Portfolio Analysis System

A Financial Data Analysis Project

Project Overview

This financial portfolio analysis system evaluates investment portfolios through quantitative and qualitative analyses and benchmarking. The application processes financial time series data, calculates risk metrics, and provides comparative analysis against market indices and alternative investments.

System Architecture

Modular Design*

The system uses port_an.py as a coordination layer that manages three analysis modules:

```
def main():
portfolio = get_portfolio_input()
analyzer_scores = run_analysis('portfolio_analyzer.py', portfolio)
risk_scores = run_analysis('portfolio_risk.py', portfolio)
correlation_scores = run_analysis('portfolio_correlation.py', portfolio)
print_combined_results(analyzer_scores, risk_scores, correlation_scores)
```

This approach separates concerns across different analytical components while maintaining data consistency through subprocess communication. The architecture allows independent development and testing of each analysis module.

Data Processing Pipeline*

The system handles financial data through several stages:

Historical data acquisition with validation

start_date = self.end_date - timedelta(days=365*5) # 5-year analysis

self.returns = self.prices.pct_change(fill_method=None).dropna()

Portfolio aggregation

for symbol, weight in self.portfolio.items():

self.portfolio_returns += self.returns[symbol] * weight

```
The pipeline includes data quality checks, missing value handling, and ensures sufficient da
```

Key Features*

1. **Portfolio Input Collection**:

• The `get_portfolio_input` function prompts the user to input stock symbols and their respective allocations (as percentages).

• It validates the input to ensure:

• Allocations are positive.

• The total allocation does not exceed 100%.

• The total allocation equals 100% (with a small tolerance for floating-point precision).

1. **Portfolio Analysis Execution**:

• The `run_analysis` function runs **three** external Python scripts as subprocesses to analyze the portfolio:

• `portfolio_analyzer.py` (performance analysis)

• `portfolio_risk.py` (risk metrics analysis)

• `portfolio_correlation.py` (correlation analysis with benchmarks)

• It passes the portfolio data to these scripts via standard input and captures their output.

1. **Enhanced Score Parsing**:

• The `parse_scores` function extracts comprehensive metrics from the output of the analysis scripts, including:

• **Portfolio Scores**: Performance, risk, and correlation scores

• **Benchmark Scores**: Performance and risk scores for predefined benchmarks

• **Additional Metrics**: Sharpe ratios and maximum drawdown percentages

• **Correlation Data**: Portfolio correlations with various benchmarks

• It maps benchmark names to their respective ETF symbols (e.g., 'S&P; 500' → 'SPY')

• Handles multiple output formats from different analysis scripts

1. **Comprehensive Results Display**:

• The `print_combined_results` function formats and prints detailed results, including:

• **Portfolio Scores**: From analyzer, risk, and correlation scripts, with a final averaged score

• **Additional Portfolio Metrics**: Sharpe ratio and maximum drawdown

• **Benchmark Scores**: Performance, risk, and final scores for each benchmark

• **Correlation Analysis**: Portfolio correlations with benchmarks

• Results are appended to `portfolio_analysis.txt`

1. **Main Workflow**:

• The `main` function orchestrates the entire process:

1. Collects portfolio input.

1. Runs the analyzer, risk, and correlation scripts.

1. Displays and saves the combined results.

Key Functions*

• **`get_portfolio_input`**:

• Collects and validates user input for portfolio symbols and allocations.

• **`run_analysis`**:

• Executes external scripts (`portfolio_analyzer.py`, `portfolio_risk.py`, `portfolio_correlation.py`) and captures their output.

• **`parse_scores`**:

• Parses the output of the analysis scripts to extract scores, Sharpe ratios, maximum drawdown, and correlation data for the portfolio and benchmarks.

• **`print_combined_results`**:

• Formats and displays comprehensive results including performance scores, risk metrics, correlations, and saves them to `portfolio_analysis.txt`.

