COM5 Commodities Capstone: Report

Introduction

This report details our algorithmic trading system designed to identify and capitalize on profitable opportunities in crude oil, heating oil, and gasoline markets. Our approach focuses on four interconnected models: fundamental news analysis, transportation arbitrage, refinery production (crack spread), and spot-futures arbitrage. Each component is implemented in Python with connectivity to the trading API, allowing for automated detection and execution of trading strategies.

Throughout the code excerpts, code and comments have been removed for brevity. The full python model can be referred to for specifics.

System Architecture

Our trading system is structured with a central orchestration module (main.py).

```
"""
main.py
Some Code/Comments removed for visibility.
"""
def main():
    while True:
        refining.refining_model(session)
        fundamental.fundamental_model(session)
        transportation.transportation_model(session)
        storage.storage_model(session)
```

This architecture allows each model to run independently while sharing the same session and market data. Below is a typical terminal output from our system execution:

```
Net position: -90.0
Tick: 290
refining_now: False
Expected profit from refining: $-111,000.00
Refining not profitable. Skipping.
Already processed this news. Skipping.
Transporting from AK to CS is not profitable. Expected profit: 400.0
Transporting from CS to NYC is not profitable. Expected profit:
-800.0
[Tick 290] CL-2F storage spread: -0.01
```

1. Fundamental News Model

Implementation

We scan every tick for new news items. If a headline includes both “ACTUAL” and “FORECAST,” and hasn’t already been traded on “(news_id != last_traded_news_id),” we extract the actual and forecast numbers, compute the difference in millions of barrels, and scale the surprise linearly using a sensitivity constant. Based on price path analysis across several COM5 runs, we calibrated that each 1 million barrel surprise correlates with a roughly $0.10 crude oil price move.

This function analyzes the difference between actual and forecasted inventory changes. Showing only function definition and docstring for brevity:

```
def fundamental_EIA_report(session):
    """
    Parse the latest EIA report headline and return the difference between actual and
forecast.

    Args:
        session (requests.Session): Authenticated session.

    Returns:
        float: Difference between actual draw/build and forecasted draw/build.
 """
```

Trading Logic

When the news indicates a significant inventory surprise (greater than 2 million barrels), our model executes a directional trade. Here is some trading logic from our EIA_trade function, the function responsible for actually executing trades from EIA reports. The code focuses on being able to make the maximum trade it can without breaking position limits (up to 90). It then saves the trade specifications so that the position management code can close it after 20 ticks (seconds).

```
# Check current position
    net = helper.get_net_position(session)

    TRADE_QUANTITY = 90
    if expected_price_move > 0.2:
        # Bullish → Buy
        max_allowed = (POSITION_LIMIT - net)
        quantity = min(TRADE_QUANTITY, max_allowed)
        quantity = quantity - quantity % 3

        helper.place_order(session,'CL-2F', quantity/3, 'BUY', 'MARKET')
        helper.place_order(session,'CL-2F', quantity/3, 'BUY', 'MARKET')
        helper.place_order(session,'CL-2F', quantity/3, 'BUY', 'MARKET')
        print(f"Placed BUY order for {quantity} contracts of CL-2F.")

        eia_active_trade = {
            "entry_tick": helper.get_tick(session),
            "side": 'BUY',
            "quantity": quantity
        }
```

Position Management

Positions are automatically closed after a fixed holding period of 20 ticks. Whenever the fundamental model is run, the model checks if existing EIA trades have been made and are still open. If they are, and the time has come to close them, the closing orders are made and trade history is cleared.

```
# === Handle closing EIA trade ===
    if eia_active_trade["entry_tick"] is not None:
        if current_tick - eia_active_trade["entry_tick"] >= HOLD_TICKS:
            print("Closing EIA trade after 20 ticks.")
            side = "BUY" if eia_active_trade["side"] == "SELL" else "SELL"
            quantity = eia_active_trade["quantity"]

            helper.place_order(session,'CL-2F', quantity/3, action=side, order_type='MARKET')
            helper.place_order(session,'CL-2F', quantity/3, action=side, order_type='MARKET')
            helper.place_order(session,'CL-2F', quantity/3, action=side, order_type='MARKET')
            print(f"Closed EIA position: {side} {quantity} contracts of CL-2F.")

            eia_active_trade = {
                "entry_tick": None,
                "side": None,
                "quantity": 0
            }
```

