

Attempt 1



In Progress

NEXT UP: Submit Assignment

Add Comment

Unlimited Attempts Allowed

▼ Details

Class Project Phase 1: Create your own trading system

VERY IMPORTANT:

DO NOT ADD MONEY TO YOUR ALPACA ACCOUNT
DO NOT SHARE YOUR API KEYS WITH ANYONE
KEEP IT SIMPLE IF YOU DO NOT HAVE TIME



Step 1: Create an account on Alpaca

1.1 Visit Alpaca's Website

- Open your web browser and go to [Alpaca's official website \(https://alpaca.markets/\)](https://alpaca.markets/).

 image.png

1.2 Sign Up for an Account

- Look for a "Sign Up" or "Get Started" button on the homepage.
- Click on the button to begin the account creation process.

1.3 Fill in the Necessary Information

- You will be prompted to provide essential information to set up your account. This typically includes:
 - **Full Name:** Enter your full legal name.
 - **Email Address:** Provide a valid email address that you have access to.
 - **Password:** Create a secure password for your account.

1.4 Agree to Terms and Conditions

- Read through Alpaca's terms and conditions or user agreement.
- Tick the box or take the necessary action to signify your agreement.

1.5 Verification

- Complete any additional verification steps required to confirm your identity. This may involve email verification or other security measures.

1.6 Account Confirmation

- After completing the signup process, you may receive a confirmation email. Follow the instructions in the email to verify your account.

1.7 Log In

- Return to the Alpaca website and log in to your newly created account using the credentials you provided during the signup process.

1.8 Explore Account Dashboard

- Once logged in, familiarize yourself with the account dashboard. This is where you'll find key information about your account, such as balances and settings.

Alert: You do not need to add funds. You do not need to add money to the account.

1.9 Set Up Two-Factor Authentication (Optional)

- For added security, consider enabling two-factor authentication if Alpaca offers this feature.



Step 2: Paper trading configuration

Alert: Do not trade your money! Do not add funds!!!

In this step, you'll be setting up and configuring your paper trading account on Alpaca. Paper trading allows you to simulate trading in a risk-free environment using virtual money. It's an essential step before deploying your algorithm with real funds.

1. Log in to your Alpaca Account:

- Visit the Alpaca website and log in to your account using the credentials you created during the signup process.

2. Navigate to "Paper Overview":

- Once logged in, find and click on the "Paper Overview" section. This is where you'll configure and manage your paper trading account.



1. Understand Paper Trading Account Settings:

- Take a close look at the settings of your paper trading account. You'll typically see details such as starting equity and buying power.
- Starting equity is the initial amount of virtual money allocated to your paper trading account.
- Buying power represents the total amount of virtual money you can use for trading.

2. Resetting Paper Trading Account:

- Note that you can reset your paper trading account at any time by clicking the "Reset" button. This action will return your starting equity and buying power to their original values.

3. Familiarize Yourself with the Interface:

- Explore the paper trading interface to understand how to place orders, view your portfolio, and track performance.
- Get comfortable with the tools and features available in the paper trading environment.

Step 3: Obtain API Keys

API keys are essential for connecting your trading algorithm to the Alpaca trading account. They act as a secure way to authenticate and authorize your algorithm to interact with the Alpaca API.

1. Sign in to Your Alpaca Account:

- Go to the Alpaca website and sign in to your account using your credentials.

2. Navigate to Your Trading Account:

- Once logged in, navigate to your trading account. This is where you'll find the information and settings related to your Alpaca API keys.

3. Locate API Keys:

- In the overview or account settings section, look for the area where API keys are displayed. Alpaca usually provides separate keys for the paper trading account and the live trading account.



4. View and Copy API Keys:

- Once you've located the API keys, you should be able to view and copy them. These keys typically include:
 - API Key ID: A unique identifier for your API key.
 - Secret Key: A secret passphrase that, along with the API Key ID, is used for authentication.

5. Keep API Keys Secure:

- Treat your API keys like sensitive information. Keep them secure and private to prevent unauthorized access to your Alpaca account.
- Avoid sharing your API keys in public forums or repositories.

6. Understand Key Permissions:

- API keys come with certain permissions that define what actions your algorithm can perform. Ensure that the permissions associated with your keys align with the requirements of your trading strategy.

7. Test API Keys:

- Before integrating the keys into your algorithm, consider testing them with simple API requests to ensure they are valid and working as expected.

Step 4: Getting Market Data from Alpaca

In this step, you'll choose the asset you want to trade (crypto or stock) and explore how to retrieve market data from Alpaca using the alpaca-py library. Feel free to choose any other library or ways to connect to Alpaca.

