Feature Engineering

Feature v1

1. Initial filtering
    a. Epoch Day
    b. Product RIC
    c. Timestamp Range
2. Grouping by counterparty and trade side
    a. Groups trades by:
        i. **Product RIC**: The financial instrument being traded.
        ii. **Counterparty**: The entity executing the trades (lastExecutedCounterparty).
        iii. **Trade Side**: Numeric side of the trade (1 = BUY, 2 = SELL).
3. **Features Extracted:**
    a. **Trade Count**: Number of trades executed by the counterparty.
    b. **Total Quantity**: Sum of quantities traded by the counterparty.
    c. **Bot Size**: Size of the market-making bot involved in the trades.
    d. **Sold Size**: Total size sold by the bot.
    e. **Executed Quantity**: Total quantity executed in the trades.
    f. **First and Last Timestamps**: The start and end times of trades executed by the counterparty.
4. **Counterparty Type Mapping**
    a. Adds a **counterparty type** field based on a mapping expression (counterparty_mapping_expr).
        i. Example types: Institutional, Retail, Latency Arbitrageur.
5. **Final Grouping with Trade Side Handling**
    a. Groups data at the **product RIC** and **counterparty type** level, aggregating metrics separately for **BUY** and **SELL** trades.
6. **Features Extracted:**
    a. **Total Trades**: Total number of trades.
    b. **Buy Trades**: Number of BUY trades.
    c. **Sell Trades**: Number of SELL trades.
    d. **Total Volume**: Total traded volume.
    e. **Buy Volume**: Volume of BUY trades.
    f. **Sell Volume**: Volume of SELL trades.
    g. **Bot Size, Sold Size, and Executed Quantity**: Aggregated metrics for bot-related sizes and executed quantities. (productRic level)
    h. **Time Range**: Tracks the first and last timestamps for trades in the group.
    i. **Counterparties**: A list of counterparties with:
        i. **Name**: Counterparty identifier.
        ii. **Trade Count**: Number of trades.
        iii. **Total Quantity**: Volume of trades.
        iv. **Side**: BUY, SELL, or UNKNOWN (converted from numeric side).

Feature Engineering

7. Things to improve
    a. sometimes they split the orders (orderId) then sum
    b. I want to track all the trades when it happened and visualise it (time range) by productRic level and counterpartyType level
    c. Overlay the signals on top of underlying price / product price
    d. Use $facet to run multiple sub pipelines in parallel for unique counterpartyTypes
        i. The number of types are based on counterparty_mapping 'type' column unique values

Feature v2

1. Improvement 1
    a. To run multiple sub-pipelines in parallel for each unique counterpartyType, we can use the $facet stage in MongoDB.
    b. The $facet stage allows you to define multiple independent pipelines that can execute concurrently within a single query.
2. Improvement 2
    a. The Problem
        i. In your current pipeline, **each sub-pipeline in $facet** is filtering data again for a specific counterpartyType using a $match. This means that MongoDB is repeatedly processing the same dataset for each sub-pipeline, which is unnecessary and inefficient.
        ii. Imagine you have a list of trades, and you want to organize them by counterpartyType (e.g., "Retail", "Institutional").
        iii. Instead of sorting the trades all at once, you're currently asking MongoDB to filter the entire list separately for each type (e.g., "Show me Retail trades", then "Show me Institutional trades", and so on).
        iv. This repetition wastes time and computational resources because MongoDB is repeatedly checking the same dataset for each counterpartyType.
    b. The Optimisation

Instead of filtering the dataset repeatedly, group all trades by counterpartyType first in a single step.

        i. Once the data is grouped, each counterpartyType already has its data organized. Now, you can pass this grouped data into sub-pipelines for further processing.
    c. With Optimization:
        i. First, group the trades by counterpartyType:
            1. Retail: [Trade 1, Trade 3, Trade 4]
            2. Institutional: [Trade 2, Trade 5]
        ii. Now, each sub-pipeline only works with its own group:
            1. The "Retail" sub-pipeline processes just the 3 Retail trades.
            2. The "Institutional" sub-pipeline processes just the 2 Institutional trades.

Feature Engineering

Feature v3

1. Improvements
    a. Added
        i. BuyTradeCount
        ii. SellTradeCount
        iii. CounterpartyBotSize
        iv. CounterpartySoldSize
        v. CounterpartyExecutedQuantity
    b. Preserved
        i. TradeCount
        ii. BotSize
        iii. SoldSize
        iv. ExecutedQuantity
        v. Timestamps

Feature v4

1. Improvements
    a. Uses pre-calculated buyTradeCount and sellTradeCount from stage 2
    b. More efficient than recalculating with $cond expressions
    c. Uses the new counterpartyBotSize and counterpartySoldSize fields
2. Example output fragment
    a. {

```
"_id": {
  "productRic": "12345.HK",
  "counterpartyType": "latencyarb"
},
"totalTrades": 500,
"buyTrades": 275,
"sellTrades": 225,
"counterparties": [
  {
    "name": "CP1",
    "tradeCount": 120,
    "buyTradeCount": 80,
    "sellTradeCount": 40,
    "totalQuantity": 30000,
    "botSize": 18000,
    "soldSize": 12000,
    "side": "PREDOMINANTLY_BUY"
  }
```

```
    ]
}
```

