# Mini Strategy Analysis Project - Documentation

## 1. Project Overview

The goal of this project was to develop a Python program to analyze breakout trading strategies. The thesis tested was: if a stock's daily volume exceeds 200% of its average daily volume over the last 20 days and the stock price rises by at least 2% compared to the previous day, it may indicate a breakout. The program buys the stock on the breakout day and holds it for 10 days before selling.

## 2. Project Requirements

The program needed to meet the following requirements:

1. Provide a web interface for user input.

2. Identify breakout points where:

- Daily volume exceeds 200% of the 20-day average.

- Stock price increases by at least 2%.

3. Buy on breakout days and sell after a 10-day holding period.

4. Generate a downloadable CSV report with breakout days and returns.

5. Ensure no lookahead bias and maintain data accuracy.

Sample Thought Process Section

## Thought Process and Design Decisions

**Initial Planning:**

I began by understanding the breakout strategy requirement. My goal was to build a modular solution so that additional strategies could be added easily. I decided to use Flask for the web interface due to its simplicity and quick deployment capabilities.

**Data Retrieval:**

I chose yfinance for fetching historical stock data because it provides accurate data with minimal coding effort. I ensured to fetch data only within the specified date range to avoid lookahead bias.

**Breakout Detection:**

To identify breakouts, I considered volume spikes and price changes. I used the 20-day average volume as a baseline to avoid short-term noise and ensured that price changes were calculated as percentages to make the strategy adaptable to various stock price levels.

**Adding Risk Management:**

To make the breakout strategy more practical, I added stop-loss and take-profit conditions. This approach helps mitigate risk by capping potential losses and securing profits.

**Visualization:**

I chose Plotly for creating interactive charts. This allows users to explore the buy and sell points dynamically, making the data more intuitive and actionable.

**DETAILED EXPLANATION OF PROJECT**

## 3. Implementation Steps

3.1. Web Application Setup

I used Flask to create a web application with input fields for:

- Ticker symbol

- Start and end dates

- Volume breakout threshold

- Price change threshold

- Holding period

The Flask app handles user input and generates reports dynamically.

### 3.2. Data Retrieval

Used the `yfinance` library to fetch historical stock data:

- **Function**: `yf.Ticker(ticker).history(start=start_date, end=end_date)`

- Retrieved closing prices, volumes, and other relevant data.

### 3.3. Technical Indicators

Calculated the following indicators using Pandas:

- **20-Day Average Volume**: `data['Volume'].rolling(window=20).mean()`

- **10-Day and 50-Day SMAs**: Used for the SMA Crossover Strategy.

### 3.4. Breakout Detection

Identified breakout points based on:

1. **Volume Breakout:** Volume exceeds the specified threshold (e.g., 200%).

2. **Price Breakout:** Price increases by the specified percentage (e.g., 2%).

**Logic:**

- `data['VolumeBreakout'] = data['Volume'] > (threshold * data['20DayAvgVolume'])`

- `data['PriceChange'] = data['Close'].pct_change() * 100`

- `data['PriceBreakout'] = data['PriceChange'] > price_change_threshold`

### 3.5. Return Calculation

Function `calculate_returns()` computes returns for each breakout point:

- **Buy** at the close of the breakout day.

- **Sell** after the specified holding period (e.g., 10 days).

- Returns calculated as: `((sell_price - buy_price) / buy_price) * 100`.

### 3.6. Error Handling

Handled scenarios where data might be missing due to holidays or invalid tickers. Displayed appropriate messages when no data was available.

## 4. Additional Efforts

### 4.1. Multiple Trading Strategies

Implemented the following additional strategies:

1. **SMA Crossover Strategy**: Buys when the 10-day SMA crosses above the 50-day SMA.

2. **Breakout Strategy with Risk Management**: Added stop-loss and take-profit conditions.

3. **Machine Learning Predicted Breakouts**: Used a Random Forest Classifier to predict breakout points.

### 4.2. Performance Metrics

Calculated metrics for each strategy:

- **Win Rate**: Percentage of profitable trades.

- **Average Return**: Mean return across trades.

- **Maximum Drawdown**: Largest peak-to-trough decline.

### 4.3. Interactive Visualizations

Used Plotly to create interactive charts showing:

- **Buy Points**: Green triangles.

- Sell Points: Red triangles.

Saved charts as HTML files for easy viewing.

### 4.4. Downloadable Reports

Generated a downloadable CSV report combining results from all strategies.

Reports include breakout dates, buy prices, sell prices, and returns.

## 5. Roadblocks

### 5.2. Roadblocks

1. Data Gaps: Handled missing data due to holidays.

2. Lookahead Bias: Ensured calculations used only historical data.

3. ML Model Performance: Balanced model complexity with available data.

4. Deployement Issues: Initially tried deploying on Heroku, but encountered challenges due to ephemeral storage and file system limitations. Switched to **Render** for deployment.