1. Choose the Asset to Trade:

- Decide whether you want to trade cryptocurrencies or stocks. Note that if you choose crypto, markets are available 24/7, while stock markets operate during specific hours.

2. Explore Alpaca GitHub Repository:

- Visit Alpaca's GitHub repository to find examples and documentation on how to interact with the Alpaca API using the alpaca-py library.
- The repository often contains code samples and tutorials that can guide you through the process of retrieving market data.

3. Install alpaca-py Library:

- Using your preferred package manager (e.g., pip), install the alpaca-py library. This library provides a Python interface for interacting with the Alpaca API.

```
pip install alpaca-trade-api
```

4. Review Alpaca API Documentation:

- Explore Alpaca's official documentation for market data to understand the available endpoints, parameters, and data formats.
- The documentation is often well-detailed and provides examples of how to make requests for market data.

5. Understand Asset Symbols and Market Hours:

- For stocks, be aware of the symbols associated with the assets you're interested in. Understand the market hours for stocks as Alpaca provides data only during these hours.
- For cryptocurrencies, you can access data 24/7.

6. Sample Code for Getting Market Data:

- Write sample code using alpaca-py to retrieve market data. This might involve specifying the asset symbol, time interval, and other relevant parameters.

```
import alpaca_trade_api as tradeapi

# Set your API key and secret
api_key = 'your_api_key'
api_secret = 'your_api_secret'
base_url = 'https://paper-api.alpaca.markets' # Use paper trading base URL for testing

# Initialize Alpaca API
api = tradeapi.REST(api_key, api_secret, base_url, api_version='v2')

# Get historical market data
symbol = 'AAPL' # Replace with your desired stock symbol
timeframe = '1D' # Daily timeframe
historical_data = api.get_barset(symbol, timeframe).df
```



Advice: If you want to trade real time, you can also use the FIX api.

7. Handle Authentication and Rate Limits:

- Ensure that your code handles authentication properly using the API key and secret.
- Be mindful of rate limits imposed by Alpaca. Consider implementing rate-limiting mechanisms in your code to avoid exceeding these limits.

8. Explore Additional Data Endpoints:

- Beyond historical data, explore other endpoints provided by Alpaca for real-time data, account information, and more. This exploration will help you gather the necessary data for making informed trading decisions.

By following these steps, you'll be able to connect to Alpaca, retrieve historical market data, and lay the foundation for building your trading algorithm. The alpaca-py library simplifies the process of interacting with the Alpaca API, making it easier to integrate market data into your trading system.

Step 5: Saving Market Data

Advice: Do not overcomplicate this part if you do not have time. You can just record the data on the disk (using the pickle library like we did it in class) and use the data.

In this step, you'll focus on devising a strategy for efficiently saving the market data you retrieve from Alpaca. Properly storing this data is crucial for backtesting, analysis, and making informed trading decisions.

1. Choose Data Storage Method:

- Decide on the method of data storage. Common options include using databases, such as SQLite, MySQL, or PostgreSQL, or saving data to flat files like CSV (Comma-Separated Values) or JSON.

2. Database Storage:

- If opting for a database, create a database schema that accommodates the type of data you'll be collecting (e.g., OHLCV - Open, High, Low, Close, Volume data).
- Establish tables to organize data efficiently, considering factors like asset symbols, timeframes, and timestamps.

3. File Storage:

- If choosing file storage, determine a structured format for your files. For example, you might save each asset's data in separate CSV files with columns representing relevant data points.
- Consider creating a directory structure to organize files by asset and timeframe.

4. Timestamps and Timezones:

- Ensure that your storage method accounts for timestamps and timezones accurately. Timestamps are crucial for aligning data with specific points in time, and timezones ensure consistency across different data sources.

5. Regular Updates:

- Plan for regular updates to your stored data. Depending on your trading frequency, you may need to update your data at specific intervals to ensure that your algorithm has access to the latest information.

**6. Automate Data Retrieval and Storage:**

- Consider automating the process of data retrieval and storage. Schedule tasks or scripts to run at specific intervals to fetch fresh data from Alpaca and update your storage accordingly.

7. Backtesting Integration:

- Design your storage system with backtesting in mind. Ensure that it's easy to retrieve historical data from your chosen storage method for use in backtesting your trading algorithm.

8. Data Cleaning and Validation:

- Implement data cleaning and validation processes to ensure the integrity of the stored data. This includes handling missing or erroneous data points and maintaining data quality.

By carefully planning and implementing a strategy for saving market data, you create a robust foundation for your algorithmic trading system. This step ensures that your trading algorithm has access to accurate and up-to-date historical data, enabling effective backtesting and analysis.

