IEOR4725 Final Project

Akiva Bachrach (ab5231)
Chun Yat Yeung (cy2623)
Jingyi Bai (jb4601)
Kirill Krupenin (kk3518)
Neeraj Sudhakar (nvs2121)

December, 2022

Abstract

We develop trading strategies on liquid binary prediction markets using order flow datasets from the Good Judgement Project (GJP).

Contents

1 GJP Markets

Wisdom of crowds aggregates the predictions of a diverse group of individuals to minimize the idiosyncratic noises. This project is inspired from the wisdom of crowds principle, by applying an aggregation method – **contribution weighted model** – to the order flows of individual "wise" traders in order to construct a belief distribution of binary events, which we trade upon as deviation is observed.

We study the Good Judgement Project (GJP), created to harness the wisdom of crowds to forecast world events by identifying individuals less influenced by cognitive biases, whose predictions are aggregated to make the most informed decisions. The datasets are composed of prediction market data tracking participants' submit orders and trades on contracts regarding prediction events. The contracts involve binary/ternary questions whose outcomes are realized over time. The contract price converges to 100 if the event takes place and 0 otherwise.

The seminal work "Identifying Expertise to Extract the Wisdom of Crowds" by Budescu and Chen (2015) motivates this project. In the paper, the authors propose a measure of contribution to gauge the performance of judges relative to the group according to a quadratic scoring function, and positive contributors are weighted to aggregate forecasts [1]. Here, instead of a scoring function, contribution of an individual trader is calculated based on the historical profit and loss (P&L) when trading in the training datasets. Traders with negative P&L are excluded, thus not contributing, while positive P&Ls are appropriately normalized to yield the contribution weights of contributing traders.

1.1 Datasets

Two datasets are used, namely

- `ifps.csv`: market meta-data e.g. ids, questions, market start/end dates, options; each line describes a unique prediction market identified by an id;
- `pm_transactions.lum1.yr2.csv`: order flow e.g. timestamps, market ids, trader ids, operation types, trade directions and quantities; each line corresponds to an operation/execution sent by a market participant, and we are mainly concerned with submit orders and trades. Note that the trade direction is identified by the "buy" and "long" columns, where True/True and False/False correspond to a buy action and otherwise a sell action, effectively a negated XOR. From here, one may create a signed trade quantity.

The datasets are previewed as follows.

`ifps.csv`

	ifp_id	q_type	q_text	q_desc	q_status	date_start	date_suspend	date_to_close	date_closed	outcome	short_title	days_open	n_opts	options
0	1001	0	Will the Six-Party talks (among the US, North ...	'In' refers to any time during the remainder o...	closed	9/1/11	12/30/11 0:00	12/31/11	1/2/12	b	Six-party talks resume	123.0	2	(a) Yes, (b) No
1	1002	0	Who will be inaugurated as President of Russia...	'In' refers to any time during the 2012 calend...	closed	9/1/11	5/14/12 0:00	5/15/12	5/6/12	b	president of Russia	248.0	3	(a) Medvedev, (b) Putin, (c) Neither
2	1003	0	Will Serbia be officially granted EU candidacy...	A 'yes' answer to this question requires not o...	closed	9/1/11	12/30/11 0:00	12/31/11	1/3/12	b	Serbia EU candidacy	124.0	2	(a) Yes, (b) No
3	1004	0	Will the United Nations General Assembly recog...	'By' means at or prior to the end of the day o...	closed	9/1/11	9/29/11 0:00	9/30/11	9/30/11	b	UN-GA recognize Palestine	29.0	2	(a) Yes, (b) No
4	1005	0	Will Daniel Ortega win another term as Preside...	If the Nicaraguan elections do not occur in la...	closed	9/1/11	11/4/11 0:00	11/5/11	11/5/11	a	Ortega win in Nicaragua	65.0	2	(a) Yes, (b) No

`pm_transactions.lum1.yr2.csv`

	timestamp	IFPID	outcome	user.ID	Op.Type	order.ID	buy	long	with.MM	matching.order.ID	price	qty	experience	by.agent
0	2012-06-16 08:24:55	1040	a	6203	SubmitOrder	1.0	True	False	NaN	nan	40	2	1.0	NaN
1	2012-06-16 08:24:55	1040	a	6203	Trade	1.0	True	False	True	nan	45	1	1.0	NaN
2	2012-06-16 08:24:55	1040	a	6203	Trade	1.0	True	False	True	nan	40	1	1.0	NaN
3	2012-06-16 08:27:04	1089	a	6203	SubmitOrder	2.0	True	False	NaN	nan	40	2	1.0	NaN
4	2012-06-16 08:27:04	1089	a	6203	Trade	2.0	True	False	True	nan	45	1	1.0	NaN

The datasets are connected by IFP ids, so that each market, carrying a unique IFP id, maps to an order flow book. We organize the datasets into the the following hashmap data structure, which is used throughout the project. At the surface level, we have the collection of ids, then for each id, we have the market meta-data summarized, with the order flow book contained inside the "market" key.

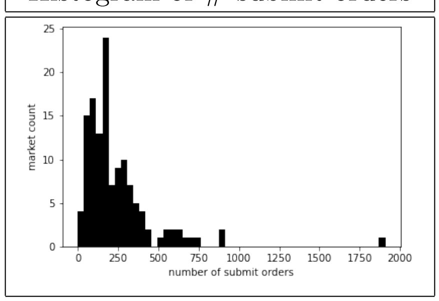
```
ifp = { # dict for all markets
    id: { # dict for market 'id'
        'short_title':  # short title (str)
        'q_text':       # event traded (str)
        'date_start':   # market start date (datetime)
        'date_suspend': # market suspend date (datetime)
        'options':      # options available (str)
        'n_opts':       # number of options (int)
        'outcome':      # option realized (str)
        'market': {
            'log':          # order flow log (dataframe)
            'order_log':    # submit orders log (dataframe)
            'trade_log':    # trades log (dataframe)
            'trade_agg':    # trades aggregated by timestamps (dataframe)
            'trade_cnt':    # trades count aggregated by trader ids (dataframe)
            'n_trades':     # total number of trades (int)
            'n_orders':     # total number of orders (int)
            'usr_ids':      # trader ids (list)
            'usr_log':      # order flow log of each trader (dict)
        }
    },
    ...
}
```

1.2 Liquid Binary Markets

ifps.csv describes a total of 617 prediction markets while pm_transactions.lum1.yr2.csv logs the transactions of 131 markets. To narrow down the scope of the project, we concern ourselves with only binary markets that are "liquid", because (1) liquidity implies active trades from which more information may be backed out and fed into our models, and (2) with liquidity, our orders have higher chance of execution and less market impact, so our simulated trades are more reflective of their actual performances.

