|  |  |
| --- | --- |
| Semester | T.E. Semester V |
| Subject | Artificial Intelligence Lab |
| Subject Professor In-charge | Ms. Rasika Ransing |
| Laboratory | L 011 |

|  |  |
| --- | --- |
| Student Name | Harsh Rawte |
| Roll Number | 22101A0047 |

|  |  |
| --- | --- |
| Experiment  Number | 06 |
| Experiment Title | Consider you are Mutual fund manager. You have to select stocks for the portfolio which maximise the overall gain and minimize any potential losses. |
| Objectives  (Skill Set /  Knowledge  Tested /  Imparted) | Minimax algorithm |
| Theory  Code | Min-Max Algorithm in Share Market Context  Theory  The Min-Max algorithm can be adapted to help in making investment decisions in the stock market by evaluating potential investment opportunities. In this context, the algorithm helps determine the best stocks to invest in by analyzing various performance metrics and predicting the worst-case scenarios for each investment option.Thus gives us best stocks taking best and worst case into consideration.  import numpy as npfrom scipy.optimize import minimizeclass Stock: def \_\_init\_\_(self, name, current\_price, all\_time\_high, all\_time\_low, price\_earnings\_ratio, dividend\_yield, risk\_factor): self.name = name self.current\_price = current\_price self.all\_time\_high = all\_time\_high self.all\_time\_low = all\_time\_low self.price\_earnings\_ratio = price\_earnings\_ratio self.dividend\_yield = dividend\_yield self.risk\_factor = risk\_factor self.normalized\_price\_score = 0 self.normalized\_pe\_score = 0 self.normalized\_dividend\_score = 0 self.normalized\_risk\_score = 0 self.score = 0 def \_\_repr\_\_(self): return (f"Name: {self.name}\n" f"Current Price: {self.current\_price:.2f}\n" f"All-Time High: {self.all\_time\_high:.2f}\n" f"All-Time Low: {self.all\_time\_low:.2f}\n" f"Price/Earnings (P/E) Ratio: {self.price\_earnings\_ratio:.2f}\n" f"Dividend Yield: {self.dividend\_yield:.2f}%\n" f"Risk Factor: {self.risk\_factor:.2f}\n" f"Normalized Price Score: {self.normalized\_price\_score:.2f}\n" f"Normalized P/E Score: {self.normalized\_pe\_score:.2f}\n" f"Normalized Dividend Score: {self.normalized\_dividend\_score:.2f}\n" f"Normalized Risk Score: {self.normalized\_risk\_score:.2f}\n" f"Score: {self.score:.2f}\n")def normalize\_data(stocks, factor): values = [getattr(stock, factor) for stock in stocks] min\_value = min(values) max\_value = max(values) if max\_value == min\_value: return [0] \* len(values) # Avoid division by zero if all values are the same return [(value - min\_value) / (max\_value - min\_value) for value in values]def calculate\_normalized\_scores(stocks): # Normalize each factor price\_scores = normalize\_data(stocks, 'current\_price') pe\_scores = normalize\_data(stocks, 'price\_earnings\_ratio') dividend\_scores = normalize\_data(stocks, 'dividend\_yield') risk\_scores = normalize\_data(stocks, 'risk\_factor') for i, stock in enumerate(stocks): stock.normalized\_price\_score = 1 - price\_scores[i] # Lower price is better stock.normalized\_pe\_score = pe\_scores[i] # Higher P/E ratio is better stock.normalized\_dividend\_score = dividend\_scores[i] # Higher dividend yield is better stock.normalized\_risk\_score = 1 - risk\_scores[i] # Lower risk is better # Combine normalized scores with weights (adjust weights as needed) stock.score = (stock.normalized\_price\_score \* 0.3 + stock.normalized\_pe\_score \* 0.2 + stock.normalized\_dividend\_score \* 0.2 + stock.normalized\_risk\_score \* 0.3)def portfolio\_optimization(stocks, num\_stocks): # Objective function to minimize (negative of total score) def objective(weights): return -sum(stock.score \* weight for stock, weight in zip(stocks, weights)) # Constraints (weights sum to 1) constraints = [{'type': 'eq', 'fun': lambda weights: sum(weights) - 1}] # Bounds (weights between 0 and 1) bounds = [(0, 1) for \_ in stocks] # Initial guess (equal weight) initial\_weights = [1 / len(stocks)] \* len(stocks) result = minimize(objective, initial\_weights, method='SLSQP', bounds=bounds, constraints=constraints) optimized\_weights = result.x # Select stocks based on optimized weights selected\_stocks = [stock for stock, weight in zip(stocks, optimized\_weights) if weight > 0] # Sort by score selected\_stocks.sort(key=lambda x: x.score, reverse=True) return selected\_stocks[:num\_stocks]def main(): # Example data print("Harsh Rawte 22101A0047") stocks = [ Stock("Nestle India", 2545.05, 3000.00, 1280.00, 75.85, 0.62, 0.15), Stock("P & G Hygiene", 17093.65, 20000.00, 870.00, 74.44, 0.62, 0.10), Stock("Colgate-Palmolive", 3576.85, 4000.00, 97.00, 68.80, 1.34, 0.20), Stock("Lloyds Metals", 756.15, 900.00, 5.00, 28.31, 0.13, 0.30), Stock("Life Insurance", 1079.75, 1200.00, 60.00, 16.30, 0.94, 0.25), Stock("Tata Consultancy Services", 4531.65, 5000.00, 27.00, 34.43, 1.22, 0.10), Stock("Coal India", 532.15, 600.00, 24.00, 8.92, 4.79, 0.35), Stock("Castrol India", 273.25, 300.00, 104.00, 30.53, 2.78, 0.40), Stock("Indian Railway Catering and Tourism Corporation", 935.80, 1100.00, 315.00, 62.86, 0.71, 0.20), Stock("Gillette India", 8279.20, 8500.00, 201.00, 69.64, 0.56, 0.15), Stock("GlaxoSmithKline Pharmaceuticals", 3077.45, 3200.00, 292.00, 67.67, 1.07, 0.25), Stock("Britannia Industries", 5837.80, 6000.00, 7.00, 63.79, 1.27, 0.20), Stock("Cams Services", 4437.00, 4500.00, 130.00, 57.11, 1.04, 0.15), Stock("Motherson Wiring", 71.93, 100.00, 16.00, 47.95, 1.11, 0.35), Stock("CG Power and Industrial Solutions", 728.10, 750.00, 5.00, 123.73, 0.18, 0.40) ] # Get user input for number of stocks to suggest num\_stocks = int(input("Enter the number of stocks to suggest: ")) # Calculate normalized scores calculate\_normalized\_scores(stocks) # Optimize portfolio top\_stocks = portfolio\_optimization(stocks, num\_stocks) # Print the recommended stocks print("\nTop Stocks Recommended:") for stock in top\_stocks: print(stock) print('-' \* 40) # Separator for better readabilityif \_\_name\_\_ == "\_\_main\_\_": main() |
| Output |  |
| Conclusion | The code implements a stock selection strategy by evaluating each stock based on financial metrics such as current price, P/E ratio, dividend yield, and risk factor. It normalizes these factors to create comparable scores for each stock, where lower prices and risk factors are preferred, while higher P/E ratios and dividend yields are favored. The portfolio\_optimization function uses these scores to determine optimal stock weights, aiming to maximize the portfolio’s overall score. By selecting stocks with the highest scores, the approach ensures a balanced investment strategy that aligns with financial goals and risk tolerance. This data-driven methodology helps investors make informed decisions, potentially improving investment returns while managing risk. |