Financial Metrics Implementation

Risk Analysis Framework

The system calculates multiple risk measures:

Risk metric calculations

self.metrics['Sortino Ratio'] = self._calculate_sortino_ratio()

self.metrics['VaR (95%)'] = self._calculate_var(0.95)

self.metrics['CVaR (95%)'] = self._calculate_cvar(0.95)

self.metrics['Information Ratio'] = self._calculate_information_ratio()

```
    These calculations include downside deviation analysis, tail risk measurement, and relative
```

Portfolio Scoring System

The scoring mechanism normalizes different metrics to a common scale:

```
    def _calculate_metric_scores(self, metrics):
    scores = {}
    scores['Annual Return'] = min(10, max(0, (metrics['Annual Return'] + 0.10) * 50))
    scores['Annual Volatility'] = min(10, max(0, 10 - metrics['Annual Volatility'] * 33.33))
    scores['Max Drawdown'] = min(10, max(0, 10 + metrics['Max Drawdown'] * 20))

    def _calculate_weighted_score(self, scores):
    for metric, score in scores.items():
    weight = self.metric_weights[metric]
    total_score += score * weight
```

The weighting system reflects the relative importance of different metrics in portfolio evaluation. Critical metrics like drawdown and diversification receive higher weights in the final calculation.

Scenario Analysis

Stress Testing Implementation

The system models portfolio performance under various market conditions:

```
    self.stress_scenarios = {
    'Severe Market Crash': {'Equity': -0.40, 'Bond': 0.05, 'Gold': 0.15, ...},
    'Stagflation': {'Equity': -0.25, 'Bond': -0.15, 'Gold': 0.35, ...},
    'Financial Crisis': {'Equity': -0.35, 'Bond': -0.10, 'Gold': 0.25, ...},
    }
```

Each scenario applies factor-based shocks to different asset classes. The approach maps individual securities to broader categories, allowing systematic risk assessment across economic environments.

Correlation Analysis

Diversification Measurement

The correlation module quantifies portfolio diversification:

```
def calculate_diversification_score(corr_matrix):
upper_triangle = corr_matrix.where(np.triu(np.ones_like(corr_matrix), k=1).astype(bool))
correlations = upper_triangle.values.flatten()
avg_correlation = np.mean(np.abs(correlations[~np.isnan(correlations)]))
return 10 * (1 - avg_correlation)
```

This calculation extracts the upper triangle of the correlation matrix to avoid double-counting pairs. The resulting score provides an intuitive measure of diversification effectiveness.

Benchmark Coverage

The system compares portfolios against a comprehensive set of instruments:

- **Equity Indices**: Major market benchmarks across regions and market capitalizations
- **Fixed Income**: Treasury bonds, corporate bonds, and international debt instruments
- **Alternatives**: Hedge fund strategies, managed futures, and market-neutral approaches
- **Commodities**: Energy, metals, agriculture, and broad commodity indices
- **Emerging Technologies**: Cryptocurrency funds and thematic technology ETFs
- **Geographic Exposure**: Country-specific and regional investment vehicles
- **Sector Coverage**: Industry-focused funds across economic sectors

This breadth allows meaningful comparison regardless of portfolio composition or investment strategy.

Data Parsing and Integration

Output Processing

The parse_scores() function extracts information from multiple analysis modules:

```
def parse_scores(output):
scores = {'Portfolio': {}, 'Benchmarks': {}, 'Correlations': {}}

for line in lines:
if 'Portfolio Performance Score:' in line:
score = float(line.split(':')[1].strip().split('/')[0])
elif parsing_correlations and ':' in line:
parts = line.split(':')
corr_value = float(parts[1].strip())
```

The parser handles different output formats from the various analysis scripts, extracting numerical results while managing potential parsing errors.

Technical Implementation Details

Error Handling

The system includes several layers of error management:

- Data validation ensuring minimum sample sizes for statistical significance
- Fallback mechanisms when external data sources are unavailable
- Graceful degradation when individual analysis components fail
- Input validation preventing common user errors

Performance Considerations

- Efficient pandas operations for large time series datasets
- Subprocess isolation preventing memory issues in long-running analysis
- Caching mechanisms for repeated benchmark calculations
- Optimized correlation matrix computations