Extra

The fundamental model also parses pipeline price updates - this is crucial for the proper functioning of the transportation model, as the transportation model is usually only profitable after changes in pricing. This is done in the pipeline_news function.

```
    news_items = helper.get_latest_news(session)

    if news_items is None or len(news_items) == 0:
        print("No news available.")
        return

    latest_news = news_items[0]
    if latest_news.get('ticker') == "AK-CS-PIPE":
        print("Pipeline news detected.")
        headline = latest_news.get('headline', '').upper()
        match = re.search(r"\$?([\d,]+)", headline)
        number_str = match.group(1)
        number = int(number_str.replace(',', ''))
        print(f"AK-CS-PIPE lease price updated to {number}")
        globals.AK_CS_PIPE = number
```

We also quantify the impact of various news and store them in the csv file. This is done in the other_news.py file and can be run by itself, instead of from the main.py file.

2. Transportation Arbitrage Model

Opportunity Detection

When regional price differences exceed transport costs, arbitrage becomes possible. In COM5, crude trades in Alaska, Cushing, and New York. Each location can deviate due to local demand/supply, but pipelines exist to move product. If oil in AK is cheaper than in CS by more than the pipeline fee, we can buy in AK, sell in CS, lease the pipeline, and lock in profit. The same logic applies to the CS → NYC route.

Our transportation arbitrage model identifies price discrepancies between crude oil at different locations (Alaska, Cushing, and New York):

```
MIN_PROFIT_THRESHOLD = 3500 # Minimum profit threshold for transportation
TRADE_QUANTITY = 100 # Quantity of crude oil to transport
def should_transport_AK_CS(session):
    # Get price data for CL-AK and CL
    cl_AK_bid, cl_AK_ask = helper.ticker_bid_ask(session, 'CL-AK')
    cl_bid, cl_ask = helper.ticker_bid_ask(session, 'CL')
    net = helper.get_net_position(session)
    Expected_profit = (10000*cl_bid - 10000*cl_AK_bid - globals.AK_CS_PIPE)

    if Expected_profit > MIN_PROFIT_THRESHOLD:
        print("Transporting from AK to CS is profitable. Expected profit: ", Expected_profit)
        return True

    else:
        print("Transporting from AK to CS is not profitable. Expected profit: ",
Expected_profit)
        return False
```

Execution Strategy

When a profitable opportunity is identified, our system executes a fully hedged arbitrage trade:

```
quantity = TRADE_QUANTITY

        helper.lease_storage(session, 'AK-STORAGE')

        for i in range(0, int(quantity/10)):
            helper.place_order(session, 'CL-AK', 10, 'BUY', 'MARKET')
            print(f"Bought spot position: BUY 10 CL-AK")
            helper.place_order(session, 'CL-2F', 10, 'SELL', 'MARKET')
            print(f"Shorted futures position: SELL 10 CL-2F")
            helper.lease_use_transport(session, 'AK-CS-PIPE', 'CL-AK', 10)
```

Position Unwinding

The system automatically unwinds transportation positions after the pipeline transport completes. The system to wait for transportation to finish had to be specifically calibrated, so that we can save money by leasing right before arrival, but without risking distressed prices. In the end, we measured time directly and started executing this code after 26 seconds had passed from the start of the transportation code.

```
if transportation_trade_info_AK["active"]:
        if time.time() - transportation_trade_info_AK["current_time"] >= 26:
            print("Closing transportation hedge and spot position after 30 ticks.")
            print(f"{30 - (time.time() - transportation_trade_info_AK['current_time'])}
seconds left in transportation")
            if time.time() - transportation_trade_info_AK["current_time"] >= 29:
                print("Too late to buy storage. Incurred Distressed Prices")
            else:
                for i in range(10):
                    helper.lease_storage(session, 'CL-STORAGE')

            while not helper.get_position_ticker(session, 'CL') == 100:
                time.sleep(0.2)
                print("Waiting for position to be 100 CL")

            if net > 70:
                for i in range(1, 5):
                    helper.place_order(session, 'CL', 25, 'SELL', 'MARKET')
                    print(f"Closed Spot ({i}) position: SELL 25 CL-2F")
                    helper.place_order(session, 'CL-2F', 25, 'BUY', 'MARKET')
                    print(f"Closed Futures ({i}) position: BUY 25 CL-2F")
```