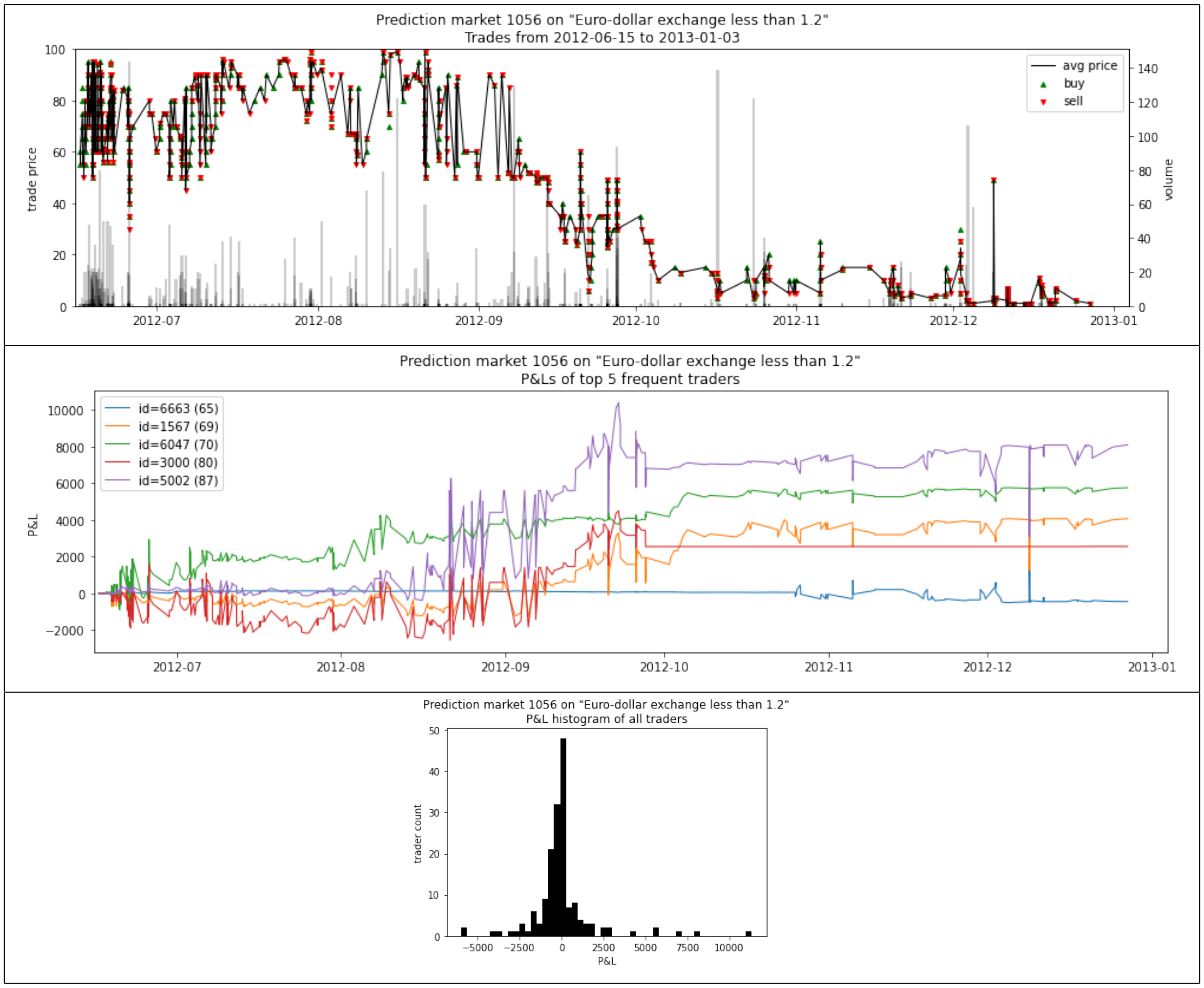
We define a liquid market as one with a number of submit orders above the 10th percentile, 69.9, and filtering, only 93 markets are binary and liquid. The distribution of the number of submit orders is as follows.

Histogram of # submit orders

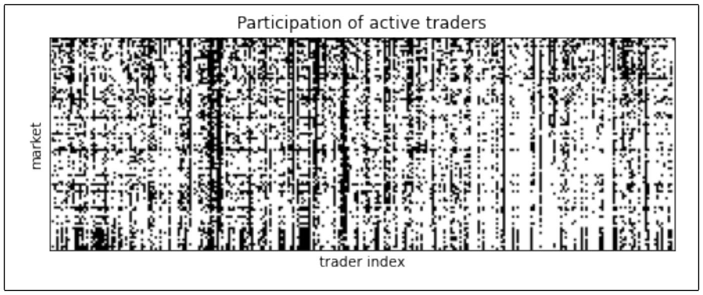


We illustrate a representative candidate of liquid binary prediction market, labeled 1056. Suspended on 12/30/12 with a false outcome, the market has 164 participating traders, 897 submit orders, 1152 trades (larger than submit orders due to order splitting), of which 573 are buys and 579 are sells.

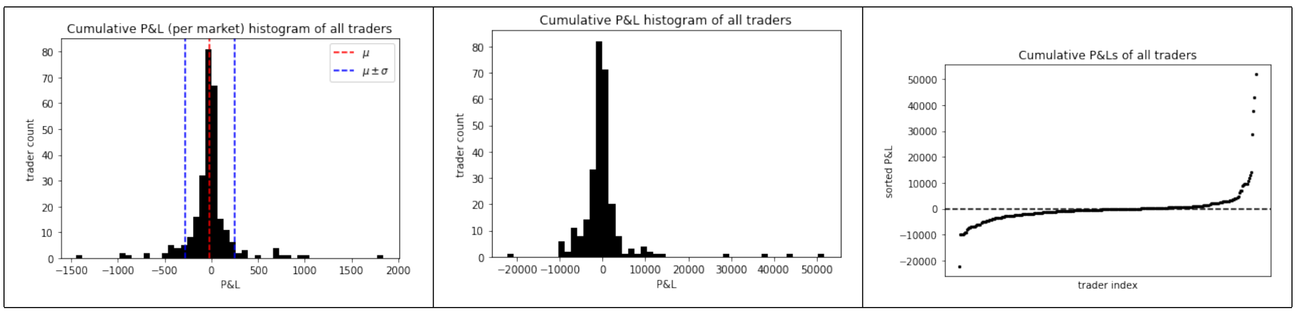
We present the price/volume series, the P&L series of the top 5 frequent traders and the P&L histogram of all traders. In the price series, a buy trade is indicated by an up green arrow while a sell trade is indicated by a down red arrow. At a fixed timestamp, multiple trades can be matched at different prices and volumes, so we also plot the volume-weighted trade price via the black line. The P&L of a trader is computed from the cumulative change in cash position throughout the market trading cycle and the final contract position held, marked to market at the final trade price. Typically, the final trade price either converges to 0 if outcome is false, as is the case here, and 100 if true, as time progresses and more information reveals. Overall, P&Ls average to 1.12 with standard deviation 1845.31, and the distribution is left-skewed with a heavy right-tail.



Now, we zoom out to inspect all 93 liquid binary markets. We represent the participation of each active trader in each market by a binary matrix, as visualized. The number of participating traders varies widely across markets, as seen from the horizontal stripes; the number of markets participated also varies widely across traders, as seen from the vertical stripes. Thus, the markets are quite inhomogeneous.



Repeating similar P&L calculations, we study the P&L distribution of all traders across the markets. On average, each trader loses 18.30 with standard deviation 265.22 per market participated; across all participated markets, where different traders participate in a different set of markets, each trader cumulatively gains 19.23 with standard deviation 6216.12. By sorting the P&Ls, we see that the distribution is heavy-tailed, but particularly prominent on the profit (right) side, spanning $[-20000, 50000]$.