Output Format

The system generates results reports, displayed in a structured format with multiple sections:

```
=== Combined Analysis Results ===

Portfolio Scores:
--------------------------------------------------
Analysis Type                      Score
--------------------------------------------------
Portfolio Performance Score         8.52
Portfolio Risk Score                7.85
Portfolio Correlation Score         8.23
Portfolio Final Score               8.18
Portfolio Sharpe Ratio              1.45
Portfolio Max Drawdown             -12.30%

Benchmark Scores:
------------------------------------------------------------------------------------------------------------
Symbol    Performance Score    Risk Score    Final Score    Sharpe     Max DD
------------------------------------------------------------------------------------------------------------
SPY              8.21              7.53           7.87        1.32      -15.20%
QQQ              8.85              8.02           8.43        1.58      -18.50%
...

Portfolio Correlations with Benchmarks:
--------------------------------------------------
Symbol     Correlation
--------------------------------------------------
SPY             0.856
QQQ             0.923
...
```

The results are designed to include both raw metrics and normalized scores, allowing users to understand both absolute and relative performance characteristics.

Applications and Extensions

This framework provides a foundation for various financial analysis applications. The modular structure supports additional analysis components, while the scoring system can accommodate different weighting schemes based on investment objectives.

The comprehensive benchmark coverage makes the system applicable to diverse portfolio types, from traditional asset allocation approaches to alternative investment strategies. The scenario analysis component provides risk assessment capabilities relevant to portfolio management and regulatory compliance requirements.

Use Case

This script is useful for investors or analysts who want to:

1. Input a portfolio of stocks and their allocations.
1. Compare the portfolio's performance, risk, and correlations against a comprehensive set of benchmarks.
1. Obtain detailed metrics including Sharpe ratios and maximum drawdown.
1. Save comprehensive results for further analysis or reporting.

Technical Stack

The implementation uses Python with pandas for data manipulation, numpy for numerical computations, and yfinance for market data acquisition. The modular architecture facilitates maintenance and allows individual components to use specialized libraries as needed.

Planned Future Updates

I will be adding the following features to the project:

1. Portfolio Optimization Module

- **Add portfolio optimization** using modern portfolio theory (Markowitz)
- **Suggest optimal weights** to maximize Sharpe ratio or minimize volatility
- **Compare current vs optimal allocation** with expected improvement metrics

2. Risk-Adjusted Performance Metrics

- **Add more risk metrics**: Sortino ratio, Calmar ratio, Value at Risk (VaR)
- **Rolling Sharpe ratio** over different time periods (1M, 3M, 6M, 1Y)
- **Stress testing** against historical market crashes

3. Factor Analysis Integration

- **Fama-French factor exposure** (Market, Size, Value, Momentum)
- **Sector diversification analysis** beyond just correlation
- **Geographic exposure breakdown**

4. Benchmark Customization

- **Dynamic benchmark selection** based on portfolio composition
- **Custom benchmark creation** (weighted average of relevant indices)
- **Peer group comparison** for similar investment styles

5. Performance Attribution

- **Asset contribution analysis** (which holdings drive returns)
- **Timing vs selection attribution**
- **Currency impact analysis** for international exposure

6. Visual Dashboard Generation*

- **Automated chart generation** (performance charts, correlation heatmaps)
- **Risk-return scatter plots** for portfolio vs benchmarks
- **Export to PDF/HTML** reports with visualizations

7. Scenario Analysis*

- **Monte Carlo simulations** for return distributions
- **What-if analysis** (interest rate changes, volatility shocks)
- **Rebalancing impact simulation**

8. Data Quality Enhancements*

- **Data freshness validation** (check for stale prices)
- **Missing data handling** with interpolation/fallback strategies
- **Currency conversion** for multi-currency portfolios

9. Comparative Analytics*

- **Portfolio vs portfolio comparison** (to track multiple strategies)
- **Performance persistence analysis** (to measure how consistent are returns)
- **Drawdown analysis** with recovery time statistics

10. Alert System*

- **Threshold alerts** (volatility spikes, correlation changes)
- **Rebalancing triggers** based on drift from target allocation
- **Performance milestone notifications**

These suggestions focus on incremental value without requiring complete system overhauls. Each could be implemented as separate modules that integrate with your existing port_an.py framework.

Sample Run*

Sample run with an adaptation of Ray Dalio's All Weather portfolio

Portfolio Composition: SPY: 30.0%, TLT: 40.0%, IEF: 15.0%, GLD: 7.5%, DBC: 7.5%

Output Files: portfolio_dashboard.html, portfolio_report.pdf, portfolio_analysis.txt