_cagr:.2%}") print(f"Analytical Volatility Drag Estimate: {approx_cagr:.2%}") print(f"Ending Wealth after 6 Years: INR {wealth_path:,.2f}")
```

4. Portfolio Drift vs Periodic Rebalancing in Python

Over time, asset classes with superior performance expand as a percentage of total portfolio wealth, while underperforming assets contract. This phenomenon, known as **Portfolio Drift**, alters the intended risk profile:

```
# Simulating Portfolio Drift vs Disciplined Rebalancing initial_wealth = 10000000.0 # INR 1 Crore target_weights = np.array([0.60, 0.40]) # 60% Equity / 40% Debt # 3-Year Annual Returns: Equity (+25%, -15%, +30%), Debt (+7%, +7%, +7%) equity_ret = [0.25, -0.15, 0.30] debt_ret = [0.07, 0.07, 0.07] eq_val = initial_wealth * target_weights[0] debt_val = initial_wealth * target_weights[1] print("PORTFOLIO WEIGHT DRIFT OVER 3 YEARS (BUY & HOLD):") for yr in range(3): eq_val *= (1 + equity_ret[yr]) debt_val *= (1 + debt_ret[yr]) tot_val = eq_val + debt_val print(f"Year {yr+1}: Equity Weight = {eq_val/tot_val:.1%} | Debt Weight = {debt_val/tot_val:.1%} | Wealth = INR {tot_val:,.2f}")
```

5. Calculating a Portfolio of Securities' Rate of Return

A commercial investment portfolio aggregates capital across multiple securities. The portfolio return is computed via vector dot product of asset weights and individual simple returns:

```
# Multi-Asset Portfolio Rate of Return Engine tickers = ["RELIANCE", "HDFCBANK", "INFY", "ITC"] weights = np.array([0.35, 0.30, 0.20, 0.15]) # Sum of weights equals 1.0 (100%) # Historical Daily Asset Prices across 5 Trading Sessions price_table = pd.DataFrame({ "RELIANCE": [2800.0, 2830.0, 2815.0, 2860.0, 2890.0], "HDFCBANK": [1520.0, 1535.0, 1510.0, 1545.0, 1560.0], "INFY": [1400.0, 1420.0, 1395.0, 1430.0, 1450.0], "ITC": [440.0, 445.0, 442.0, 450.0, 452.0] }) # Step 1: Calculate Daily Simple Returns Matrix asset_returns = price_table.pct_change().dropna() # Step 2: Compute Portfolio Return via Vector Dot Product (w . R) portfolio_daily_returns = asset_returns.dot(weights) print("DAILY ASSET RETURNS MATRIX: ", asset_returns) print(" PORTFOLIO AGGREGATED DAILY RETURNS: ", portfolio_daily_returns) print(f" Mean Daily Portfolio Return: {portfolio_daily_returns.mean():.4%}") print(f"Annualized Expected Portfolio Return: {portfolio_daily_returns.mean() * 252:.2%}")
```

UNIT 2: MEASURING INVESTMENT RISK & PORTFOLIO DIVERSIFICATION

In financial economics, **risk** is defined as the dispersion or uncertainty of actual future returns around the expected mean return. While return is desired, risk is the penalty that investors must manage.

1. Variance, Volatility & Annualization Mechanics

Asset volatility is measured mathematically by the **Standard Deviation** of its return distribution:

> **Volatility Formulation & The Square Root of Time Rule:**
> • Sample Variance: s^2 = sum((R_t - R_mean)^2) / (N - 1)
> • Daily Volatility (Standard Deviation): sigma_daily = sqrt(s^2)
> • Annualized Volatility: sigma_annual = sigma_daily * sqrt(252)
> — The multiplier sqrt(252) stems from the assumption that daily returns are independent and identically distributed (i.i.d.) random variables across 252 trading sessions per fiscal year.

2. Covariance Matrix & Pearson Correlation

Measuring the interaction between pairs of assets is the cornerstone of Modern Portfolio Theory:

• **Covariance (`Cov(X, Y)`):** Measures the directional co-movement of two assets. A positive covariance

indicates assets tend to rise and fall together; a negative covariance indicates inverse movement.
• **Correlation Coefficient (rho_{xy}):** Standardizes covariance into a dimensionless metric bounded strictly between -1.0 and +1.0:
rho_{xy} = Cov(X, Y) / (sigma_x * sigma_y)