2 Contribution Weighted Model

Here we present our **contribution weighted model** (`cwm`), by first sketching the ideas and then providing the algorithmic details.

Under `cwm`, positively contributing traders are identified based on their cumulative P&Ls (profits) in the train markets. These traders are deemed "better informed" as they perform (i.e. positive P&Ls) in the long run. Contribution weights w of traders correlate with their historical profits, with losing traders discarded, and w is normalized s.t. it sums to unity. We simulate trading in the test markets by keeping track of a time series of Beta-distributed belief $q_t \sim \text{Beta}(\alpha_t, \beta_t)$, updated according to buy/sell *submit* orders of contributing traders, whose beliefs are backed out of a *belief function* q(price, demand, wealth, gamma) under CRRA utility.

Now, for the trading strategy, if the mean of q_t (our belief) deviates sufficiently from (market) traded price p_t, we enter into a position, e.g. if $q_t \gg p_t$, buy certain amount of contracts according to a *demand function* n(price, belief, wealth, gamma). The level of deviation may be quantified by s.d. of q_t. As submit orders of contributing traders flow in, update α_t, β_t, and the mean and s.d. of the Beta-distributed belief q_t vary accordingly. How α_t, β_t get updated depends on specific strategy, e.g. contribution weighted model updates according to contribution weights w, volume weighted model updates according to "qty" of submit orders.

We lay out the trading algorithm:

1. Compute cumulative $P\&L_i$ for each active trader i in the train markets. Form the set of contributing traders $\{j : P\&L_j > 0\}$ who yield positive cumulative P&Ls.

2. Contribution weight w_j of contributing trader j is calculated by

$$w_j = \frac{P\&L_j}{\sum P\&L_j}, \tag{1}$$

strictly positive.

3. Set up contribution weighted model based on defined parameters in section 2.1: $\boldsymbol{W}, \boldsymbol{G}, \boldsymbol{C}, \boldsymbol{K}, \boldsymbol{T}, \boldsymbol{P}$, bold here for emphasis.

4. With wealth and risk-aversion parameters $\boldsymbol{W}, \boldsymbol{G}$, impute beliefs of contributing traders based on their submit order flow. Denote $q_{jt} = \text{belief}(p_t, n_t, \boldsymbol{W}, \boldsymbol{G})$ as the belief of trader j at time t, at which they submitted a buy/sell order.

5. Our belief of the binary event outcome is assumed to follow Beta-distribution $\text{Beta}(\alpha_t, \beta_t)$. As information (i.e. submit orders) flows in, α_t, β_t are updated accordingly. Specifically,

   - at inception, $\alpha_0 = \beta_0 = 0$;
   - as submit order with imputed belief q_{jt} flows in, update our belief distribution according to

   $$\begin{aligned} \alpha_t &= (1 - \boldsymbol{C} w_j)\alpha_{t-1} + \boldsymbol{C} w_j q_{jt} \\ \beta_t &= (1 - \boldsymbol{C} w_j)\beta_{t-1} + \boldsymbol{C} w_j (1 - q_{jt}) \end{aligned} \tag{2}$$

   where $\boldsymbol{C} < 1/\max w_j$ is our `cwm` weight parameter;
   - our mean belief is given by

   $$\hat{q}_t = \frac{\alpha_t}{\alpha_t + \beta_t} \tag{3}$$

   and s.d. is given by

   $$\hat{\sigma}_t = \frac{\sqrt{\alpha_t \beta_t / (\alpha_t + \beta_t + 1)}}{\alpha_t + \beta_t}; \tag{4}$$

   - check that upon first submit order from trader j, our mean belief is exactly q_{j1}.

6. With our time series of mean belief $\hat{q}_t$ and s.d. $\hat{\sigma}_t$, we execute (simulate) our trades following rules below:

   - we trade at time t only when some trades by other participants happen at time t;
   - suppose that at time t, trades occur at average price p_t with volume v_t, and p_t is the price we trade at should we submit an order;
   - compute a normalized trading signal

   $$s_t = \frac{p_t - \hat{q}_t}{\hat{\sigma}_t} \tag{5}$$

   and a temporary demand $n_t = \text{demand}(p_t, \hat{q}_t, \boldsymbol{W}, \boldsymbol{G})$;
   - we require that (1) n_t is rounded to an integer, (2) the absolute magnitude of n_t cannot exceed half of trading volume at time t, and (3) after trading n_t, our position size (total number of contracts held) does not fall outside the bound $[-\boldsymbol{P}, \boldsymbol{P}]$ – this prevents frequent trading as we stop over-sizing as the bound is reached;
   - with trigger level $\boldsymbol{K}$, if $s_t < -\boldsymbol{K}$ or $s_t > \boldsymbol{K}$, we execute a trade at price p_t with signed quantity n_t, appropriately updating the cash position, with transaction cost $\boldsymbol{T}|n_t p_t|$;

- the above procedure repeats, and cash, position etc. information is kept track of until the termination of market;
- at termination (the last trade), we obtain a cumulative P&L under strategy `cwm` for a specific market under parameters $\boldsymbol{W}, \boldsymbol{G}, \boldsymbol{C}, \boldsymbol{K}, \boldsymbol{T}, \boldsymbol{P}$.

Remark. *This is effectively a mean-reversion trading strategy, with the mean backed out of a Beta-distributed belief, expecting that traded price from noisy traders oscillates around the mean, which we take advantage of. The update rule is not exactly Bayesian but Bayesian-inspired as there is no obvious likelihood function with a Beta prior s.t. the Beta parameters update exponentially.*

2.1 Strategy Parameters

1. W: wealth of each trader e.g. 1000; this is just a parameter in belief/demand function, and will not constrain one's trading in any way;
2. G: risk aversion parameter, gamma, under CRRA utility e.g. 2; again, just a parameter in belief/demand function;
3. C: weight parameter of `cwm` e.g. 2; larger C means new order flow information is more emphasized;
4. K: trigger level of trading signal e.g. 0.5; smaller K means trading signal is more easily triggered;
5. T: transaction cost percentage e.g. 0.05;
6. P: position constraint s.t. $-P \leq \text{position} \leq P$ e.g. 500.

2.2 Inferring Belief

Assuming CRRA utility parametrized by risk-aversion parameter G, and that the investor computes a demand to maximize the expected utility transacting a contract, the demand functions (given belief q) and belief (given demand n) read

$$\begin{aligned} &\text{demand}(p, q, W, G) = W \cdot (a-1)/(1+p(a-1)), \qquad a = [q(1-p)/p(1-q)]^{1/G} \\ &\text{belief}(p, n, W, G) = p \cdot b/(1+p(b-1)), \qquad b = [1+n/(W-np)]^{G}, \end{aligned} \tag{6}$$

where n is the number of contracts demanded, $p \in [0,1]$ is the market price of contract, $q \in [0,1]$ is the investor's belief, W is the investor's wealth, and G is the risk-aversion parameter. The demand equation results from [2], and the belief equation is obtained from inverting the demand equation.

2.3 Assumptions

1. We know the *complete order flow* with trader information e.g. id – this is very unrealistic but that is what data provide to us. More realistically, we know only traded price and volume information. See benchmark strategies in section 2.5.
2. As we submit an order, it immediately gets executed at the average price at the next trading instance.
3. Price slippage is summarized by transaction cost parameter T.
4. We can take on short position on contracts with position size limit $[-P, P]$.

