Stock Analysis Script - Product Requirement Document

1. Project Overview

A Python script that analyzes a single stock ticker, provides key financial metrics, and generates visualizations of historical performance.

2. Technical Requirements

2.1 Environment

- **Operating System**: macOS
- **IDE**: Cursor
- **Package Management**: conda-forge via Homebrew
- **Python Version**: 3.9+

2.2 Dependencies

```
conda create -n stock-analysis python=3.9
conda activate stock-analysis
conda install -c conda-forge yfinance pandas matplotlib numpy seaborn
```

3. Functional Requirements

3.1 Core Features

- Stock data retrieval via API (Yahoo Finance)
- Historical price analysis
- Basic technical indicators calculation
- Performance metrics generation
- Data visualization
- Save results to files

3.2 User Inputs

- Stock ticker symbol
- Analysis timeframe (e.g., 1mo, 3mo, 6mo, 1y, 5y)
- Optional: Specific indicators to calculate

3.3 Outputs

- Summary statistics (mean, std, min, max, etc.)

- Technical indicators (SMA, EMA, MACD, RSI)
- Performance metrics (returns, volatility, Sharpe ratio)
- Price charts and indicator visualizations
- CSV export of calculated metrics

4. Code Structure

4.1 Main Modules

- `stock_analyzer.py` - Core script with main function
- `data_fetcher.py` - Data retrieval functions
- `indicators.py` - Technical analysis functions
- `visualizer.py` - Plotting and chart generation
- `utils.py` - Helper functions

4.2 Class Design

python

```python
class StockAnalyzer:
    """Main class for analyzing stock data"""

    def __init__(self, ticker, period="1y"):
        """Initialize with stock ticker and analysis period"""

    def fetch_data(self):
        """Get stock data from Yahoo Finance"""

    def calculate_indicators(self):
        """Calculate technical indicators"""

    def calculate_metrics(self):
        """Calculate performance metrics"""

    def visualize(self):
        """Create visualizations"""

    def save_results(self, path="./results"):
        """Save results to files"""

    def run_analysis(self):
        """Run the full analysis pipeline"""
```

5. Implementation Phases

5.1 Phase 1: Basic Setup

- Set up environment and dependencies
- Implement data fetching
- Create basic visualizations

5.2 Phase 2: Core Analysis

- Implement technical indicators
- Calculate performance metrics
- Generate comprehensive visualizations

5.3 Phase 3: Refinement

- Add error handling
- Optimize code performance
- Enhance user interface
- Add additional analysis options

6. Error Handling

- Handle network connectivity issues
- Validate user inputs
- Manage API limits and timeouts
- Create informative error messages

7. Testing Plan

- Unit tests for indicator calculations
- Integration test for data pipeline
- Manual tests for visualization accuracy
- Edge case tests for unusual market data

8. Usage Example

```python
# Example usage
if __name__ == "__main__":
    # Parse command line arguments
    parser = argparse.ArgumentParser(description="Stock Analysis Tool")
    parser.add_argument("ticker", type=str, help="Stock ticker symbol")
    parser.add_argument("--period", type=str, default="1y", help="Analysis timeframe (
    parser.add_argument("--output", type=str, default="./results", help="Output direct
    args = parser.parse_args()

    # Create analyzer and run
    analyzer = StockAnalyzer(args.ticker, args.period)
    analyzer.run_analysis()
    analyzer.save_results(args.output)
```

9. Future Enhancements

- Multiple stock comparison
- Sentiment analysis integration
- Export to interactive dashboard
- Automated trading signals
- Backtesting capabilities

10. Development Timeline

- Setup and basic features: 1-2 days
- Core analysis implementation: 2-3 days
- Testing and refinement: 1-2 days