```
# Covariance and Correlation Calculation in Python cov_matrix = asset_returns.cov() *
252 # Annualized Covariance Matrix corr_matrix = asset_returns.corr() # Dimensionless
Correlation Matrix print("ANNUALIZED COVARIANCE MATRIX: ", cov_matrix.round(6)) print("
PEARSON CORRELATION MATRIX: ", corr_matrix.round(4))
```

3. Portfolio Risk Calculation: The 2-Asset vs N-Asset Matrix Framework

The total risk of an N-asset portfolio is not a simple weighted average of individual volatilities; it incorporates all pairwise cross-covariances:

MODERN PORTFOLIO THEORY RISK FORMULATION PORTFOLIO VARIANCE

PORTFOLIO VARIANCE: sigma_p^2 = w^T × COVARIANCE MATRIX (Sigma) × w

- For a 2-Asset Portfolio: sigma_p^2 = (w_1^2 * sigma_1^2) + (w_2^2 * sigma_2^2) + (2 * w_1 * w_2 * Cov_{1,2}).
- For an N-Asset Portfolio: Computed via quadratic matrix multiplication in NumPy: np.dot(weights.T, np.dot(cov_matrix, weights)).
- **The Diversification Dividend:** If correlation rho < 1.0, the total portfolio risk sigma_p is strictly less than the weighted average of individual asset risks, proving that diversification mathematically eliminates risk without sacrificing return.

```
# Calculating Total Portfolio Volatility in Python portfolio_variance =
np.dot(weights.T, np.dot(cov_matrix, weights)) portfolio_volatility =
np.sqrt(portfolio_variance) # Benchmark: Weighted Average Volatility (Without
Diversification Benefit) individual_annual_volatilities = asset_returns.std() *
np.sqrt(252) weighted_avg_volatility = np.dot(weights, individual_annual_volatilities)
diversification_benefit = weighted_avg_volatility - portfolio_volatility print(f"Total
Annualized Portfolio Risk (sigma_p): {portfolio_volatility:.2%}") print(f"Weighted
Average Individual Risk: {weighted_avg_volatility:.2%}") print(f"Diversification Risk
Reduction Dividend: {diversification_benefit:.2%}")
```

4. Systematic vs Idiosyncratic (Diversifiable) Risk Decomposition

Total investment risk is bifurcated into two distinct economic categories:

1. Systematic Risk (Market / Non-Diversifiable)

• **Definition:** Pervasive macroeconomic factors (interest rate shifts, GDP contractions, inflation spikes, geopolitical crises) affecting all market securities simultaneously.
• **Diversification: Cannot be eliminated** through diversification, regardless of how many stocks are added to the portfolio.
• **Metric:** Quantified by **Beta (`beta`)**.

2. Idiosyncratic Risk (Specific / Diversifiable)

• **Definition:** Microeconomic shocks unique to a specific firm or industry (patents, product recalls, labor disputes, corporate executive changes).
• **Diversification: Can be completely diversified away**. In an equally weighted portfolio, as the number of assets N exceeds 30 to 40 securities, idiosyncratic variance approaches zero.

5. Mathematical Convergence of Diversification

In an equally weighted portfolio where `w_i = 1/N`, total portfolio variance can be factored into two terms:
`sigma_p^2 = (1/N) × Average_Asset_Variance + (1 - 1/N) × Average_Covariance`
As `N → ∞`, the term `(1/N) × Average_Variance → 0`, meaning individual asset variance disappears completely, and the remaining risk of the portfolio is dictated exclusively by the average covariance between assets!

