Contents

[Data\_fetcher.py 2](#_Toc187844108)

[Overview 2](#_Toc187844109)

[Strengths 2](#_Toc187844110)

[Suggested Improvements 3](#_Toc187844111)

[Plot Trend Analysis Script Documentation 5](#_Toc187844112)

[Trend Pre-Processing Script Documentation 8](#_Toc187844113)

[Countermoves and Reversal Analysis Script Documentation 11](#_Toc187844114)

[Countermoves Analysis Script Documentation 14](#_Toc187844115)

# 

Folder structure (order by process):

* Beta\_stock: list of trade stocks
* Raw: tickers with time stamps, High, Low, Open, Close, and Volume values.
* Trend: raw files with trend analysis (Uptrend or Downtrend)
* Counter\_reversal: trend files with countermoves and reversal indications
* Stats: stocks behavioral statistics

# 

# Data\_fetcher.py

## Overview

This script is a data fetching module that uses `yfinance` to download stock market data. It's well-structured with proper error handling, logging, and configuration management.

Key Components

1. Class Structure

1.```python

2. class DataFetcher:

3. def \_\_init\_\_(self, save\_dir: Optional[Path] = None):

4. # Initializes paths and settings from config

5. # Sets up market hours, intervals, and required columns

6. ```

1. Main Methods

1. ```python

2. def fetch\_date\_range(ticker, start\_date, end\_date)

3. def fetch\_intraday\_data(ticker, date, interval)

4. def \_save\_data(data, ticker, date)

5. def validate\_data(df)

6. ```

## Strengths

1. Robust Error Handling

1. ```python

2. try:

3. # Operation code

4. except Exception as e:

5. logger.error(f"Error: {str(e)}")

6. return None

7. ```

1. Proper Logging

1. ```python

2. logger = logging.getLogger(\_\_name\_\_)

3. logger.info(f"Data saved to: {filepath}")

4. logger.error(f"Error fetching data: {str(e)}")

5. ```

1. Type Hints

1. ```python

2. def fetch\_date\_range(self,

3. ticker: str,

4. start\_date: Union[str, datetime],

5. end\_date: Optional[Union[str, datetime]] = None) -> Dict[str, pd.DataFrame]:

6. ```

1. Configuration Management

1. ```python

2. from nne\_strategy.config.config import config

3. start\_time = config.get('market', 'hours', 'start')

4. end\_time = config.get('market', 'hours', 'end')

5. ```

### Suggested Improvements

1. Add Data Validation

1. ```python

2. def validate\_data(self, df: pd.DataFrame) -> bool:

3. # Add more validation checks:

4. # 1. Check for negative prices

5. if (df[['Open', 'High', 'Low', 'Close']] < 0).any().any():

6. logger.error("Found negative prices in data")

7. return False

8.

9. # 2. Check for High < Low

10. if (df['High'] < df['Low']).any():

11. logger.error("Found High price less than Low price")

12. return False

13. ```

1. Add Rate Limiting

1. ```python

2. from time import sleep

3.

4. def fetch\_intraday\_data(self, ...):

5. # Add rate limiting

6. sleep(0.5) # Add delay between requests

7. ```

1. Add Data Caching

1. ```python

2. def fetch\_intraday\_data(self, ticker: str, date: str, interval: Optional[str] = None) -> Optional[pd.DataFrame]:

3. # Check if data already exists

4. cache\_path = self.save\_dir / f"{ticker}\_{date}.csv"

5. if cache\_path.exists():

6. return pd.read\_csv(cache\_path)

7. ```

1. Add Progress Tracking

1. ```python

2. from tqdm import tqdm

3.

4. def fetch\_date\_range(self, ...):

5. for date in tqdm(dates, desc=f"Fetching {ticker} data"):

6. # existing code

7. ```

**Usage Example**

1. ```python

2. # Initialize fetcher

3. fetcher = DataFetcher()

4.

5. # Fetch single day

6. data = fetcher.fetch\_intraday\_data("AAPL", "2024-01-15")

7.