2.4 Training

We follow the routine below to imply the parameters from the train markets:

1. By backtesting (optimizing) a strategy in the train markets, we can imply parameters W, G, C, K. Our experience is that strategy P&L is most sensitive to C, K, so maybe we can first fix $W = 1000, G = 2$ and optimize C, K, then fine-tune W, G. Optimization over continuous space is impractical (too slow) and unnecessary (e.g. G is never precise) so we adopt grid search, over $C = 0.1, 0.2, ..., 0.5, 1, ..., 3$ and $K = 0, 0.1, ..., 1$.

2. Parameters T, P are set according to market natures/rules. For simplicity, we assume $T = 0.05, P = 500$. T is intentionally set large to avoid underestimation of price slippage, as prediction market is not liquid.

3. We do not train over all markets in `train_ids` as this is too time-consuming. Instead, we train over the top 5 liquid markets, top 10 liquid markets etc. until we observe convergence.

4. Mathematically, optimal parameters

$$C^*, K^* = \text{argmax}_{C,K} \frac{1}{N} \sum_{i=1}^{N} \text{strategyP\&L}_i(W = 1000, G = 2, C, K), \tag{7}$$

where N is the number of markets used in training. Then,

$$W^*, G^* = \text{argmax}_{W,G} \frac{1}{N} \sum_{i=1}^{N} \text{strategyP\&L}_i(W, G, C^*, K^*). \tag{8}$$

Optionally, we may optimize position limit P.

2.5 Benchmark Strategies

Here we do not update according to traders' cumulative P&Ls, which `cwm` adopts; instead, we consider the following strategies:

1. *Volume weighting of all submit order flows* (`vol`): choose belief update weight

$$w_t = \min(v_t / \max_{t' \le t} v_{t'}, w_{\max}), \tag{9}$$

say $w_{\max} = 0.5$ (called `wmax`), where v_t is traded volume at time t.

2. *Equal weighting of all submit order flows* (`eq`): choose belief update weight

$$w_t = w_0, \tag{10}$$

say $w_0 = 0.05$ (called `w0`).

3. *Herd trading of contributing traders* (`herd`): once we observe the submit order of trader j at time t, we trade with his belief q_{jt} at the next traded price at probability

$$p_j = \min(w_j / \max w_j, p_{\max}), \tag{11}$$

say $p_{\max} = 0.8$ (called `pmax`).

2.6 Functions

We outline some of the core functions used for trading simulation and backtest. For a complete list with the implementation details, see section 5.

1. `belief(price, demand, W, G)`: belief function depending on `price` and `demand`;
2. `demand(price, belief, W, G)`: demand function depending on `price` and `belief`;
3. `get_weight(ids)`: get weight for contribution weighted model from cumulative P&Ls of traders;
4. `trade_market_cwm(id, weight, W, G, ...)`: simulate trading in market `id` based on contribution weighted model `cwm`;
5. `trade_market_vol(id, W, G, ...)`: simulate trading in market `id` based on volume weighting of all submit order flows;
6. `trade_market_eq(id, W, G, ...)`: simulate trading in market `id` based on equal weighting of all submit order flows;
7. `trade_market_herd(id, W, G, ...)`: simulate trading in market `id` based on herd trading of contributing traders;
8. `trade_market_rand(id, W, G, ...)`: simulate trading in market `id` based on random decisions (as control);
9. `backtest(ids, strat, **kwargs)`: backtest `strat` in markets `ids` with parameters `**kwargs`.

2.7 Examples

We illustrate the performances of the four strategies, namely `cwm`, `vol`, `eq` and `herd`, on a representative market 1056, analyzed in section 1.2. The former three strategies adopt a belief distribution and the deviations are traded upon. Comparing with the (control) herding strategy, they all outperform with a rough cumulative P&L 100 at the termination of market.

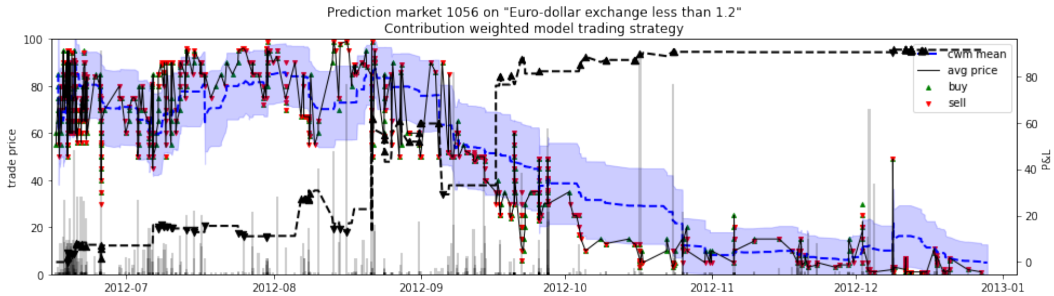
Along the price series, we construct a belief band, where the centered blue dashed line represents our mean belief $\hat{q}_t$ and the surrounding band represents our trigger level $\pm K \cdot \hat{\sigma}_t$, so that we take advantage of deviating, noisy trades occurring outside the band. The up/down black arrows indicate our buy/sell decisions, which are a function of the noisy trades and our position cap P. As time progresses, order flow information from contributing traders are taken in, via our exponential updates for parameters α_t, β_t, causing both the mean $\hat{q}_t$ and s.d. $\hat{\sigma}_t$ to evolve. As expected, $\hat{q}_t$ converges to the market outcome at market close and $\hat{\sigma}_t$ narrows down, meaning our belief becomes more certain over time. Note that by our assumption, the simulated trades occur precisely at timestamps where actual trades occur, and the volume traded does not exceed half the traded volume (as the counterparties take our orders), subject to our position cap.

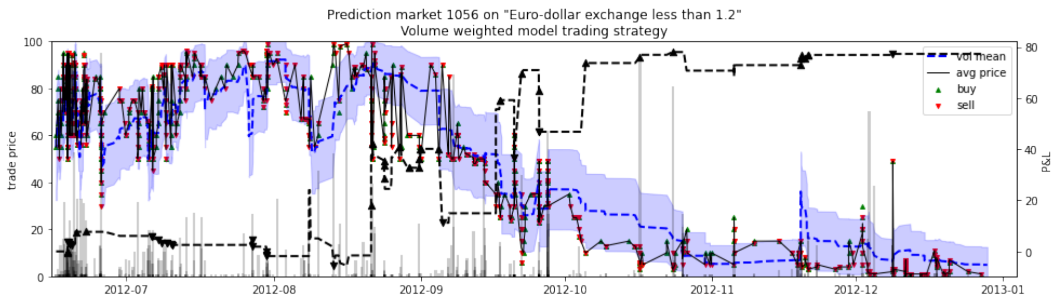
Here we fix $K = 0.5$ and we see that most trades occur within our bands. Those outside are traded, with the speculation that the deviation is only temporary and later trades will occur around our belief bands.