```
# Simulating the Diversification Limit as N Increases avg_individual_var = 0.09 # 30% Individual Volatility avg_cross_cov = 0.02 # Cross-Asset Covariance print("PORTFOLIO RISK CONVERGENCE AS ASSET COUNT (N) GROWS:") print(f"{'Asset Count (N)':<18}{'Unique Risk Contribution':>25}{'Covariance Risk':>18}{'Total Portfolio Vol':>22}") print("-" * 83) for n in [1, 2, 5, 10, 20, 30, 50, 100, 500]: unique_term = (1 / n) * avg_individual_var cov_term = (1 - 1 / n) * avg_cross_cov p_vol = np.sqrt(unique_term + cov_term) print(f"{n:<18}{unique_term:>25.6f}{cov_term:>18.6f}{p_vol:>22.2%}")
```

6. Historical Simulation vs Parametric Value at Risk (VaR) in Python

Value at Risk (VaR) answers the fundamental executive risk question: "*What is the maximum monetary loss expected over a defined holding period at a specific statistical confidence level (e.g., 95% or 99%)?*"

```
# Parametric vs Historical VaR Computation in Python portfolio_value = 50000000.0 # INR 5 Crores Portfolio confidence_level = 0.95 # 95% Confidence (Z = 1.645) z_score = 1.644853 # 1. Parametric (Variance-Covariance) Normal VaR daily_port_vol = portfolio_volatility / np.sqrt(252) daily_port_mean = portfolio_daily_returns.mean() parametric_var_1d = portfolio_value * (z_score * daily_port_vol - daily_port_mean) # 2. Historical Simulation Non-Parametric VaR historical_percentile = np.percentile(portfolio_daily_returns, (1 - confidence_level) * 100) historical_var_1d = -portfolio_value * historical_percentile print(f"Portfolio Asset Base: INR {portfolio_value:,.2f}") print(f"1-Day Parametric 95% VaR: INR {parametric_var_1d:,.2f} ({parametric_var_1d/portfolio_value:.2%})") print(f"1-Day Historical Simulation 95% VaR: INR {historical_var_1d:,.2f} ({historical_var_1d/portfolio_value:.2%})")
```

UNIT 3: REGRESSION ANALYSIS FOR FINANCIAL DATA IN PYTHON

Ordinary Least Squares (OLS) Linear Regression is the premier econometric workhorse used to quantify the relationship between an individual security's return (dependent variable Y) and the broader market

benchmark return (independent variable X).

1. The Single-Index Market Model

The Single-Index Model represents the return of stock `i` as a linear function of market index return `R_m`:

> **Econometric Market Model Formulation:**
> `R_{i,t} = alpha_i + (beta_i * R_{m,t}) + epsilon_{i,t}`
> • **`beta_i` (Beta):** Slope coefficient measuring systematic sensitivity: `Cov(R_i, R_m) / Var(R_m)`.
> • **`alpha_i` (Alpha):** Intercept coefficient measuring excess return unexplained by market movements: `R_mean_i - (beta_i * R_mean_m)`.
> • **`epsilon_{i,t}` (Residual Error):** Random zero-mean idiosyncratic error term.
> • **`R^2` (R-Squared):** Percentage of total asset variance explained by market index fluctuations.

2. Running Financial Regressions in Python

Python offers two primary libraries for econometric regression: `scipy.stats.linregress` for fast summary metrics and `statsmodels.api.OLS` for exhaustive econometric diagnostic tables:

```
from scipy import stats import statsmodels.api as sm # Simulating 252 Trading Sessions: Market Index vs Equity Security np.random.seed(42) market_daily = np.random.normal(0.0005, 0.011, 252) # NIFTY 50 Benchmark # Asset with True Beta = 1.25, True Alpha = 0.0002, plus idiosyncratic noise stock_daily = 0.0002 + (1.25 * market_daily) + np.random.normal(0, 0.006, 252) # Method 1: Fast Parameter Extraction via SciPy linregress slope_beta, intercept_alpha, r_value, p_value, std_err = stats.linregress(market_daily, stock_daily) print("SCIPY OLS REGRESSION OUTPUT:") print(f"Empirical Beta (Market Sensitivity): {slope_beta:.4f}") print(f"Daily Alpha (Excess Intercept): {intercept_alpha:.6f} (Annualized: {intercept_alpha * 252:.2%})") print(f"R-Squared (Explained Variance): {r_value**2:.4f} ({r_value**2:.1%} explained by market)") print(f"P-Value (Statistical Significance): {p_value:.4e} (Statistically significant at 1% level)") # Method 2: Comprehensive Econometric Table via Statsmodels X = sm.add_constant(market_daily) # Adds intercept column ols_model = sm.OLS(stock_daily, X).fit() # print(ols_model.summary()) # Generates full academic summary table with t-stats and F-test
```

3. Interpreting Beta Across Financial Sectors

- **`beta = 1.0` (Market Neutrality):** The security moves in tandem with the broad market index (e.g., diversified large-cap mutual funds).
- **`beta > 1.0` (Aggressive / High Beta):** The asset amplifies market volatility. A beta of `1.40` implies that if NIFTY advances by 10%, the stock is expected to surge by 14%; conversely, a 10% market crash causes a 14% plunge. Typical of Banking, Technology, and Real Estate sectors.
- **`beta < 1.0` (Defensive / Low Beta):** The asset is less volatile than the market index. A beta of `0.65` provides capital stability during market downturns. Typical of Fast-Moving Consumer Goods (FMCG), Pharmaceuticals, and Public Utilities.
- **`beta < 0.0` (Inverse Beta):** The asset moves in the opposite direction of the market index. Typical of Gold ETFs, inverse volatility instruments, and long government bond hedges.

4. Multi-Factor Asset Pricing: The Fama-French 3-Factor Model

While the single-index CAPM model relies exclusively on market Beta, Eugene Fama and Kenneth French (1993) demonstrated that market beta alone fails to explain cross-sectional equity returns. The **Fama-French Three-Factor Model** expands CAPM by incorporating two additional fundamental risk factors:

- **Market Risk Premium (`R_m - R_f`):** Traditional market sensitivity factor.
- **Size Premium (Small Minus Big - `SMB`):** Captures the historical outperformance of small-capitalization firms over large-capitalization corporations due to higher risk and growth potential.
- **Value Premium (High Minus Low - `HML`):** Captures the excess return of high book-to-market (value) stocks relative to low book-to-market (growth) stocks.

```
# Multi-Factor Fama-French Regression in Python # R_i - R_f = alpha + beta_m*(R_m - R_f) + beta_smb*SMB + beta_hml*HML + epsilon n_days = 252 mkt_premium = np.random.normal(0.0004, 0.010, n_days) smb_factor = np.random.normal(0.0001, 0.006, n_days) hml_factor = np.random.normal(0.0002, 0.007, n_days) # Stock returns driven by market beta=1.1, size beta=0.45, value beta=-0.25 stock_excess = (1.10 * mkt_premium) + (0.45 * smb_factor) - (0.25 * hml_factor) + np.random.normal(0, 0.004, n_days) X_factors = pd.DataFrame({"Mkt_RF": mkt_premium, "SMB": smb_factor, "HML": hml_factor}) X_factors = sm.add_constant(X_factors) ff_model = sm.OLS(stock_excess, X_factors).fit() print("FAMA-FRENCH 3-FACTOR MULTI-REGRESSION COEFFICIENTS:") print(ff_model.params.round(4)) print(f"Multi-Factor Model R-Squared: {ff_model.rsquared:.2%}")
```

UNIT 4: CAPITAL ASSET PRICING MODEL (CAPM) & PERFORMANCE ATTRIBUTION

Formulated independently by William Sharpe (1964), John Lintner, and Jan Mossin, the **Capital Asset Pricing Model (CAPM)** establishes the theoretical relationship between the non-diversifiable systematic risk of an asset and its equilibrium expected rate of return.

1. The CAPM Formula & The Security Market Line (SML)

CAPM asserts that investors require no compensation for holding idiosyncratic risk (since it can be diversified away freely). Expected return depends exclusively on systematic risk:

THE CAPITAL ASSET PRICING MODEL (CAPM)

EQUILIBRIUM PRICING

EXPECTED RETURN: E(R_i) = R_f + beta_i × [E(R_m) - R_f]

- **`R_f` (Risk-Free Rate):** Return on a default-free government instrument (e.g., 91-Day Reserve Bank of India Treasury Bill yield, typically ~6.5%).
- **`E(R_m)` (Expected Market Return):** Expected long-term return of the diversified market index (e.g., NIFTY 50 historical mean ~13.5%).
- **`[E(R_m) - R_f]` (Equity Market Risk Premium - ERP):** The additional hurdle return demanded by rational investors for holding risky equity instead of risk-free cash (~7.0%).
- **Security Market Line (SML):** The graphical line plotting Beta on the x-axis and Expected Return on the y-axis. Securities plotting *above the SML* generate higher returns than warranted by risk (undervalued / attractive buy); securities plotting *below the SML* are overpriced.

2. Computing CAPM Expected Return in Python

