**Python Project Verification Steps**

**Steps to Test the Basic Structure**

1. **Verify Directory Structure and File Presence**
   1. **Action**: Check that all eight .py files are correctly placed in /home/gmafanasiev/ai\_trading\_bot/ and that /home/gmafanasiev/historical\_analysis\_APPL.json exists.
   2. **Command in PythonAnywhere Bash**:
   3. ls -l /home/gmafanasiev/ai\_trading\_bot/
   4. ls -l /home/gmafanasiev/historical\_analysis\_APPL.json
   5. **Expected Output**:
   6. /home/gmafanasiev/ai\_trading\_bot/:
   7. config.py logging\_utils.py clients.py data\_utils.py prediction.py trade\_analysis.py order\_execution.py trading\_loop.py main.py
   8. /home/gmafanasiev/historical\_analysis\_APPL.json
   9. **Purpose**: Confirms all files are in place and accessible.
2. **Check Syntax and Import Integrity**
   1. **Action**: Use Python’s -m py\_compile to verify syntax and ensure modules can be imported without errors. This checks for syntax issues and missing dependencies.
   2. **Command** (run from /home/gmafanasiev/ai\_trading\_bot/):
   3. python -m py\_compile \*.py
   4. **Expected Outcome**: No output if all modules are syntactically correct. If errors occur (e.g., invalid syntax or missing imports), you’ll see error messages like SyntaxError: ....
   5. **Additional Check**: Run a Python script to test imports:
   6. # test\_imports.py (create temporarily in /home/gmafanasiev/ai\_trading\_bot/)
   7. import config
   8. import logging\_utils
   9. import clients
   10. import data\_utils
   11. import prediction
   12. import trade\_analysis
   13. import order\_execution
   14. import trading\_loop
   15. print("All modules imported successfully")

Run with:

python /home/gmafanasiev/ai\_trading\_bot/test\_imports.py

**Expected Output**: All modules imported successfully. If imports fail (e.g., due to missing dependencies like alpaca or pandas), you’ll see an ImportError.

1. **Validate Environment Variables**

A screenshot of a computer

AI-generated content may be incorrect.

Done – all good!

A screen shot of a computer

AI-generated content may be incorrect.

1. **Test Logging Setup**
   1. **Action**: Verify logging\_utils.py can initialize logging and write to both console and file.
   2. **Test Script**:
   3. # test\_logging.py
   4. from logging\_utils import setup\_logging
   5. logger, file\_handler = setup\_logging('/home/gmafanasiev/ai\_trading\_bot/test\_log.log')
   6. logger.info("Test log message")
   7. file\_handler.flush()
   8. print("Logging setup completed")

Run with:

python /home/gmafanasiev/ai\_trading\_bot/test\_logging.py

* 1. **Expected Outcome**:
     1. Console output: 2025-07-17 12:02:00,123 - Test log message and Logging setup completed.
     2. A file /home/gmafanasiev/ai\_trading\_bot/test\_log.log is created with the same message.
  2. **Purpose**: Confirms logging\_utils.py works and can be used by other modules.

1. **Test JSON File Access**
   1. **Action**: Ensure prediction.py can read /home/gmafanasiev/historical\_analysis\_APPL.json.
   2. **Test Script**:
   3. # test\_json\_access.py
   4. from prediction import load\_historical\_analysis
   5. import logging
   6. logging.basicConfig(level=logging.INFO)
   7. result = load\_historical\_analysis('/home/gmafanasiev/historical\_analysis\_APPL.json', 'AAPL')
   8. print(f"JSON loaded: {result is not None}")

Run with:

python /home/gmafanasiev/ai\_trading\_bot/test\_json\_access.py

* 1. **Expected Outcome**: Logs summarizing historical data (e.g., avg RSI=65.23) and console output: JSON loaded: True. If the file is missing or malformed, you’ll see a logged error and JSON loaded: False.

1. **Mock Module Interactions**
   1. **Action**: Test key functions with mock inputs to verify interoperability without real API calls.
   2. **Test Script**:
   3. # test\_modules.py
   4. import logging
   5. from logging\_utils import setup\_logging
   6. from data\_utils import prepare\_grok4\_input
   7. import pandas as pd
   8. import numpy as np
   9. # Setup logging
   10. logger, file\_handler = setup\_logging('/home/gmafanasiev/ai\_trading\_bot/test\_log.log')
   11. # Mock data for prepare\_grok4\_input
   12. mock\_data = pd.DataFrame({'close': np.random.rand(60) \* 100})
   13. stats = prepare\_grok4\_input(mock\_data, 60)
   14. logger.info(f"Mock stats: {stats}")
   15. file\_handler.flush()
   16. print("Module interaction test passed")

Run with:

python /home/gmafanasiev/ai\_trading\_bot/test\_modules.py

* 1. **Expected Outcome**: Logs the mock stats (e.g., Mock stats: {'mean': 0.5, ...}) and prints Module interaction test passed. This verifies data\_utils.py and logging\_utils.py work together.
  2. **Note**: Testing clients.py, prediction.py (API call), trade\_analysis.py (log parsing), and order\_execution.py (Alpaca orders) fully requires real API access or mocking libraries (e.g., unittest.mock). For a basic structure check, you can skip these or mock them if needed.

1. **Check Module Dependencies**
   1. **Action**: Ensure required libraries are installed in PythonAnywhere.
   2. **Command**:

pip show pandas numpy requests scikit-learn alpaca-py

* 1. **Expected Outcome**: Confirms libraries are installed. If missing, install them:

pip install --user pandas numpy requests scikit-learn alpaca-py

* 1. **Purpose**: Prevents runtime errors due to missing dependencies.

1. **Static Code Analysis (Optional)**
   1. **Action**: Use pylint or flake8 to check for PEP8 compliance and potential issues.
   2. **Command**:

pip install --user pylint

pylint /home/gmafanasiev/ai\_trading\_bot/\*.py

* 1. **Expected Outcome**: Reports style issues or potential errors (e.g., unused imports). The provided modules are PEP8-compliant, so warnings should be minimal.

**Notes**

* **Scope**: These tests validate syntax, imports, logging, JSON access, and basic function execution without running the full trading loop or making API calls, avoiding real trades or data fetches.
* **Limitations**: Full functionality (e.g., Alpaca/xAI API calls, async trading loop) requires real execution. Mocking APIs would need additional setup (e.g., unittest.mock for requests.post or alpaca-py).
* **PythonAnywhere**: Run these tests in a Bash console. For scheduled tasks, redirect outputs to a file (e.g., python test\_imports.py > test\_output.txt).
* **Cleanup**: Remove temporary test scripts (test\_imports.py, test\_config.py, etc.) and test logs after validation.

If you want a specific test script combining all checks or assistance setting up mocks for API-dependent modules, let me know!