**Python Project Directory Structure**

Below is the folder structure for the AI trading bot, designed for PythonAnywhere compatibility. All Python modules are stored in a dedicated directory, and the historical analysis JSON file resides in the user's home directory.

**A screenshot of a computer code

AI-generated content may be incorrect.**

* **Run Command**: Execute python /home/gmafanasiev/ai\_trading\_bot/main.py in a PythonAnywhere console or scheduled task.
* **Environment Variables**: ALPACA\_API\_KEY, ALPACA\_API\_SECRET, and XAI\_API\_KEY are set in .env
* **Log File**: Logs are written to /home/gmafanasiev/ai\_trading\_bot/trade\_alpaca\_grok4.log.
* **JSON File**: Historical data is read from  /home/gmafanasiev/historical\_analysis\_APPL.json

+-----------------+

| main.py | 1. Start

+-----------------+

|

v

+-----------------+ +-----------------+

| config.py |<---| .env | (API keys)

| (SIMULATION\_MODE| +-----------------+

| thresholds, |

| open positions)|

+-----------------+

|

v

+-----------------+

| logging\_utils.py|-------> [trade\_alpaca\_grok4.log] (overwrite)

| (setup logging) |

+-----------------+

|

v

+-----------------+

| clients.py |-------> [Alpaca API or Mock]

| (Alpaca clients)| (real/simulated data)

+-----------------+

|

v

+-----------------+ +-----------------+

| trading\_loop.py |----| equity\_plot.png | (overwrite)

| (main loop) | +-----------------+

+-----------------+

| +-----------------+

| | trade\_analysis.py|----> [trade\_alpaca\_grok4.log]

| | (P/L, win rate) | (read trades)

| +-----------------+

| |

| +-----------------+

| | data\_utils.py |----> [Alpaca API or Mock]

| | (fetch prices, | (price data)

| | preprocess) |

| +-----------------+

| |

| +-----------------+

| | prediction.py |----> [historical\_analysis\_APPL.json]

| | (Grok-3 API) | (historical data)

| +-----------------+

| |

| +-----------------+

| | order\_execution.py|----> [Alpaca API or Mock]

| | (place orders, | (positions, orders)

| | update SL/TP) |

| +-----------------+

|

v

[Loop every ~5s for 390 min]

**Module List**

1. config.py
   1. **Function**: Configuration and global state management
   2. **Description**: Defines constants (e.g., SYMBOLS = ["AAPL"], SEQUENCE\_LENGTH, RISK\_PER\_TRADE), API credentials (loaded from environment variables for security), and global state dictionaries (open\_positions, sl\_prices, tp\_prices) for tracking AAPL positions, stop-loss, and take-profit prices. Provides the foundation for system parameters and secure credential handling.
2. logging\_utils.py
   1. **Function**: Logging setup
   2. **Description**: Configures a logger with both file (trade\_alpaca\_grok4.log) and console handlers to output real-time logs to Bash and persist them to a file. The setup logging function ensures consistent logging across all modules, with INFO level messages for events like trade placements and data fetches.
   3. **Details**: When running python main.py in /home/gmafanasiev/ai\_trading\_bot/, trading status updates (e.g., fetches, predictions, orders) appear in real-time in the Bash console and are simultaneously written to ./trade\_alpaca\_grok4.log. The file updates automatically—no refresh needed. Use console for live monitoring and the log for persistent records. This is handled by the logging\_utils.py module via its StreamHandler for console and FileHandler for the log file.
3. clients.py
   1. **Function**: Alpaca API client initialization
   2. **Description**: Initializes Alpaca trading and data clients using credentials from config.py. The init\_alpaca\_clients function creates clients for trading (order execution) and data retrieval (historical and real-time bars), handling connection errors with logging.
4. data\_utils.py
   1. **Function**: Data fetching and preprocessing
   2. **Description**: Provides fetch\_bar\_data to asynchronously retrieve AAPL historical bar data (close prices) from Alpaca and prepare\_grok4\_input to compute scaled statistical inputs (mean, std, min, max, last 5 prices) for predictions. Ensures robust data handling with error logging.
5. prediction.py
   1. **Function**: Prediction generation with historical data integration
   2. **Description**: Uses load\_historical\_analysis to read and summarize AAPL historical data from /home/gmafanasiev/historical\_analysis\_APPL.json (e.g., average RSI, ROC, support/resistance levels). The get\_grok4\_prediction\_and\_adjustments function combines these with real-time stats to call the xAI API, returning a price direction prediction (0.0–1.0), threshold, and risk adjustments.
6. trade\_analysis.py
   1. **Function**: Trade performance analysis
   2. **Description**: The analyze\_trades function parses the log file to calculate daily P/L, trade count, and win rate for AAPL, including unrealized P/L for open positions. It provides a summary used for interim and final daily reports, logged via trading\_loop.py.
7. order\_execution.py
   1. **Function**: Order placement and position management
   2. **Description**: Manages AAPL positions with get\_position (queries current position) and place order (executes buy/sell orders via Alpaca, updates stop-loss/take-profit). Tracks position state in config.py globals and logs trade outcomes.
8. trading\_loop.py
   1. **Function**: Core trading logic
   2. **Description**: Implements the async trading logic function, which runs the trading loop. It fetches account equity, closes positions at simulation end (currently 8 minutes), updates trailing stop-loss/take-profit, retrieves real-time data, gets predictions, and places orders for AAPL based on thresholds. Logs interim and final reports.
9. main.py
   1. **Function**: Program entry point
   2. **Description**: Orchestrates the system by initializing logging (logging\_utils.py), Alpaca clients (clients.py), and starting the trading loop (trading\_loop.py). Uses asyncio.run to execute the async main function, ensuring all components work together for trading.

**Notes**

* **Execution Flow**: Start with main.py, which sets up logging and clients, then runs trading\_loop.py. The loop fetches data (data\_utils.py), gets predictions (prediction.py), analyzes trades (trade\_analysis.py), and executes orders (order\_execution.py), all configured for AAPL via config.py.
* **Historical Data**: prediction.py integrates AAPL’s historical analysis, enhancing predictions with metrics like RSI and support/resistance levels.
* **Logging**: Real-time logs (Bash) and file logs are managed by logging\_utils.py, with daily reports generated by trade\_analysis.py and logged via trading\_loop.py.