Step 7: Develop a Trading Strategy

In this step, you'll define a clear and effective trading strategy that your algorithm will follow. A trading strategy outlines the conditions under which your algorithm will make buy or sell decisions. Here are the key aspects to consider:

1. Define Your Trading Goals:

- Clearly articulate your financial goals and risk tolerance. Are you looking for short-term gains, long-term growth, or a balance of both? Understanding your goals will guide the development of your strategy.

2. Select Trading Instruments:

- Determine the financial instruments you want your algorithm to trade. This could include stocks, cryptocurrencies, or other assets. Consider factors like liquidity, volatility, and market hours.

3. Technical Indicators and Signals:

- Choose the technical indicators and signals that your algorithm will use to make decisions. Common indicators include moving averages, relative strength index (RSI), and moving average convergence divergence (MACD).

Advice: You can use a cross moving average signal or a linear regression.

4. Backtesting:

- Backtest your trading strategy using historical market data. This involves running your algorithm on past data to simulate how it would have performed. Backtesting helps you identify strengths and weaknesses in your strategy.

5. Paper Trading:

- Before deploying your algorithm, test it in a simulated environment using paper trading. This allows you to validate its performance in real market conditions without risking actual money.

6. Real-time Monitoring:

- Once live, monitor your algorithm's performance in real-time. Keep an eye on key metrics, such as the success rate of trades, drawdowns, and overall portfolio performance. Be prepared to make adjustments based on market conditions.

Developing a robust and effective trading strategy is a critical step in the success of your algorithmic trading system. It requires a combination of technical analysis, risk management, and adaptability to navigate the dynamic nature of financial markets.

Deliverables



1. Python Code Uploaded to Canvas:

- Upload the complete Python code for your algorithmic trading system to the Canvas platform (class project phase 3). Ensure that the code is well-organized, commented, and includes necessary documentation within the code itself.

2. Document Explaining What Was Done:

- Create a comprehensive document that provides an overview of your algorithmic trading system. This document should include the following sections:
- **Introduction:**
 - Briefly introduce the purpose and goals of your trading system.
- **Market Data Retrieval:**
 - Explain how you retrieve market data from Alpaca using the alpaca-py library. Include code snippets and examples.
- **Data Storage Strategy:**
 - Describe your strategy for saving market data. Discuss the chosen storage method (database or file system), the structure of data storage, and considerations for timestamps and timezones.
- **Trading Strategy Development:**
 - Outline the steps taken to develop your trading strategy. Discuss the selection of trading instruments, technical indicators, risk management rules, and position sizing.
- **Code Explanation:**
 - Provide a detailed explanation of key sections of your Python code. Highlight important functions, variables, and decision-making processes.
- **Testing and Optimization:**
 - Discuss how you tested your trading strategy, including backtesting and optimization steps. Explain any adjustments made based on testing results.
- **Automation and Scheduling:**
 - Detail how you automated the data retrieval process and scheduled tasks to update market data. Include information on error handling, logging, and script version control.
- **Paper Trading and Monitoring:**

- Explain how you utilized Alpaca's paper trading feature to simulate live market conditions. Discuss how you monitored your algorithm's performance in a risk-free environment.
- **Results and Lessons Learned:**
 - Reflect on the results of your project. Discuss any challenges encountered, lessons learned, and potential improvements for future iterations.
- **Compliance and Legal Considerations:**
 - Briefly touch on any compliance or legal considerations related to algorithmic trading. Mention how your system aligns with relevant financial regulations.
- **Conclusion:**
 - Summarize your project, highlighting key achievements and outcomes.

By preparing these deliverables, you'll provide a comprehensive overview of your algorithmic trading system, making it easier for your students or peers to understand, evaluate, and potentially replicate or build upon your work.



Example

Below is a simple Python code example to get you started with a basic algorithmic trading project using Alpaca. This example focuses on fetching historical market data and placing alternating Buy and Sell orders every 5 seconds in a paper trading environment.

Please note that this is a basic illustration, and a complete trading system would require more sophisticated strategies, risk management, and error handling.