8. # Fetch date range

9. results = fetcher.fetch\_date\_range(

10. ticker="AAPL",

11. start\_date="2024-01-01",

12. end\_date="2024-01-15"

13. )

14. ```

**Command Line Usage**

1. ```bash

2. python data\_fetcher.py 20250115 20250120 --ticker NNE

3. ```

**Recommendations**

1. Add Retry Logic

1. ```python

2. from retrying import retry

3.

4. @retry(stop\_max\_attempt\_number=3, wait\_fixed=2000)

5. def fetch\_intraday\_data(self, ...):

6. # existing code

7. ```

1. Add Data Quality Checks

1. ```python

2. def check\_data\_quality(self, df: pd.DataFrame) -> bool:

3. # Check for price jumps

4. price\_changes = df['Close'].pct\_change().abs()

5. if (price\_changes > 0.1).any(): # 10% threshold

6. logger.warning("Large price changes detected")

7. ```

1. Add Memory Management

1. ```python

2. def fetch\_date\_range(self, ...):

3. # Clear memory periodically

4. if idx % 10 == 0: # Every 10 dates

5. import gc

6. gc.collect()

7. ```

1. Add Parallel Processing Option

1. ```python

2. from concurrent.futures import ThreadPoolExecutor

3.

4. def fetch\_parallel(self, tickers: List[str], date: str):

5. with ThreadPoolExecutor(max\_workers=3) as executor:

6. futures = [executor.submit(self.fetch\_intraday\_data, ticker, date)

7. for ticker in tickers]

8. ```

23/01/2025 Code Enhancement:  
  
Task: Add a "Multiple Stock Fetching" Feature to the "Data Processing" Screen  
  
**Requirements**:   
1. Enable users to upload an Excel file containing a column named {**"Ticker"}**.   
2. Implement a process to read the uploaded Excel file and extract all stock tickers listed in the "Ticker" column.   
3. Allow the fetching process to run for either:   
- A **single date**, or

- A **specified date range**.   
4. Ensure that the retrieved stock data is saved in the following directory:   
*D:\NNE\_strategy\nne\_strategy\data\raw\\{stock\_name}* If the folder does not exist, create it.  
5. Save output files using the naming convention:   
*raw\_{stock\_name}\_YYYYMMDD.csv*

# Plot Trend Analysis Script Documentation

**Overview**

This script analyzes stock price data to identify trends by detecting local minima and maxima, and generates both visual and data outputs of the trend analysis.

**File Structure**

1. ```python

2. nne\_strategy/data/plot\_trend\_analysis.py

3. ```

**Dependencies**

1. ```python

2. import pandas as pd

3. import numpy as np

4. import matplotlib.pyplot as plt

5. from scipy.signal import argrelextrema

6. import os

7. ```

**Configuration**

1. ```python

2. # Input/Output Paths

3. input\_folder = r'D:\NNE\_strategy\nne\_strategy\data\raw'

4. output\_folder = r'D:\NNE\_strategy\nne\_strategy\data\stock\_trend\_complete'

5. ```

6.

**Core Functions**

1. ensure\_min\_max\_alternating()

``` 1. python

2. def ensure\_min\_max\_alternating(selected\_points):

3. """

4. Ensures that minimum and maximum points alternate in the sequence.

5.

6. Args:

7. selected\_points (DataFrame): Points identified as local minima/maxima

8.

9. Returns:

10. DataFrame: Filtered points with alternating min/max values

11. """

12. ```

1. process\_all\_files()

``` 1. python

2. def process\_all\_files(input\_folder, output\_folder):

3. """

4. Main processing function that handles all CSV files in the input folder.

5.

6. Key Steps:

7. 1. Load and preprocess data

8. 2. Identify local minima and maxima

9. 3. Group and filter extrema points

10. 4. Generate trend analysis

11. 5. Create visualizations

12. 6. Save results

13. """

14. ```

**Processing Pipeline**

1. Data Loading

- Reads CSV files from input directory

- Converts datetime strings to pandas datetime objects

1. Extrema Detection

1. ```1. python

2. 2. # Find local minima and maxima using scipy

3. 3. df['min'] = df.iloc[argrelextrema(df['Close'].values, np.less\_equal, order=n)[0]]['Close']

4. 4. df['max'] = df.iloc[argrelextrema(df['Close'].values, np.greater\_equal, order=n)[0]]['Close']

5. 5.