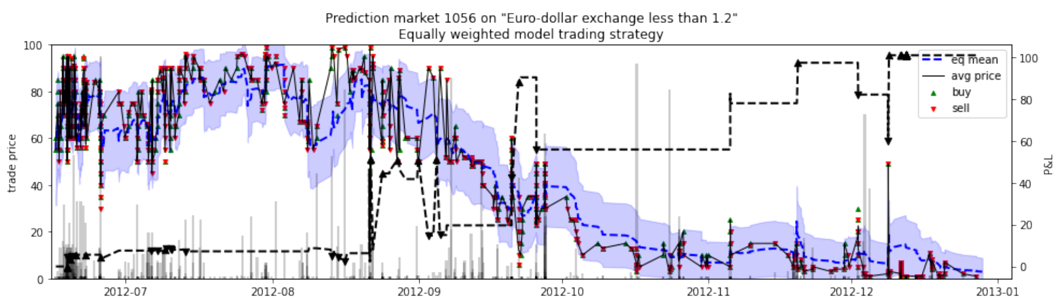
As for P&L series, where the P&L at a certain time point is marked to the market trade price, it is observed to rise steadily. Although the total profit is not significant, the strategy is extremely safe with no significant drawdown. At this point, the strategy parameters have not been optimized yet and they take the following typical values:

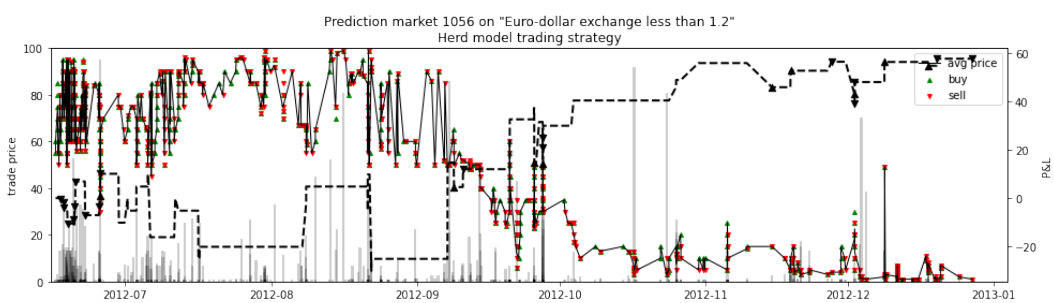
- `cwm`: `W=1000, G=2, C=2, K=0.5, T=0.05, P=100`;
- `vol`: `W=1000, G=2, C=2, K=0.5, T=0.05, P=100, wmax=0.1`;

- eq: `W=1000, G=2, C=2, K=0.5, T=0.05, P=100, w0=0.05`;
- herd: `W=1000, T=0.05, P=100, pmax=0.8`.









3 Strategy P&Ls

Each strategy comes with its own set of model parameters. For example, the contribution weighted model (`cwm`) is parametrized by C, K, W, G. To optimize the parameters, we train the strategy over some subset of markets, called the "train markets", by maximizing the per market P&L over a grid of model parameters. Then, we evaluate the strategy performances out-of-sample, over another disjoint subset of markets, called the “test markets”, by calculating the cumulative P&L, given the trained parameters. We divide all 93 liquid binary markets, sorted in their IFP ids, into a 60:40 split, respectively for the train and test markets.

Training is performed over only the liquid markets, because (1) this narrows down our training sample size, thus parameters may be optimized faster, and (2) with high liquidity, there is a wider sample of order flow, so more information may be taken into our model (model parameters are more frequently updated), enabling a higher certainty of model parameters thus P&Ls. Testing is performed over markets sorted in descending order of their liquidity, i.e. the total number of orders submitted. Discussion of the training and testing results below.

3.1 Model Training

Consider specifically training over top 5 and 10 liquid markets, for each strategy `cwm`, `vol`, `eq` and `herd`, where fixing the initial parameters at their typical values detailed in section 2.7:

1. for `cwm`, we first optimize over C, K, then W, G;
2. for `vol`, we first optimize $w_{\max}$, then C, K, finally W, G;
3. for `eq`, we first optimize w_0, then C, K, finally W, G;
4. for `herd`, we optimize $w_{\max}$.

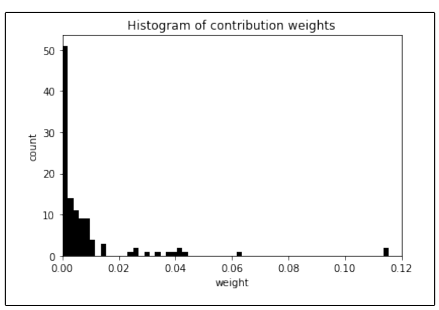
Recall that C, K relate to trading strategy while W, G relate to belief calculation. Thus, intuitively P&L is most sensitive to C, K and less to W, G, so we choose to optimize over C, K first.

The IFP ids of the top 10 liquid markets, in ascending total submit orders, read:

```
['1165', '1105', '1098', '1128', '1095', '1114', '1109', '1051', '1159', '1056']
```

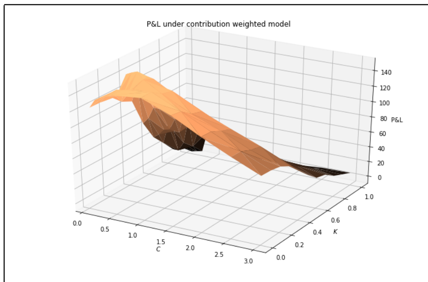
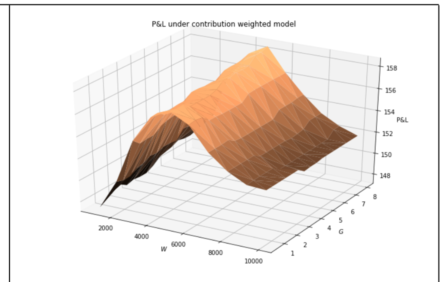
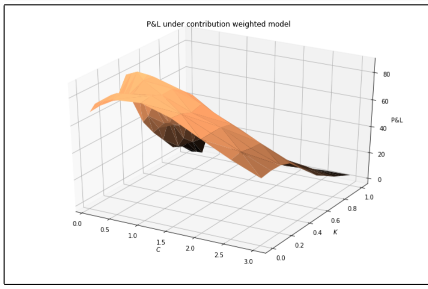
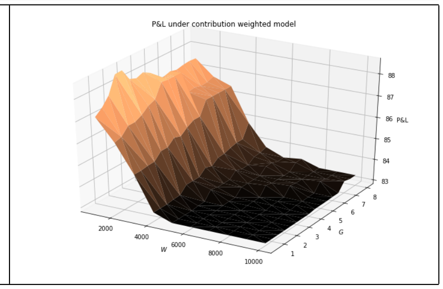
3.1.1 Contribution Weighted Model

First, we obtain the contribution weights of the contributing traders by considering their cumulative historical profits over the train markets. The weights are distributed as follows, with a long right-tail.



