Abhi Lad - Comity Coding Challenge

Assumptions considered:

- Currently, bids are filtered on absolute value, i.e. bids >=0.1 and bids<=-0.1 are kept. This can be changed to keep only positive bids >=0.1
- Spread = DA_price – RT_price
- PnL per node and hour = Spread * bid_mw
- Bid_type = DEC(buy) if bid_mw >=0 else INC(sell)
- Total PnL = Sum(all PnL per node and hour for all days)

Note: Using default number of iterations in the minimize routine will cause premature termination for 100MW and 500 Risk Tolerance. The generated bids will neither be optimal nor satisfy the constraint.

I recommend setting maxiter to 1000. This will make all the bids satisfy the constraint sum|w| <= V and atleast half of them will reach optimal solution. Increasing maxiter to a higher number will guarantee optimal solution for all days but will increase processing time.

The number of iterations will depend on the max daily volume limit and the risk tolerance value. But in general, setting the maxiter to higher value will be more appropriate.

The implementations of functions are based on the above considerations and literature found in public domain. Multiple implementations have been provided on all assumptions where my understanding of the challenge document was murky. Instructions on how to toggle these assumptions are provided in README file.

Answers to the challenge questions

1. The basic working code was ready in 1.5 hours with the optimization functions and data manipulation functions. I spent another 5-6 hours on additional functionalities.

The main problem I faced was with some vagueness in the challenge document, regarding point-in-time estimates and output formatting. But more time was spent on coming up with metrics and charts, since we are not dealing with traditional returns.

I have implemented the following additional functionalities:

- Option to choose only bids with value >=0.1 MW (only positive bids) or bids with absolute value >= 0.1 MW
- Risk Tolerance tuning by maximizing PnL, by specifying range of risk tolerance, fraction of data to run tuning on
- Specify daily volume limit in the terminal
- Specify maximum number of iterations allowed
- Option to print if optimization fails to converge
- Parallelizing the optimization routine to speed up backtest
- Metrics:
    - Win/Loss rate
    - Profit factor
    - Sharpe ratio (crude implementation)
- Charts:
    - DART spreads and Expected DART spreads
    - Drawdown (crude implementation)
    - Cumulative returns

2. In the optimization routine, the loop which performs the optimization on each day is the bottleneck. It is obvious that the calculations inside the loop are independent of previous loop iterations, so it can be parallelized. I have used joblib Parallel module to schedule jobs in parallel and improve performance and speed up the backtest.

Original implementation: 5 minutes

Parallelized implementation: ~40 seconds

Performance speed up: 6.5x

3. There are multiple factors to consider when setting the risk tolerance value. Factors like investment goals, either short term profit -> high risk tolerance, long term growth -> lower risk tolerance. Market conditions and underlying events like natural disasters, geopolitical events etc. which can change energy consumption pattern, thus affecting risk tolerance.

But considering the information that we have at hand, we can plot and observe the expected DART spread to check for volatility. If there is high volatility, we set risk tolerance factor to a lower value. And if there is low volatility, we set risk tolerance factor to a higher value. This strategy to set risk tolerance is conservative in nature for safer investment.

We can iterate over multiple risk tolerances to check which works best in our market.

In practice, people tend to take higher risk tolerance on highly volatile market to earn higher profits but at the same time there is possibility of higher losses as well.

4.

Risk Tolerance	Max Volume	PnL	Win/Loss Ratio
500	100	28299.53	4.37
	200	62485.64	2.57
	400	63362.62	2.51
	800	63360.05	2.51

Max Volume	Risk Tolerance	PnL	Win/Loss Ratio
100	200	26339.91	3.45
	500	28299.53	4.37
	700	21315.90	5.55
800	200	26338.14	3.45
	500	63360.05	2.51
	700	89140.66	2.27

As can be seen above, as the maximum daily volume limit is increased keeping the risk tolerance constant, the PnL also increases and reaches a saturation point. Even if max volume limit is increased, the solution never outputs bids that utilize the max available volume. Instead, number of losing bids increase.

Now, in the table 2, we keep max volume same and change the risk tolerance. We find that for lower trading volume, increasing risk increases chances of wins but at the same time, the gross loss incurred is also increased. Thus, we risk higher gross loss for high chances of profit.

And for higher trading volume, increasing risk allows the bids to reach the max daily volume limit, thereby maximizing the profits. Even though the number of winning bids decrease, we see increase in gross profits. Thus, we risk higher chances of loss for higher gross profits.

5.

