
**In the Name of God**

Shiraz University
Department of Computer Science and Engineering

# Financial Portfolio Planning Using Operations Research

Course: **Operations Research**
Professor: **Dr. Ziarati**
Teaching Assistant: **Mohammadmehdi Zamani**

Prepared By:

**MohammadMehdi Katibeh - 9932128**

**Yousof Asadi - 9932135**

**Github repo**

# Abstract

This study develops a comprehensive system for optimizing a financial portfolio containing Gold, Bitcoin, Ethereum, and Bonds by integrating time-series forecasting, risk-reward assessment, and mathematical optimization. Using historical price data, the system employs linear regression with engineered features to forecast future asset prices. Subsequently, a risk-based decision framework determines buy/sell signals, which feed into a constraint programming model implemented in MiniZinc to optimally allocate capital under risk and leverage constraints. The system updates capital weekly based on real market outcomes. This report thoroughly documents the data preparation, modeling strategies, implementation details, performance evaluation, and discusses implications and future extensions.


# 1. Introduction

Investment portfolio management aims to maximize returns while managing risks. This project focuses on constructing a dynamic, data-driven portfolio allocation strategy integrating:

- Time-series forecasting for predicting asset prices.
- Risk-reward evaluation to make buy/sell decisions.
- Optimization via constraint programming to allocate capital considering risk tolerance and leverage constraints.
- Capital update mechanisms reflecting actual market behavior to track performance.

The novelty lies in the seamless integration of machine learning forecasts with MiniZinc optimization, providing explainability and flexibility to incorporate constraints.

## 2. Data Handling and Feature Engineering

### 2.1 Data Sources

Raw market data for Gold, Bitcoin, and Ethereum is obtained from historical CSV files containing daily Open, High, Low, Close, and Volume values, with the date column labeled ‘Price’.

The `prepare_data_for_this_week.py` script extracts a rolling window of the most recent 100 days from these raw data files for focused training and prediction.

### 2.2 Loading and Indexing Data

The `preprocessing.py` module uses pandas to:

- Load CSV data with `parse_dates=['Price']` to interpret dates correctly.
- Set the 'Price' column as the datetime index, facilitating time-based operations.

This indexing supports time-lagged feature creation and temporal filtering.

### 2.3 Feature Engineering

Predicting future prices from historical data requires features that capture trends, seasonality, and autocorrelation.

Implemented features include:

**Lagged features**: For each variable (Open, High, Low, Close, Volume), the value from 1, 2, and 5 days prior is added. This helps capture momentum and recent trends.

For example:

```
Open_lag1 = Open price on previous day

Open_lag2 = Open price two days ago

Open_lag5 = Open price five days ago
```

- 

**Moving averages (MA)**: 5-day and 10-day rolling averages smooth out short-term volatility.

```
Open_ma5 = Average Open price over past 5 days

Open_ma10 = Average Open price over past 10 days
```

- 

These engineered features increase the model's ability to recognize patterns beyond raw daily prices.

### 2.4 Target Variable Preparation

The model aims to forecast prices `horizon` days into the future. The target variable is created by shifting the chosen price attribute (`pred`) backward by `horizon` days.

This means:

- For row i, the target is the price at i + horizon.

- The last `horizon` rows are dropped from training as their targets are unknown.

### 2.5 Data Scaling

Feature scaling improves numerical stability in regression:

- StandardScaler fits on training features and transforms both training and prediction sets.

- Target values are similarly scaled.

Scaling ensures features have zero mean and unit variance.

## 3. Forecasting Methodology

### 3.1 Linear Regression Overview

Linear regression models the relationship:

```
y = X * Beta + ε
```

Where:

- y: vector of target prices (future values).

- X: matrix of features (lagged prices, moving averages, intercept).

- Beta: vector of coefficients to be estimated.
- ε: error term.

The least squares solution is obtained via normal equations:

```
(Beta) = (X^T * X)^(-1) * X^T * y
```

### 3.2 MiniZinc Implementation

The regression is formulated as a constraint satisfaction problem in MiniZinc (`linear_regression.mzn`):

- Variables `beta` represent coefficients.

Constraints encode the normal equations as:

For each coefficient j:

```
sum over i ( sum over k (X[k,i] * X[k,j]) * beta[i] ) = sum over k (X[k,j] * y[k])
```

- The model finds Beta satisfying these constraints.

Prediction for new data point `X_predict` is:

```
prediction = sum over j (X_predict[j] * beta[j])
```

This approach enables exact solving and easy integration with subsequent optimization.

---

# 4. Risk-Reward Analysis and Trading Signal Generation

### 4.1 Price Predictions and Actual Prices

For each asset, the system forecasts the minimum and maximum prices in the upcoming period and compares these to current market prices.

### 4.2 Computing Profit Potentials

Potential profits for buying and selling are calculated as:

- Buy profit = (Predicted Max Price - Current Price) / Current Price
- Sell profit = (Current Price - Predicted Min Price) / Current Price

These quantify expected relative gains.

### 4.3 Decision Logic

A simple rule:

- If Buy profit ≥ Sell profit → decide to **buy** (signal 0)
- Else → decide to **sell** (signal 1)

### 4.4 Risk-Reward Ratio

Risk-reward ratio is:

- The absolute ratio of the smaller profit potential over the larger loss potential, capped at 10.

This ratio gauges trade attractiveness considering potential upside vs downside.

### 4.5 Implementation Details

The function `compute_risk_reward(predictions)` performs these steps:

- Retrieves today's prices from CSVs.
- Calculates profit potentials.
- Determines buy/sell signals.
- Returns arrays of signals, risk-reward ratios, and profit/loss magnitudes for use in optimization.