6. 6.

7. ```

1. Point Selection

- Groups extrema by hour

- Selects highest maxima and lowest minima per hour

- Adds additional points at 1/3 and 2/3 of the timeline

1. Trend Analysis

- Identifies UpTrend and DownTrend patterns

- Records trend start/end times and prices

- Applies trend labels to the dataset

1. Output Generation

- Creates CSV files with trend information

- Generates visualization plots

**Outputs**

1. CSV Files

1. ```

2. trend\_analysis\_NNE\_YYYYMMDD.csv

3. ```

- Contains original data plus trend labels

- One file per input day

1. Plot Files

1. ```

2. trend\_analysis\_plot\_NNE\_YYYYMMDD.png

3. ```

- Visual representation of price movement

- Includes:

- Close price line

- Local minima/maxima points

- Trend lines

- Legend and grid

**Usage**

```1. python

2. if \_\_name\_\_ == "\_\_main\_\_":

3. process\_all\_files(input\_folder, output\_folder)

4. ```

**Error Handling**

- Creates directories if they don't exist

- Prints processing status for each file

- Uses try-except blocks for file operations

# Trend Pre-Processing Script Documentation

**Overview**

This script preprocesses trend analysis data by handling missing values and generating reports. It supports both single file and batch processing modes.

**File Structure**

1. ```python

2. nne\_strategy/trend\_pre\_processing.py

3. ```

**Dependencies**

1. ```python

2. import pandas as pd

3. import os

4. from datetime import datetime

5. ```

**Configuration**

1. ```python

2. # Base Paths

3. base\_path = r'D:\NNE\_strategy\nne\_strategy\data\stock\_trend\_complete'

4. output\_base\_path = r'D:\NNE\_strategy\nne\_strategy\data\preprocess\_trend\_data'

5. ```

**Core Functions**

1. preprocess\_trend\_data()

1. ```python

2. def preprocess\_trend\_data(input\_file, output\_file, report\_file):

3. """

4. Process a single trend data file

5.

6. Args:

7. input\_file (str): Path to input CSV file

8. output\_file (str): Path to save processed CSV

9. report\_file (str): Path to save processing report

10.

11. Processing Steps:

12. 1. Load CSV data with datetime parsing

13. 2. Check initial missing values

14. 3. Interpolate numerical missing values

15. 4. Fill missing trend values

16. 5. Generate processing report

17. 6. Save processed data and report

18. """

19. ```

20.

1. process\_all\_files()

1. ```python

2. def process\_all\_files(base\_path, output\_base\_path):

3. """

4. Batch process all trend analysis files in directory

5.

6. Args:

7. base\_path (str): Directory containing input files

8. output\_base\_path (str): Directory for processed files

9.

10. Returns:

11. int: Number of files processed

12.

13. Processing Steps:

14. 1. Create output directory

15. 2. Process all matching CSV files

16. 3. Generate individual reports

17. 4. Track processing statistics

18. """

19. ```

**File Naming Conventions**

1. Input Files

2. trend\_analysis\_NNE\_YYYYMMDD.csv

1. Output Files

2. trend\_analysis\_pp\_NNE\_YYYYMMDD.csv

3. trend\_analysis\_pp\_NNE\_YYYYMMDD\_report.txt

**Processing Steps**

1. Data Loading

- Reads CSV with datetime parsing

- Validates file existence and format

1. Missing Value Analysis

1. ```python

2. # Before processing check

3. missing\_before = df.isnull().sum()

4.

5. # After processing check

6. missing\_after = df.isnull().sum()

7. ```

1. Data Cleaning

1. ```python

2. # Numerical columns

3. df.interpolate(method='linear', inplace=True)

4.

5. # Trend column

6. df['Trend'].fillna(method='ffill', inplace=True)

7. ```

1. Report Generation

- Documents missing values before/after

- Saves processing statistics

- Creates individual file reports

**Usage Modes**

1. Single File Processing

1. ```python

2. # User inputs date (YYYYMMDD)

3. # Processes specific file for that date

4. ```

1. Batch Processing

1. ```python

2. # Processes all matching files in directory

3. # Generates reports for each file

4. ```

**Error Handling**

1. File Operations

- Creates missing directories

- Validates file existence

- Handles file read/write errors

Processing Errors

1. ```python

2. try:

3. # Processing logic

4. except Exception as e:

5. print(f"Error processing {file}: {str(e)}")

6. ```

**Output Structure**

1. Processed Data File

- Original data with filled missing values

- Interpolated numerical values

- Forward-filled trend values

1. Report File

- Missing value statistics before processing

- Missing value statistics after processing

- Processing completion status

# Countermoves and Reversal Analysis Script Documentation

**Overview**

This script analyzes preprocessed trend data to identify market reversals and countermoves within established trends.

File Structure

1. ```python

2. nne\_strategy/countermoves\_reversal\_analysis.py

3. ```

4.

Dependencies

1. ```python

2. import pandas as pd

3. import os

4. from datetime import datetime