By optimizing over the top 5 and 10 liquid markets, we obtain the following P&L surface, where the z-axis indicates the per market P&L given a certain set of parameters. We look for the parameter values s.t. the P&L is maximized. For `cwm`, we have

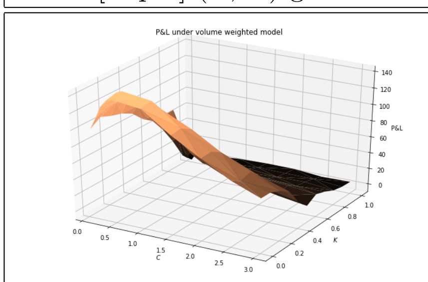
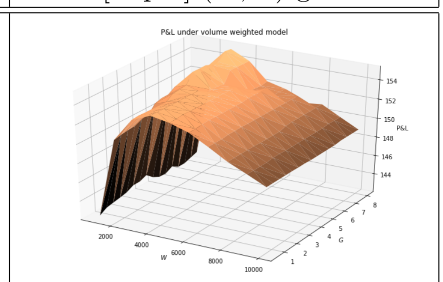
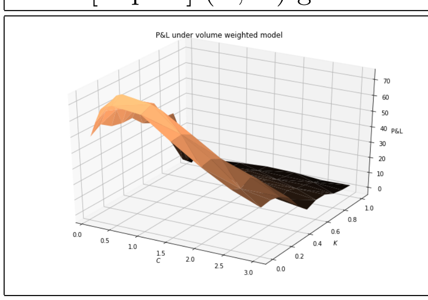
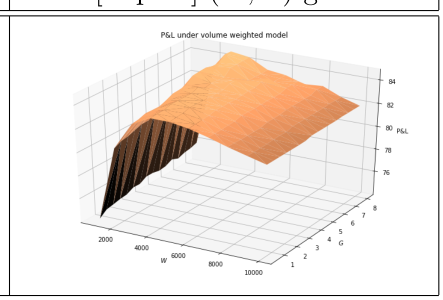
1. Top 5 liquid markets: $C = 1, K = 0.1, W = 5000, G = 2$, with per market P&L maximized at 156.85;
2. Top 10 liquid markets: $C = 0.4, K = 0.3, W = 1000, G = 2$, with per market P&L maximized at 88.57.

[Top 5] (C, K) grid	[Top 5] (W, G) grid
	
[Top 10] (C, K) grid	[Top 10] (W, G) grid
	

3.1.2 Volume Weighted Model

Repeating similar procedure, for `vol`, we have

1. Top 5 liquid markets: $w_{\max} = 0.25, C = 0.5, K = 0.1, W = 4000, G = 7$, with per market P&L maximized at 154.16;
2. Top 10 liquid markets: $w_{\max} = 0.25, C = 0.5, K = 0.1, W = 4000, G = 7$, with per market P&L maximized at 84.38.

[Top 5] (C, K) grid	[Top 5] (W, G) grid
	
[Top 10] (C, K) grid	[Top 10] (W, G) grid
	

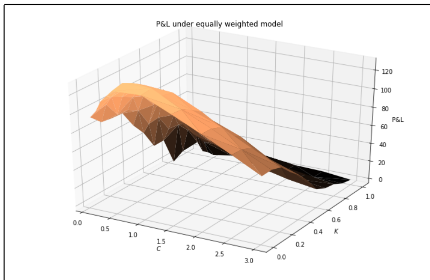
3.1.3 Equally Weighted Model

Repeating similar procedure, for `eq`, we have

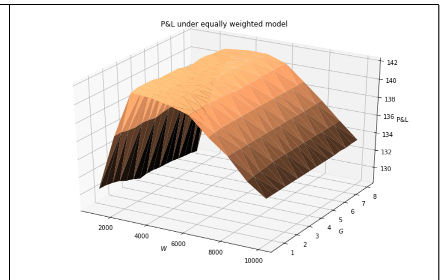
1. Top 5 liquid markets: $w_0 = 0.04, C = 1, K = 0.1, W = 3000, G = 2$, with per market P&L maximized at 142.03;

2. Top 10 liquid markets: $w_0 = 0.04, C = 0.5, K = 0.1, W = 3000, G = 2$, with per market P&L maximized at 73.15.

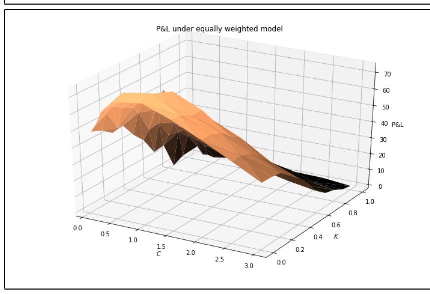
[Top 5] (C, K) grid



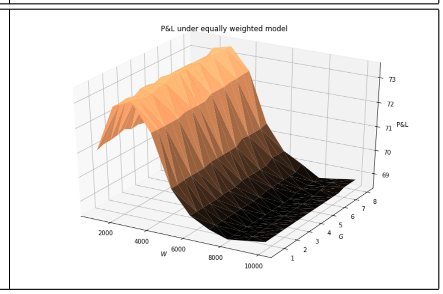
[Top 5] (W, G) grid



[Top 10] (C, K) grid



[Top 10] (W, G) grid



3.1.4 Herd Model

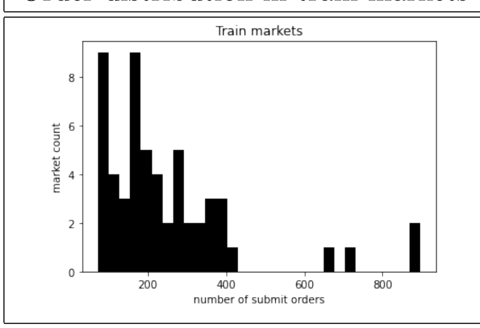
We use the herd model as the control/baseline strategy, which simply looks at the submit orders of contributing traders and follows with a certain probability correlated with the contributing weights. The maximum probability allowed is indicated by $p_{\max}$.

We observe that over a range of $p_{\max}$ spanning $[0.05, 1]$, for both top 5 and 10 liquid markets, the per market P&L oscillates roughly between $[10, 40]$, and there is no monotonic pattern. Thus, we conclude that the herd model does not compete with the belief-based strategies analyzed above.

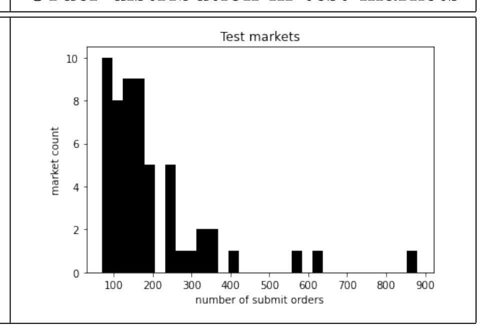
3.2 Model Testing

As we train the strategies over liquid markets, in the test markets, we expect that they work better for liquid than illiquid ones. First, we inspect the distribution of liquidity across the train and test markets – they are seen to distribute roughly identically, so the performances in train markets are expected to carry over to test markets.

Order distribution in train markets



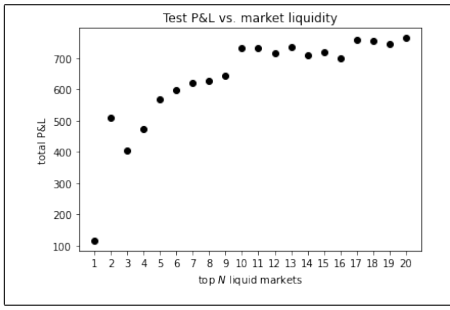
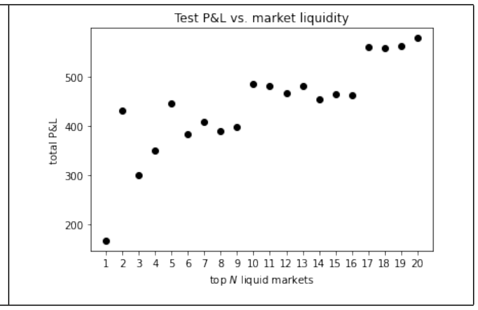
Order distribution in test markets



To examine the performances, we compute the total P&L over the top N liquid test markets, with N progressively growing from 1 to 20.