Feature v5

1. Problem
    a. For participationByTrades and participationByVolume, denominator should be the sum of all trades across ALL counterparty types
    b. Requires cross-pipeline aggregation
2. Options
    a. Option 1
        i. Pre-calculate totals
3. Improvements
    a. Correct Participation Calculations:
        i. Now uses true global totals as denominators
        ii. Percentages will accurately reflect market share
    b. Performance Considerations:
        i. The pre-calculation approach (Option 1) is more efficient
        ii. Only requires one additional grouping stage
        iii. Maintains all data in a single pipeline
    c. Data Accuracy:
        i. Ensures all participation percentages sum to 100% across categories
        ii. No double-counting or incorrect ratios
4. Example output fragment
    a.

```
{
  "latencyarb": [
    {
      "productRic": ".ESZ3",
      "counterpartyType": "latencyarb",
      "metrics": {
        "participation": {
          "byTrades": 0.4528,  // 45.28% of ALL trades
          "byVolume": 0.3875   // 38.75% of ALL volume
        }
      }
    }
  ],
  "volarb": [
    {
      "productRic": ".ESZ3",
      "counterpartyType": "volarb",
      "metrics": {
        "participation": {
```

Feature Engineering

```
            "byTrades": 0.3215,  // 32.15% of ALL trades
            "byVolume": 0.4012   // 40.12% of ALL volume
          }
        }
      }
    ]
  }
```

Feature v6

1. Improvements
    a. New Hierarchy Structure
    b. Top-level activity and participationByCounterpartyType categories
    c. Nested trades and volume under activity
2. Old
    a. Participation, imbalance, frequency, volume, trades
3. New
    a. Activity
        i. Trades
            1. Counts
            2. Trade imbalance
            3. Frequency (per minute)
        ii. Volume
            1. Amounts
            2. Volume imbalance
            3. Volume per trade
    b. ParticipationByCounterpartyType
        i. ByTrades
        ii. ByVolume

Feature v7

1. Basic structure of results
    a. The results is a list of dictionaries
        i. results = [
            {
              "latencyarb": [...], # Array of documents for latency arbitrage trades
              "volarb": [...],   # Array of documents for vol arbitrage trades
              "retail": [...]   # Array of documents for retail trades
            }
          ]
    b. **Single-element list**: The $facet stage always returns **one dictionary** inside a list, where each key is a counterparty type.

c. **Per-counterparty arrays**: Each key's value is an array of processed documents for that counterparty type.
d. **Contents of each array**: Contains the output from your final projection stage for that counterparty type.

2. Detailed document format
   a. {


```
"productRic": "68783.HK",
"counterpartyType": "latencyarb",
"metrics": {
  "activity": {
    "trades": {
      "total": 1428,
      "buy": 832,
      "sell": 596,
      "imbalance": 0.1653,
      "frequencyPerMinute": 18.72
    },
    "volume": {
      "total": 2856000,
      "buy": 1686400,
      "sell": 1169600,
      "imbalance": 0.1811,
      "perTrade": 2000
    }
  },
  "participationByCounterpartyType": {
    "byTrades": 0.4271,
    "byVolume": 0.4019
  }
},
"counterparties": [
  {
    "name": "CP1",
    "tradeCount": 420,
    "buyTradeCount": 280,
    "sellTradeCount": 140,
    "totalQuantity": 840000,
    "side": "PREDOMINANTLY_BUY"
  }
]
}
```

Feature Engineering

3. Why this format?
    a. **$facet behavior**: Creates parallel sub-pipelines for each counterparty type.
    b. **Your pipeline design**: The final projection stage shapes each document to include:
        i. Product identification (productRic)
        ii. Counterparty classification (counterpartyType)
        iii. Calculated metrics (trade/volume stats)
        iv. Detailed counterparty breakdowns
4. Accessing the data
    a.
```
# Get all latency arbitrage trades
latency_trades = results[0]["latencyarb"]

# Get first retail trade document
retail_data = results[0]["retail"][0]

# Print participation rate for vol arb
print(results[0]["volarb"][0]["metrics"]["participationByCounterpartyType"]["byTrades"])
```

Feature v8

1. Process the faceted results into a cleaner format
    a. Initialise an empty dictionary
    b. Loops through each counterparty type
    c. Checks two conditions before adding data to the output
    d. Populates the output dictionary
        i. Keys: counterpartyTypes
        ii. Values: data array
2. Why This Structure?
    a. The $facet stage in MongoDB returns results in this format:
```
[{
  "latencyarb": [...data...],
  "volarb": [...data...],
  # (only includes types that matched data)
}]
```

    b. This code transforms it into a more usable:
```
{
  "latencyarb": [...data...],
  "volarb": [...data...]
}
```

Feature v9

1. Improvements
    a. Handle all cases (single, multiple, or all products) in a single aggregation pipeline
    b. ProductRic followed by counterpartyType
2. Example output

Feature Engineering

a.


```
{
  "68783.HK": {
    "latencyarb": {
      "metrics": {
        "activity": {
          "trades": {
            "total": 10,
            "buy": 6,
            "sell": 4,
            "imbalance": 0.2,
            "frequency": 12.34
          },
          "volume": {
            "total": 1000,
            "buy": 600,
            "sell": 400,
            "imbalance": 0.2,
            "perTrade": 100
          }
        },
        "participation": {
          "byTrades": 0.5,
          "byVolume": 0.5
        }
      }
    }
  }
}
```