5. ```

6.

Configuration

1. ```python

2. # Base Paths

3. base\_path = r'D:\NNE\_strategy\nne\_strategy\data\preprocess\_trend\_data'

4. output\_base\_path = r'D:\NNE\_strategy\nne\_strategy\data\counter\_riversal\_analysis'

5. ```

Core Functions

1. analyze\_trends()

2. def analyze\_trends(input\_file, output\_file):

3. """

4. Analyzes trend data to identify Reversals and Countermoves

5.

6. Logic:

7. 1. Reversal: When trend direction changes (Up→Down or Down→Up)

8. 2. Countermove: Price movement against current trend direction

9.

10. Args:

11. input\_file (str): Path to preprocessed trend data CSV

12. output\_file (str): Path to save analysis results

13.

14. Returns:

15. bool: True if analysis completed successfully

16. """

1. process\_all\_files()

2. def process\_all\_files(base\_path, output\_base\_path):

3. """

4. Batch processes all trend files in directory

5.

6. Args:

7. base\_path (str): Input directory path

8. output\_base\_path (str): Output directory path

9.

10. Returns:

11. int: Number of successfully processed files

12. """

**Analysis Logic**

1. Reversal Detection

1. ```python

2. # Identifies trend direction changes

3. if (current\_trend == 'UpTrend' and previous\_trend == 'DownTrend') or \

4. (current\_trend == 'DownTrend' and previous\_trend == 'UpTrend'):

5. df.loc[i, 'Action'] = 'Reversal'

6. ```

1. Countermove Detection

1. ```python

2. # Identifies price movements against trend

3. elif current\_trend == previous\_trend:

4. if (current\_trend == 'UpTrend' and current\_close < previous\_close) or \

5. (current\_trend == 'DownTrend' and current\_close > previous\_close):

6. df.loc[i, 'Action'] = 'Countermove'

7. ```

**File Naming Conventions**

1. Input Files

2. trend\_analysis\_pp\_NNE\_YYYYMMDD.csv

2. Output Files

2. counter\_riversal\_NNE\_YYYYMMDD.csv

**Processing Modes**

1. Single File Mode

2. # Process specific date

3. analyze\_trends(input\_file, output\_file)

2. Batch Processing Mode

2. # Process all files in directory

3. process\_all\_files(base\_path, output\_base\_path)

**Error Handling**

1. Input Validation

2. # Required columns check

3. required\_columns = ['Datetime', 'Close', 'Trend']

4. if not all(col in df.columns for col in required\_columns):

5. raise ValueError(f"Missing required columns")

2. Date Format Validation

2. # Date format check

3. datetime.strptime(date\_input, '%Y%m%d')

3. File Operations

- Directory existence checks

- File existence validation

- Error collection and reporting

**Output Structure**

1. Data Columns

- Datetime: Timestamp

- Close: Price at close

- Trend: Current trend direction

- Action: Identified pattern (Reversal/Countermove)

2. Processing Summary

- Total files processed

- Error count and details

- Processing status for each file

Usage Examples

1. Single File Processing

2. # Run script and select mode 1

3. Enter the date for analysis (YYYYMMDD): 20240101

1. Batch Processing

2. # Run script and select mode 2

3. # Processes all files in input directory

# Countermoves Analysis Script Documentation

**Overview**

This script analyzes market countermoves by processing trading data to identify patterns, calculate statistics, and categorize movements based on their characteristics.

**File Structure**

2. nne\_strategy/analyze\_countermoves.py

Dependencies

1. ```python

2. import pandas as pd

3. import numpy as np

4. import json

5. import glob

6. import os

7. ```

**Core Functions**

1. read\_data(csv\_path: str) -> pd.DataFrame

1. ```python

2. def read\_data(csv\_path: str) -> pd.DataFrame:

3. """

4. Loads and prepares trading data for analysis

5.

6. Args:

7. csv\_path (str): Path to input CSV file

8.

9. Returns:

10. pd.DataFrame: Processed DataFrame with:

11. - Parsed datetime

12. - Sorted by time

13. - Reset index

14. """

15. ```

1. find\_countermoves(df: pd.DataFrame)

1. ```python

2. def find\_countermoves(df: pd.DataFrame):

3. """

4. Identifies consecutive countermove segments in the data

5.

6. Process:

7. 1. Scans for 'Countermove' actions

8. 2. Groups consecutive countermoves

9. 3. Handles edge cases

10.

11. Returns:

12. list: Tuples of (start\_idx, end\_idx) for each countermove segment

13. """

14.

1. analyze\_countermove\_segments(df: pd.DataFrame, segments: list)

1. ```python

2. def analyze\_countermove\_segments(df: pd.DataFrame, segments: list):

3. """

4. Calculates key metrics for each countermove segment

5.

6. Metrics Calculated:

7. - StartTime: Beginning of countermove

8. - EndTime: End of countermove

9. - Duration: Length in minutes

10. - PriceAction: Total price change

11. - PricePct: Percentage price change

12. - Volume: Total volume during countermove

13.

14. Returns:

15. pd.DataFrame: Analysis results for each segment

16. """

1. categorize\_countermoves(countermove\_df: pd.DataFrame, group\_by='PricePct')