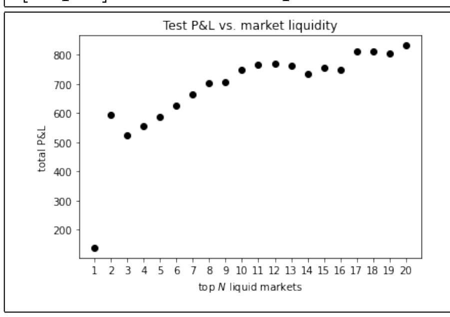
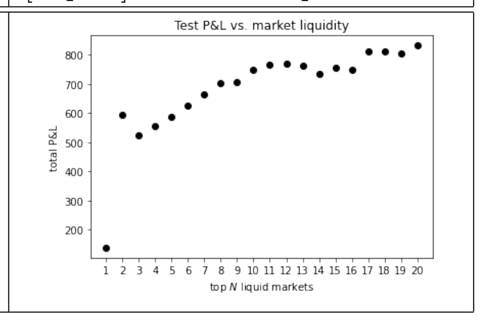
3.2.1 Contribution Weighted Model

The total P&L rises almost monotonically as N increases, stabling off at 700 with parameters trained under the top 5 liquid markets, and 600 with parameters trained under the top 10 liquid markets. Averaging, `cwm` yields 650 total profit out-of-sample.

[Top 5] Test market performance	[Top 10] Test market performance
	

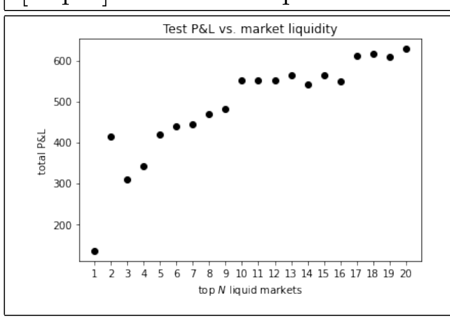
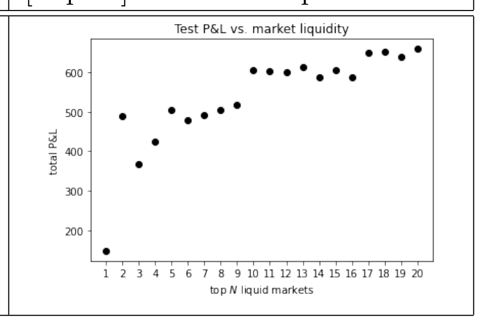
3.2.2 Volume Weighted Model

Again, the total P&L rises monotonically, with similar pattern for both top 5 and 10 markets, stabling off at 800. So, `vol` yields 800 total profit out-of-sample.

[Top 5] Test market performance	[Top 10] Test market performance
	

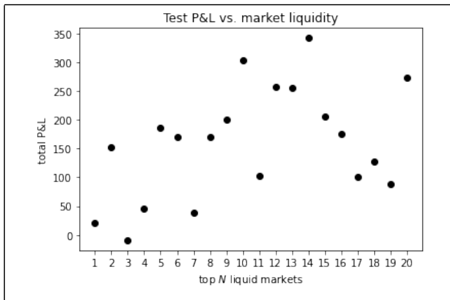
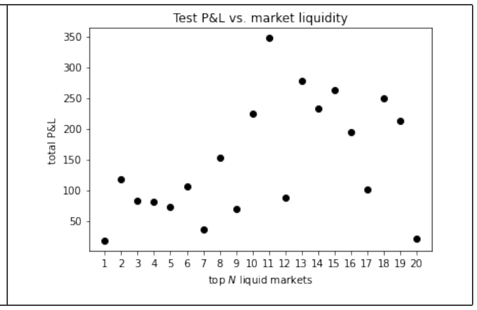
3.2.3 Equally Weighted Model

Again, the total P&L rises monotonically, with similar pattern for both top 5 and 10 markets, stabling off at 600. So, `eq` yields 600 total profit out-of-sample.

[Top 5] Test market performance	[Top 10] Test market performance
	

3.3 Herd Model

The total P&L oscillates with no clear pattern, averaging at roughly 150. Hence, the herd model, as the control/baseline, underperforms our belief-based strategies.

[Top 5] Test market performance	[Top 10] Test market performance
	

3.4 Comparison with Market Participants

In the top 5 liquid markets, considering the per market P&L, our contribution weighted model yields 113.58, while the market participants have an average P&L of -36.63 with standard deviation 1108.09. This places us at the 78th percentile.

In the top 10 liquid markets, considering the per market P&L, our contribution weighted model yields 47.67, while the market participants have an average P&L of -10.09 with standard deviation 685.35. This places us at the 76th percentile.

The volume weighted model slightly beats the contribution weighted model, placing us at respectively 79th and 81th percentile for top 5 and 10 markets.

4 Conclusion

We propose the contribution weighted model which establishes a belief distribution based on the order flow of individual "wise" traders, who perform historically in the train markets, and trades according to the deviating orders. As benchmarks, we also consider the volume and equally weighted model. These fall under the umbrella of belief-based strategies. Compared to the herding strategy serving as the control/baseline, they all outperform. Among all belief-based strategies, volume weighted model performs the best, placing us at the 80th percentile among all market participants in the test markets. While the total profit is not significant, the strategies are extremely safe with limited drawdown and generally yield positive but with low correlation to the market (we do not take directional bets), given sufficient liquidity in the prediction market. The profit may be interpreted as a premium for price discovery.

5 Codes

The datasets used are `ifps.csv` and `pm_transactions.lum1.yr2.csv`, inside the `data` folder. All the results and plots in this report can be replicated by running the python notebook `project.ipynb` attached. For completeness, here we present the core functions.

5.1 Data Import

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```python
from tqdm import tqdm

def init():
    # import data and filter for liquid binary markets
    df_ifp = pd.read_csv('/content/drive/MyDrive/IEOR4725 Prediction Market/data/ifps.csv', encoding='unicode_escape')
    df_ord = pd.read_csv('/content/drive/MyDrive/IEOR4725 Prediction Market/data/pm_transactions.lum1.yr2.csv')

    df_ifp['ifp_id'] = df_ifp['ifp_id'].apply(lambda x: x.split('-')[0])
    df_ord['IFPID'] = df_ord['IFPID'].astype('str').apply(lambda x: x.split('.')[0])
    df_ord['matching.order.ID'] = df_ord['matching.order.ID'].astype('str').apply(lambda x: x.split('.')[0])
    df_ord['timestamp'] = pd.to_datetime(df_ord['timestamp'])

    print('columns:')
    print(df_ifp.columns)
    print(df_ord.columns)
    print('-'*50)

    print('preview:')
    print(df_ifp.head())
    print(df_ord.head())
    print('-'*50)

    ifp = dict()
    ord_id = df_ord['IFPID'].to_list()
    for i, row in df_ifp.iterrows():
        id = row['ifp_id']
        if id in ord_id:
            log = df_ord[df_ord['IFPID']==id]