But if you are gettingn something similar, you will get full points. The more you enhance your system, the better it will be for your interviews.

```
import alpaca_trade_api as tradeapi
import time

# Set your Alpaca API key and secret
API_KEY = 'your_api_key'
API_SECRET = 'your_api_secret'
BASE_URL = 'https://paper-api.alpaca.markets' # Use paper trading base URL for testing

# Initialize Alpaca API
api = tradeapi.REST(API_KEY, API_SECRET, base_url=BASE_URL, api_version='v2')

# Specify the asset symbol and trading interval
symbol = 'AAPL' # Replace with your desired stock symbol
interval = 5 # Time interval in seconds

# Main trading loop
while True:
    try:
        # Get historical market data for the specified symbol and interval
        historical_data = api.get_barset(symbol, '1Min', limit=1).df[symbol]

        # Get the last close price
        last_close_price = historical_data['close'].iloc[-1]

        # Determine whether to place a Buy or Sell order based on alternating intervals
        if time.time() % (2 * interval) < interval:
            # Place Buy order
            api.submit_order(
                symbol=symbol,
                qty=1, # Adjust quantity as needed
                side='buy',
                type='limit',
                time_in_force='gtc',
                limit_price=last_close_price * 1.01, # Place a limit order slightly above the current price
            )
            print("Buy order placed at {}".format(last_close_price))
```

```

else:
    # Place Sell order
    api.submit_order(
        symbol=symbol,
        qty=1, # Adjust quantity as needed
        side='sell',
        type='limit',
        time_in_force='gtc',
        limit_price=last_close_price * 0.99, # Place a limit order slightly below the current price
    )
    print("Sell order placed at {}".format(last_close_price))

# Wait for the next interval
time.sleep(interval)

except Exception as e:
    print(f"An error occurred: {e}")
    # Add proper error handling based on your requirements

```

Please replace 'your_api_key' and 'your_api_secret' with your actual Alpaca API key and secret. This example uses a basic alternating Buy/Sell strategy, and you would need to adapt and expand upon this foundation to create a more sophisticated and effective trading algorithm.

Additionally, keep in mind that algorithmic trading involves real financial risks, and it's crucial to thoroughly test and understand strategy in a simulated environment before deploying it with real funds.



Sharing your work with recruiters (optional)

Protecting one's code and ensuring its security is crucial, especially when showcasing it on platforms like GitHub. Let's add a section on using GitHub for code visibility while maintaining security.

Deliverables

5. GitHub Repository:

- Create a private GitHub repository to host your algorithmic trading system code. Mark the repository as private to control access to the code.
- Provide access only to trusted individuals, such as recruiters, mentors, or collaborators.

6. Readme Documentation:

- Include a comprehensive [README.md \(http://readme.md/\)](http://readme.md/) file in your GitHub repository. This file should serve as a guide for anyone accessing the repository and should include:
 - **Introduction:**
 - Briefly introduce the project, its goals, and the technologies used.
 - **Repository Structure:**
 - Outline the structure of your repository, indicating the main Python script, any additional scripts, and supporting documents.
 - **Setup Instructions:**
 - Provide clear instructions on how to set up and run the code locally. Include any dependencies or configurations needed.
 - **Code Explanation:**
 - Offer a high-level explanation of your code's architecture, key functions, and main logic.
 - **Usage Examples:**
 - If applicable, provide examples of how to use or interact with your algorithm. This can include sample inputs or scenarios.

- **Results and Screenshots:**

- Include any relevant results, performance metrics, or screenshots demonstrating the output of your algorithm.

- **Contributing Guidelines:**

- If you're open to contributions, include guidelines for potential contributors. This might cover areas such as submitting issues, proposing enhancements, or making pull requests.

7. Security Measures:

- Emphasize the importance of security when sharing code on GitHub:

- **Private Repository:**

- Reiterate that the repository is set to private to restrict access.

- **Access Control:**

- Only provide repository access to trusted individuals. Use GitHub's collaboration features to manage access levels carefully.

- **API Key Security:**

- If your code involves API keys, provide guidance on how to handle them securely. Encourage users to use environment variables or other secure methods to protect sensitive information.

- **Avoid Hardcoding Secrets:**

- Highlight the importance of avoiding the hardcoding of sensitive information like API keys directly in the code.



VERY IMPORTANT: DO NOT SHARE YOUR API KEYS

8. Canvas Submission Update:

- Update the Canvas submission instructions to include a link to the private GitHub repository. Clearly communicate that this repository is for viewing purposes only and should not be shared publicly.

By leveraging GitHub, you provide a platform for recruiters to easily access and review your code while maintaining control over who has access to it. Emphasizing security measures ensures responsible code sharing and protects sensitive information.

You will upload your code and you write up using a zip/tar file OR you can upload the file and write the write up in canvas.

Keep in mind, this submission will count for everyone in your HW Part2 group.

Choose a submission type


Text


Web URL


Upload


More



(<https://canvas.uchicago.edu/courses/64841/modules/items/2795789>)



Assignment (<https://canvas.uchicago.edu/courses/64841/modules/items/27>)