``` 1. python

2. def categorize\_countermoves(countermove\_df: pd.DataFrame, group\_by='PricePct'):

3. """

4. Categorizes countermoves and calculates group statistics

5.

6. Categories:

7. - Small: Below 33rd percentile

8. - Medium: Between 33rd and 66th percentile

9. - Large: Above 66th percentile

10.

11. Statistics:

12. - AvgDuration

13. - AvgPriceAction

14. - AvgPricePct

15. - AvgVolume

16.

17. Returns:

18. tuple: (Categorized DataFrame, Group statistics DataFrame)

19. """

20.

**Data Processing Pipeline**

1. Data Loading

1. ```python

2. directory\_path = r"D:\NNE\_strategy\nne\_strategy\data\counter\_riversal\_analysis"

3. csv\_files = glob.glob(os.path.join(directory\_path, '\*.csv'))

4. ```

5.

1. Countermove Identification

1. ```python

2. segments = find\_countermoves(combined\_df)

3. countermove\_df = analyze\_countermove\_segments(combined\_df, segments)

4. ```

1. Direction-based Analysis

1. ```python

2. pos\_df = countermove\_df[countermove\_df['PriceAction'] > 0].copy()

3. neg\_df = countermove\_df[countermove\_df['PriceAction'] < 0].copy()

1. Statistical Analysis

1. ```python

2. \_, pos\_grouped\_stats = categorize\_countermoves(pos\_df, 'PricePct')

3. \_, neg\_grouped\_stats = categorize\_countermoves(neg\_df, 'PricePct')

4. ```

5.

Output Format

**JSON Structure**

``` 1. json

2. {

3. "positive": [

4. {

5. "SizeGroup": "Small/Medium/Large",

6. "AvgDuration": float,

7. "AvgPriceAction": float,

8. "AvgPricePct": float,

9. "AvgVolume": float

10. }

11. ],

12. "negative": [

13. {

14. "SizeGroup": "Small/Medium/Large",

15. "AvgDuration": float,

16. "AvgPriceAction": float,

17. "AvgPricePct": float,

18. "AvgVolume": float

19. }

20. ]

21. }

22. ```

**Usage**

1. ```bash

2. python analyze\_countermoves.py

3. ```

**Output Files**

- Location: `{directory\_path}/countermove\_analysis.json`

- Contains: Statistical analysis of positive and negative countermoves

- Format: JSON with grouped statistics