```
def calculate_capm_return(risk_free_rate, market_return, stock_beta): """Computes equilibrium expected return under CAPM.""" equity_risk_premium = market_return - risk_free_rate expected_return = risk_free_rate + (stock_beta * equity_risk_premium) return expected_return # Capital Cost Modeling for Indian Equities rf = 0.065 # 6.5% 91-Day T-Bill Yield rm = 0.135 # 13.5% Expected NIFTY 50 Annual Return beta_infy = 1.15 # Infosys Beta cost_of_equity_infy = calculate_capm_return(rf, rm, beta_infy) print(f"Risk-Free Rate (Rf): {rf:.2%}") print(f"Market Return (Rm): {rm:.2%}") print(f"Infosys Beta: {beta_infy:.2f}") print(f"CAPM Required Cost of Equity: {cost_of_equity_infy:.2%}")
```

3. Risk-Adjusted Performance Attribution: Sharpe, Treynor & Jensen's Alpha

Institutional asset managers must prove that high returns are the product of superior investing acumen rather than excessive speculative risk taking:

1. Sharpe Ratio (Total Risk)

`S_p = (R_p - R_f) / sigma_p`

• Evaluates excess return generated per unit of **total volatility**.
• Sharpe > 1.0 is Good; Sharpe > 2.0 is Exceptional.

2. Treynor Ratio (Market Risk)

`T_p = (R_p - R_f) / beta_p`

• Evaluates excess return generated per unit of **systematic risk (Beta)**.
• Ideal for evaluating well-diversified equity portfolios.

3. Jensen's Alpha (Manager Skill)

`alpha_J = R_p - [R_f + beta_p * (R_m - R_f)]`

• Measures true abnormal return in excess of the CAPM equilibrium benchmark.
• Positive alpha confirms manager skill.

```
# Performance Attribution Implementation in Python portfolio_actual_return = 0.175 # 17.5% Annual Return achieved by fund portfolio_volatility = 0.15 # 15.0% Annualized Portfolio Volatility portfolio_beta = 1.10 # Portfolio Systematic Beta rf_rate = 0.065 # 6.5% Risk-Free Benchmark market_benchmark_return = 0.135 # 13.5% NIFTY Index Return # 1. Sharpe Ratio sharpe_ratio = (portfolio_actual_return - rf_rate) / portfolio_volatility # 2. Treynor Ratio treynor_ratio = (portfolio_actual_return - rf_rate) / portfolio_beta # 3. Jensen's Alpha capm_benchmark = rf_rate + (portfolio_beta * (market_benchmark_return - rf_rate)) jensens_alpha = portfolio_actual_return - capm_benchmark print("INSTITUTIONAL PERFORMANCE ATTRIBUTION REPORT:") print(f"Achieved Portfolio Return: {portfolio_actual_return:.2%}") print(f"CAPM Required Benchmark: {capm_benchmark:.2%}") print(f"Annualized Sharpe Ratio: {sharpe_ratio:.4f}") print(f"Treynor Ratio: {treynor_ratio:.4f}") print(f"Jensen's Alpha (Outperformance): {jensens_alpha:.2%} ({jensens_alpha * 10000:.0f} basis points)")
```

4. Advanced Risk Metrics: Sortino Ratio & Information Ratio

Traditional Sharpe ratio penalizes both upside volatility and downside volatility equally. The **Sortino Ratio** improves on this by penalizing only harmful downside volatility:

```
# Computing Sortino Ratio & Information Ratio in Python target_return_mar = 0.065 # 6.5% Minimum Acceptable Return (MAR) excess_over_mar = portfolio_daily_returns - (target_return_mar / 252) # Downside Deviation considers only negative returns relative to target downside_returns = excess_over_mar[excess_over_mar < 0] downside_deviation = np.sqrt(np.mean(downside_returns**2)) * np.sqrt(252) sortino_ratio = (portfolio_actual_return - target_return_mar) / downside_deviation print(f"Annualized Downside Risk Deviation: {downside_deviation:.2%}") print(f"Sortino Ratio (Downside Protection): {sortino_ratio:.4f}")
```

5. Capital Market Line (CML) vs Security Market Line (SML) Architecture

Understanding portfolio equilibrium requires distinguishing between the Capital Market Line and the Security Market Line:

Feature Dimension	Capital Market Line (CML)	Security Market Line (SML)
Risk Metric on X-Axis	Total Risk measured by **Standard Deviation (`sigma`)**.	Systematic Risk measured by **Beta (`beta`)**.
Applicability Scope	Applies **strictly to efficient portfolios** lying on the tangent line combining risk-free asset and market portfolio.	Applies **universally to all individual assets**, inefficient portfolios, and efficient portfolios alike.
Slope of the Line	Sharpe Ratio of the Market Portfolio: `[E(R_m) - R_f] / sigma_m`.	Equity Market Risk Premium: `[E(R_m) - R_f]`.
Mispricing Diagnostics	Inefficient portfolios plot strictly *below* the CML; no asset can plot above the CML.	Underpriced securities plot *above* the SML (positive alpha); overpriced securities plot *below* the SML.

ENTERPRISE CASE BENCHMARK: QUANTITATIVE HEDGE FUND PERFORMANCE RESTRUCTURING

Malabar Multi-Asset Arbitrage Fund: An alternative investment fund (AIF Category III) in Kochi managing ₹250 Crores advertised consistent market outperformance, reporting a raw 19.5% annualized return compared to NIFTY's 14.0%. However, institutional pension trustees questioned whether the fund was delivering true investment skill or merely loading up on speculative high-beta leverage.

• By implementing Python's rigorous CAPM and econometric performance attribution models:

• The OLS regression revealed a fund Beta of `1.65` (indicating extreme vulnerability to equity market drawdowns).

• Under the CAPM formula: Required Return = `6.5% + 1.65 × (14.0% - 6.5%) = 18.88%`.

• Jensen's Alpha was computed as only `+0.62%` (62 bps), exposing that 97% of the fund's excess return was driven by excessive market leverage rather than manager stock-picking skill.

Business Result: The investment committee restructured portfolio allocations, instituting a mandatory Beta cap of `1.10` and adopting Python-driven factor hedging, which protected fund capital during the subsequent 12% market correction.

MODULE IV COMPREHENSIVE MASTERY SUMMARY

EXECUTIVE SUMMARY

LOG RETURNS + COVARIANCE MATRICES + OLS REGRESSION + CAPM + SHARPE RATIO = QUANTITATIVE ASSET PRICING

Analytical Domain	Core Quantitative Mechanism	Strategic Financial Role
Rates of Return	Simple percentage returns, continuous log returns (`np.log`), weighted vector dot products.	Guarantees time-additivity in econometric modeling and accurate multi-asset portfolio accounting.
Investment Risk Measurement	Annualized volatility (`sqrt(252)`), covariance matrix, quadratic portfolio variance (`w.T @ Sigma @ w`).	Quantifies total portfolio dispersion and unlocks Markowitz diversification benefits.
Risk Decomposition	Systematic (market) risk vs idiosyncratic (company-specific) risk.	Isolates non-diversifiable risk factors and confirms elimination of diversifiable risk via N assets.
Econometric Regressions	OLS linear regression (`scipy.stats` & `statsmodels`), empirical Beta, Alpha, and R-squared.	Measures security market sensitivity, validates pricing models, and isolates statistical significance.
CAPM & Performance Attribution	Capital Asset Pricing Model (SML), Sharpe Ratio, Treynor Ratio, Jensen's Alpha.	Determines equilibrium required hurdle rates and separates true fund manager skill from market leverage.