---

## 5. Portfolio Optimization via MiniZinc

### 5.1 Problem Variables

- **Investment amounts** per asset: `x_gold`, `x_btc`, `x_eth`, `x_bond`.
- **Uninvested capital**: `x_nothing`.
- **Leverage factors** for BTC and ETH: integers from 1 to 3.

### 5.2 Objective

Maximize total return considering profits, leverage, and bond interest:

```
total_return =

  x_gold * (1 + GOLD_PROFIT) +

  x_btc * (1 + BTC_PROFIT) * lev_btc +

  x_eth * (1 + ETH_PROFIT) * lev_eth +

  x_bond * (1 + 0.0055 / 4) +

  x_nothing
```

### 5.3 Constraints

Capital conservation:

```
x_nothing = CAPITAL - (x_gold + x_btc + x_eth + x_bond)
```

- Risk tolerance limits based on risk-reward and accuracy:

  ```
  x_asset ≤ CAPITAL * ((RW_asset * ACC_asset) / SOP_RW_ACC)
  ```

- Leverage constraints keep leveraged exposure under a threshold:

  ```
  lev_btc * ML_BTC ≤ 0.9
  ```

```
lev_eth * ML_ETH ≤ 0.9
```

- Allocation must be non-negative and not exceed capital.

### 5.4 MiniZinc Model Structure

The MiniZinc model (`main_model.mzn`) defines:

- Parameters for profits, risk-reward ratios, accuracies, maximum leverage levels.
- Decision variables for investments and leverage.
- Objective to maximize returns.
- Constraints to ensure realistic and safe portfolio composition.

---

# 6. Weekly Capital Update and Tracking

### 6.1 Mechanism

After one week, actual market data is compared to predictions:

- If price movements meet or exceed predicted thresholds favorably → full profit.
- If prices miss thresholds → partial profit (~30%) or loss applied.
- Bond investments accumulate fixed interest.

### 6.2 Tracking Metrics

- Total number of trades.
- Number of correct predictions.
- Accuracy computed as the ratio of correct trades over total trades.

### 6.3 Implementation

The function `update_capital(...)` in `update_capital.py` handles:

- Reading actual weekly price highs/lows.
- Calculating profit/loss per asset.
- Updating capital and tracking statistics.

---

## 7. Complete System Workflow

1. **Data Preparation:** Extract recent 100 days data (`prepare_data_for_this_week.py`).
2. **Preprocessing:** Feature engineering and scaling (`preprocessing.py`).
3. **Forecasting:** Run MiniZinc regression, produce next period predictions.
4. **Risk-Reward Analysis:** Compute buy/sell signals (`compute_risk_reward.py`).
5. **Optimization:** Run MiniZinc model to determine capital allocation.
6. **Capital Update:** Adjust portfolio value based on actual market data (`update_capital.py`).

7. **Performance Tracking:** Record accuracies, capital growth, and risk metrics.

---

# 8. Performance Evaluation and Statistical Analysis

### 8.1 Runtime Performance

- Feature engineering: typically < 5 seconds per asset.
- MiniZinc regression solving: 3-10 seconds depending on data size.
- Portfolio optimization solving: 2-5 seconds.
- Entire weekly iteration cycle: under 30 seconds, feasible for near real-time operation.

### 8.2 Accuracy and Return Metrics

- Accuracy tracked per asset as successful prediction ratio.
- Capital tracked weekly to observe growth.
- Return on Investment (ROI), Sharpe ratio, and drawdown can be calculated as enhancements.

---

# 9. Visualization and Results Interpretation

**Suggested Charts:**

- **Price Forecast vs Actual:** Overlay predicted min/max and actual highs/lows.
- **Capital Growth:** Line graph of capital over time showing portfolio performance.
- **Allocation Breakdown:** Weekly distribution of capital across assets and leverage levels.
- **Accuracy Trends:** Line or bar chart showing prediction success rates.
- **Risk-Reward Profiles:** Scatter plots showing trade attractiveness over time.

These visualizations facilitate intuitive understanding of model behavior and trading effectiveness.

---

## 10. Limitations and Future Work

- **Linear Regression Assumption:** Assumes linear relations, may miss nonlinear market dynamics.
- **Static Risk Limits:** Fixed risk constraints could be enhanced with adaptive methods.
- **Leverage Model Simplification:** More detailed margin and leverage rules could improve realism.

- **Transaction Costs:** Ignored in current model, should be incorporated.

- **Alternative Forecasting Models:** Testing ARIMA, Random Forests, or Neural Networks could improve accuracy.

- **Real-time Data Integration:** Implement streaming data for intraday decision making.

---

## 11. Conclusion

This project illustrates the integration of statistical forecasting, risk-based decision-making, and mathematical optimization in a practical portfolio management system. The modular architecture promotes easy extensibility and transparency. Performance tracking and visualization support ongoing analysis and refinement.

---

## Appendix: Plain Text Formulas

- Normal equations for regression coefficients:

```
(X-transpose multiplied by X) multiplied by Beta =
X-transpose multiplied by y
```

- Predicted price:

```
Predicted price = sum over j of (feature_j * Beta_j)
```

- Buy profit potential:

  ```
  (Predicted max price - Current price) divided by Current price
  ```

- Sell profit potential:

  ```
  (Current price - Predicted min price) divided by Current price
  ```

- Capital allocation constraint:

  ```
  Investment in asset ≤ Capital * (Risk-Reward of asset * Accuracy of asset) divided by sum of all (Risk-Reward * Accuracy)
  ```

- Leverage constraint:

  ```
  Leverage level multiplied by max leverage ≤ 0.9
  ```

- Capital conservation:

  ```
  Uninvested capital = Total capital - sum of invested capital in all assets
  ```