**Options Chain and Margin Calculation**

**Project Overview**

This project interacts with the Upstox API to retrieve options chain and margin data for various financial instruments. The program then processes this data to calculate margin requirements and display key insights for options trading. This document provides an overview of each function and the logic used in the code**.**

**1. API Authentication and Setup**

import urllib.parse

import requests

apiKey = 'your\_api\_key'

secretKey = 'your\_secret\_key'

rurl = urllib.parse.quote('https://127.0.0.1:5000/', safe="")

**Explanation**

* **Purpose**: Sets up the authentication and request parameters for the Upstox API.
* **API Key and Secret**: apiKey and secretKey are used to authorize requests.
* **Redirect URL**: rurl encodes a URL that will be used for redirecting upon successful authentication.

2. **Token Generation**

url = 'https://api.upstox.com/v2/login/authorization/token'

headers = {

'accept': 'application/json',

'Api-version': '2.0',

'Content-Type': 'application/x-www-form-urlencoded'

}

data = {

'code': 'auth\_code',

'client\_id': apiKey,

'client\_secret': secretKey,

'redirect\_uri': 'https://127.0.0.1:5000/',

'grant\_type': 'authorization\_code'

}

response = requests.post(url, headers=headers, data=data)

json\_response = response.json()

access\_token = json\_response['access\_token']

**Explanation**

* **Token Request**: Uses authorization code flow to retrieve an access token.
* **Access Token**: Extracts access\_token from the response, which is required to authorize future API calls.

**3. Fetching Margin Data**

url = 'https://api.upstox.com/v2/user/get-funds-and-margin'

headers = {

'accept': 'application/json',

'Api-Version': '2.0',

'Authorization': f'Bearer {access\_token}'

}

params = {'segment': 'COM'}

response = requests.get(url, headers=headers, params=params)

margin\_data = response.json()

**Explanation**

* **Purpose**: Requests the margin data for a specified segment (in this case, commodities).
* **Parameters**: The Authorization header uses the access\_token for authentication.

4. **Data Retrieval and Preprocessing**

fileUrl = 'https://assets.upstox.com/market-quote/instruments/exchange/complete.csv.gz'

symboldf = pd.read\_csv(fileUrl)

symboldf['expiry'] = pd.to\_datetime(symboldf['expiry']).apply(lambda x: x.date())

**Explanation**

* **Purpose**: Downloads a CSV file containing all trading instruments.
* **Expiry Date Formatting**: Converts expiry columns to a date format to ensure consistency in further filtering operations.

5. **Data Cleaning**

df\_cleaned = symboldf.dropna(subset=['name', 'expiry', 'option\_type'])

**Explanation**

* **Purpose**: Cleans the data by removing rows where essential columns (name, expiry, option\_type) have missing values.
* **Result**: df\_cleaned now holds a structured dataset ready for further processing.

6. **Retrieve Options Chain Data**

def get\_option\_chain\_data(instrument\_name: str, expiry\_date: str, side: str) -> pd.DataFrame:

expiry\_date = pd.to\_datetime(expiry\_date, format='%d-%m-%Y').date()

options = df\_cleaned[

(df\_cleaned['name'] == instrument\_name) &

(pd.to\_datetime(df\_cleaned['expiry']).dt.date == expiry\_date) &

(df\_cleaned['option\_type'] == side)

]

output\_data = []

if side == 'PE':

highest\_bid = options['last\_price'].max()

for strike in options['strike'].unique():

output\_data.append({

'instrument\_name': instrument\_name,

'strike\_price': strike,

'side': side,

'bid/ask': highest\_bid

})

elif side == 'CE':

highest\_ask = options['last\_price'].max()

for strike in options['strike'].unique():

output\_data.append({

'instrument\_name': instrument\_name,

'strike\_price': strike,

'side': side,

'bid/ask': highest\_ask

})

return pd.DataFrame(output\_data)

**Explanation**

* **Purpose**: Retrieves option chain data based on the instrument name, expiry date, and option type (e.g., "CE" for Call, "PE" for Put).
* **Filter Logic**:
  + Filters df\_cleaned by instrument\_name, expiry\_date, and side.
  + The code then selects the highest bid/ask prices and organizes the data by strike for each option\_type.
* **Returns**: A DataFrame with option details including the instrument\_name, strike\_price, side, and bid/ask.

7. **Margin and Premium Calculation**

def get\_margin\_requirement(instrument\_name: str, strike\_price: float, side: str) -> float:

return strike\_price \* 0.10

def calculate\_margin\_and\_premium(data: pd.DataFrame, lot\_size: int) -> pd.DataFrame:

if data.empty:

print("No data available to calculate margin and premium.")

return data

data['margin\_required'] = data.apply(

lambda row: get\_margin\_requirement(row['instrument\_name'], row['strike\_price'], row['side']), axis=1

)

data['premium\_earned'] = data['bid/ask'] \* lot\_size

return data

**Explanation**

* **Purpose**: Calculates the required margin and premium earned for options.
* **Margin Calculation**:
  + get\_margin\_requirement is a placeholder function that calculates margin as 10% of the strike\_price.
  + This could be expanded based on more specific margin rules.
* **Premium Calculation**:
  + Computes the premium based on bid/ask price and lot\_size

8. **Sample Usage**

# Example to fetch options data and calculate margin and premium

instrument\_name = "NATURALGAS"

expiry\_date = "23-12-2024"

side = "PE"

lot\_size = 10

option\_chain\_df = get\_option\_chain\_data(instrument\_name, expiry\_date, side)

option\_chain\_with\_margin = calculate\_margin\_and\_premium(option\_chain\_df, lot\_size)

print("Option Chain Data with Margin and Premium:")

print(option\_chain\_with\_margin)

**Explanation**

* **Purpose**: Demonstrates how to fetch option chain data for "NATURALGAS" expiring on "23-12-2024" and calculate margins.
* **Result**: Outputs a DataFrame with option chain details, including margin and premium information.

**Notes on AI Assistance**

Throughout this project, ChatGPT was used to support various aspects of development, including function generation, debugging, and concept clarification:

1. **Function Generation**:
   * ChatGPT was used to draft initial function structures for tasks like data retrieval from the Upstox API, filtering the options chain data, and calculating margin requirements.
   * Example: Generated the initial structure of get\_option\_chain\_data and calculate\_margin\_and\_premium functions, which were later customized to fit specific project needs.
2. **Debugging Support**:
   * ChatGPT assisted with debugging syntax errors and troubleshooting issues in data processing functions.
   * Example: In debugging get\_option\_chain\_data, ChatGPT suggested adjustments in the filtering conditions and DataFrame handling, which helped resolve issues with data extraction and formatting.
3. **Concept Clarification**:
   * ChatGPT provided explanations on key financial terms and concepts relevant to the project, such as "options margin requirements," "bid/ask pricing," and "lot size."
   * Example: Clarified the calculation basis for margin requirements and helped determine appropriate values for placeholder calculations