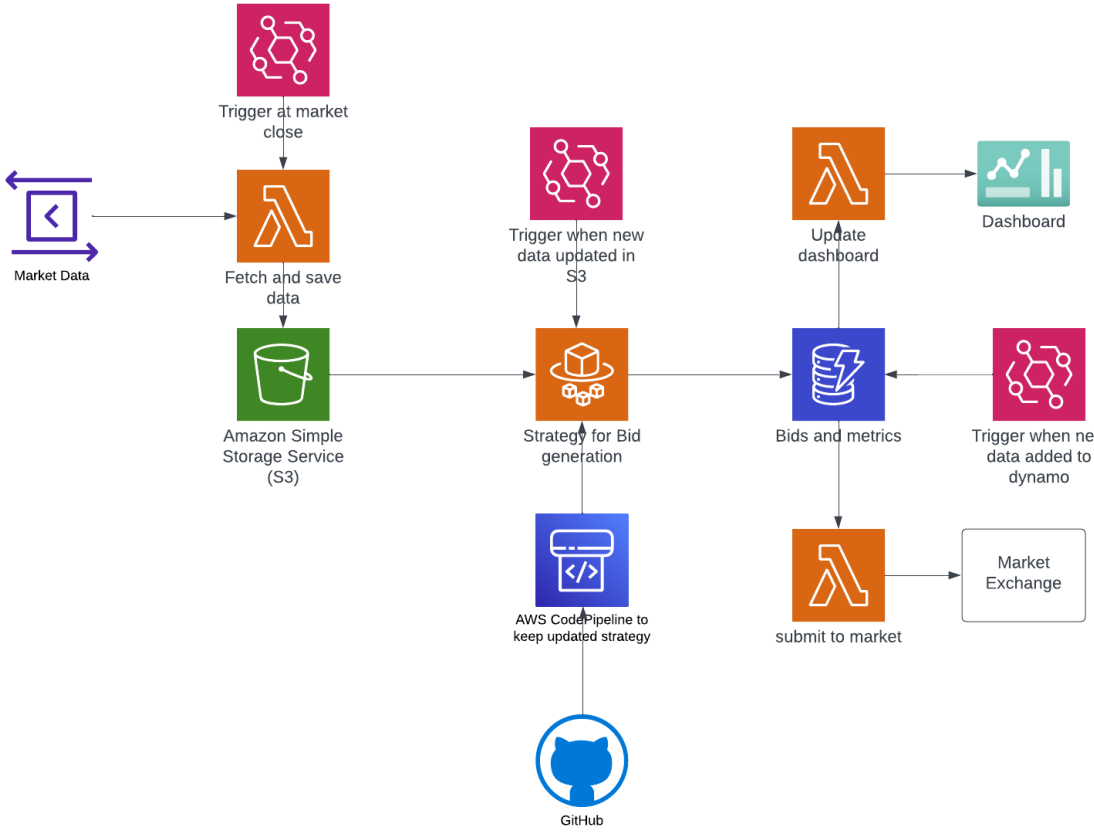
This is a simplified way to productionize the code. There might be some components missing, but the overall idea to productionize the code is captured.

Steps:

- Modularize the code
    - Strategy module which contains generate_daily_bids() and get_bids_from_daily_portfolios() functions. It simply runs the strategy and generates the daily bids
    - Data processing module which generates point-in-time estimates and other data transformations
    - Data visualization module which contains metrics and plotting functions
    - Try to remove circular dependencies while separating these 3 modules
- Convert data processing module to lambda function which fetches data from market and processes it to format required in strategy module and saves in S3 bucket. The eventbridge triggers it once everyday after the last operating hour.
- AWS Fargate is triggered which fetches the updated data from S3 and runs the bid generation strategy, and saves the bids to DynamoDB.
- Since data processing and visualization doesn't change, only the strategy changes, we set CI/CD for strategy module on Fargate to fetch from updated github repo.
- Update on bids DynamoDB triggers another event, which updates the dashboard and submits the portfolio to market.
- Additionally, remove unnecessary print statements and unnecessary saves in data processing and visualization modules.

Note: Fargate can be replaced with Lambda or EC2 depending on computational resource requirements

Since the strategy module is separate, we can modify it to improve the results during backtesting on local environment. When we push the changes to github, CI/CD ensures that the latest strategy is fetched and updated on AWS Fargate for production and is the same as we used for backtesting.



Bonus features implemented:

- More metrics quantifying the strategy backtest performance
- Charts of the strategy backtest performance
- Functionality to tune the risk tolerance factor
- Computational performance improvements/speedups for the backtest