3. Refinery Production (Crack Spread) Model

Based on the terminal output, our refinery model evaluates the profitability of refining crude oil into heating oil and gasoline:

```
refining_now: False
Expected profit from refining: $-111,000.00
Refining not profitable. Skipping.
```

The model calculates the crack spread by comparing the cost of crude oil plus refining costs against the potential value of refined products:

```python
def should_refine(ho_price, rb_price, cl_price)
    REFINING_COST = 300_000
    MIN_PROFIT_THRESHOLD = 40_000
    CONTRACTS = 30
    BARRELS_PER_CONTRACT = 1000
    BBL_TO_GALLONS = 42_000

    total_revenue = (10 * ho_price * BBL_TO_GALLONS) + (20 * rb_price * BBL_TO_GALLONS)
    total_cost = (CONTRACTS * cl_price * BARRELS_PER_CONTRACT) + REFINING_COST
    total_profit = total_revenue - total_cost

    return total_profit >= MIN_PROFIT_THRESHOLD, total_profit
```

When profitable, the system would execute the refining operation by:

1. Leasing storage for crude oil
2. Purchasing crude oil
3. Buying imperfect hedge (through futures)
4. Leasing and using the refinery
5. Selling the resulting products (after the 45 seconds has passed)

Full function that we call in main.py:

```python
def refining_model(session):
    refining_now, lease_id, lease_end_tick = helper.get_refinery_lease_info(session)
    print(f"refining_now: {refining_now}")
    if refining_now:
        print("Refinery is being used. Continuing to next model.")
        time.sleep(1)
    else:
        positions = helper.get_positions(session)
        if positions['HO'] > 0 and positions['RB'] > 0 and positions['CL-2F'] < 0:
            print("Closing Refinery positions.")
            helper.place_order(session, 'HO', positions['HO'], 'SELL', 'MARKET')
            helper.place_order(session, 'RB', positions['RB'], 'SELL', 'MARKET')
            helper.place_order(session, 'CL-2F', 30, 'BUY', 'MARKET')
            print("Refinery is available. Attempting to refine.")
        try_refining(session)
```

4. Storage Arbitrage Model

Our storage model identifies mispricings between spot and futures markets, accounting for the cost of carry and time decay. It does this by calculating the spread between spot and futures prices and comparing it to the expected difference based on storage costs. This is the only model that is not automated, but if we choose to, we can manually trade it by using the spread info that our storage model gives us ~ every second in the terminal.

Our model handles both futures contracts dynamically based on the current round:

```
def storage_model(session):
    # Call for both futures
    tick = helper.get_tick(session)
    # In round 1, both futures exist
    if round_num == 1:
        CL_future_arb(session, "CL-1F", tick)
        CL_future_arb(session, "CL-2F", tick)
    # In round 2, CL-1F has expired
    elif round_num == 2:
        CL_future_arb(session, "CL-2F", tick)
```

Trading Strategy:

When the futures price is significantly higher than expected (positive spread above threshold), we implement a cash-and-carry arbitrage by:

1. Buying spot crude oil (CL)

2. Simultaneously selling futures contracts (CL-1F or CL-2F)

3. Leasing storage facilities to hold the crude

4. Delivering against the futures contract at expiration

Our model accounts for time decay in expected storage costs, reflecting the diminishing carrying cost as we approach futures expiration. This dynamic adjustment ensures we capture only genuine arbitrage opportunities rather than expected term structure differences.

Helper Functions and API Interaction

Our trading system relies heavily on a custom helper module that handles API interactions. This was by far the biggest file, and ended up being over 400 lines of code (albeit with docstrings). However, the help it gave through modularity and abstraction was immense, making the code better readable throughout. Every function that interacted with the api was defined in the helper.py file.

We also have a config file, in which you can define port and API key. It gives the base url for api interaction for the rest of the files.

Conclusion

Our trading system provides a robust and integrated approach to capturing profit opportunities across the crude oil complex. By combining fundamental news analysis, transportation arbitrage, refinery crack spread, and spot-futures arbitrage, we can effectively identify and execute on the most profitable opportunities at any given time.

Furthermore, we have fully automated three of the four models, allowing us to reliably and consistently gain profits with low risk due to human error. Though our hedging is imperfect due to basis risk, we still see consistent gains. The storage model, which is not automated, is still tradeable manually and still assisted by the storage model in python, which gives us the spread for spot-futures arbitrage.

I have abided by the Babson Code of Ethics in this work and pledge to be better than that which would compromise my integrity.