            order_log = log[log['Op.Type']=='SubmitOrder'].copy()
            idx_buy = ~(order_log['buy']^order_log['long'])
            order_log['buy_sell'] = idx_buy
            order_log['signed_qty'] = order_log['qty']*(2*idx_buy-1)

            trade_log = log[log['Op.Type']=='Trade'].copy()
            idx_buy = ~(trade_log['buy']^trade_log['long'])
            trade_log['buy_sell'] = idx_buy
            trade_log['signed_qty'] = trade_log['qty']*(2*idx_buy-1)

            wm = lambda x: np.average(x, weights=trade_log.loc[x.index,'qty'])
            trade_agg = trade_log.groupby('timestamp').agg(price=('price',wm), qty=('qty','sum'))

            trade_cnt = trade_log.groupby('user.ID').size()

            n_trades = len(trade_log)
            n_orders = len(order_log)

            usr_ids = log['user.ID'].unique()
            usr_log = dict()
            for usr in usr_ids:
                usr_log[usr] = log[log['user.ID']==usr]

            ifp[id] = {
                'short_title':  row['short_title'],
                'q_text':       row['q_text'],
                'date_start':   row['date_start'],
                'date_suspend': row['date_suspend'],
```

```python
                'options':       row['options'],
                'n_opts':        row['n_opts'],
                'outcome':       row['outcome'],
                'market': {
                    'log':           log,
                    'order_log':     order_log,
                    'trade_log':     trade_log,
                    'trade_agg':     trade_agg,
                    'trade_cnt':     trade_cnt,
                    'n_trades':      n_trades,
                    'n_orders':      n_orders,
                    'usr_ids':       usr_ids,
                    'usr_log':       usr_log,
                },
            }

    ids = list(ifp.keys())
    print('ids (first 5):', ids[:5])
    print('-'*50)

    n_orders = [ifp[id]['market']['n_orders'] for id in ids]
    ord_thres = np.percentile(n_orders, 10)
    print('ord_thres (bottom 10%):', ord_thres)
    print('-'*50)

    print('histogram of n_orders:')
    plt.hist(n_orders, bins=50, color='k')
    plt.xlabel('number of submit orders')
    plt.ylabel('market count')
    plt.show()
    print('-'*50)

    ifp_liquid = {id: ifp[id] for id in ids if ifp[id]['market']['n_orders']>ord_thres
    and ifp[id]['n_opts']==2}
    ids_liquid = list(ifp_liquid.keys())

    return ifp_liquid, ids_liquid

ifp_liquid, ids_liquid = init()
```

5.2 Exploratory Data Analysis

```python
def market_summary(id):
    # summary of single market
    if id in ids_liquid:
        log = ifp_liquid[id]['market']['log']
        trade_log = ifp_liquid[id]['market']['trade_log']
        idx_buy = trade_log['buy_sell']
        idx_sell = ~idx_buy

        t0 = log['timestamp'].iloc[0]+pd.DateOffset(-1)
        t1 = log['timestamp'].iloc[-1]+pd.DateOffset(1)
        q = ifp_liquid[id]['short_title']
        mkt_agg = ifp_liquid[id]['market']['trade_agg']

        trade_cnt = ifp_liquid[id]['market']['trade_cnt']
        usr_ids = trade_cnt.sort_values().index[-5:]

        pnl = {'timestamp': mkt_agg.index}

        for usr in usr_ids:
```

```
        usr_log = trade_log[trade_log['user.ID']==usr]

        cash = list() # cash
        posn = list() # contract position
        mktv = list() # contract market value
        port = list() # portfolio value

        for t in mkt_agg.index:
            usr_log_t = usr_log[usr_log['timestamp']<=t]
            cash_t = -(usr_log_t['price']*usr_log_t['signed_qty']).sum()
            posn_t = usr_log_t['signed_qty'].sum()
            mktv_t = posn_t*mkt_agg.loc[t]['price']
            port_t = cash_t+mktv_t
            cash.append(cash_t)
            posn.append(posn_t)
            mktv.append(mktv_t)
            port.append(port_t)

        pnl[usr] = port

    pnl = pd.DataFrame(pnl)
    # print(pnl)

    pnl_T = dict()

    for usr in ifp_liquid[id]['market']['usr_ids']:
        usr_log = trade_log[trade_log['user.ID']==usr]
        cash_T = -(usr_log['price']*usr_log['signed_qty']).sum()
        posn_T = usr_log['signed_qty'].sum()
        mktv_T = posn_T*mkt_agg.iloc[-1]['price']
        port_T = cash_T+mktv_T
        pnl_T[usr] = port_T

    pnl_T = pd.Series(pnl_T)
    # print(pnl_T)

    print(len(pnl_T), pnl_T.mean(), pnl_T.std())

    print('-'*50)
    print('id:           ', id)
    print('q_text:       ', ifp_liquid[id]['q_text'])
    print('date_start:   ', ifp_liquid[id]['date_start'])
    print('date_suspend:', ifp_liquid[id]['date_suspend'])
    print('outcome:      ', ifp_liquid[id]['outcome'])
    print('n_traders:    ', len(ifp_liquid[id]['market']['usr_ids']))
    print('n_orders:     ', ifp_liquid[id]['market']['n_orders'])
    print('n_trades:     ', ifp_liquid[id]['market']['n_trades'])
    print('n_buys:       ', sum(idx_buy))
    print('n_sells:      ', sum(idx_sell))
    print('-'*50)

    fig, ax1 = plt.subplots(figsize=(16,4))
    ax1.scatter(trade_log[idx_buy]['timestamp'], trade_log[idx_buy]['price'], s=15, c='g', marker='^', label='buy')
    ax1.scatter(trade_log[idx_sell]['timestamp'], trade_log[idx_sell]['price'], s=15, c='r', marker='v', label='sell')
    ax1.plot(mkt_agg.index, mkt_agg['price'], 'k', lw=1, label='avg price')
    ax2 = ax1.twinx()
    ax2.bar(mkt_agg.index, mkt_agg['qty'], color='k', width=0.5, alpha=0.2, label='volume')
    ax1.set_title(f'Prediction market {id} on "{q}"\nTrades from {t0.date()} to {t1
```

```
.date()}')
        ax1.set_xlim(t0,t1)
        ax1.set_ylim(0,100)
        ax1.set_ylabel('trade price')
        ax2.set_ylabel('volume')
        ax1.legend()
        plt.show()

        fig = plt.figure(figsize=(16,4))
        for usr in usr_ids:
            plt.plot(pnl['timestamp'], pnl[usr], lw=1, label=f'id={usr} ({trade_cnt[usr
]})')
        plt.title(f'Prediction market {id} on "{q}"\nP&Ls of top {len(usr_ids)}
frequent traders')
        # plt.xticks(rotation=30)
        plt.xlim(t0,t1)
        plt.ylabel('P&L')
        plt.legend()
        plt.show()

        fig = plt.figure(figsize=(6,4))
        plt.hist(pnl_T, bins=50, color='k')
        plt.title(f'Prediction market {id} on "{q}"\nP&L histogram of all traders')
        plt.xlabel('P&L')
        plt.ylabel('trader count')
        plt.show()
```

```
def market_summary(id):
    # summary of single market
    if id in ids_liquid:
        log = ifp_liquid[id]['market']['log']
        trade_log = ifp_liquid[id]['market']['trade_log']
        idx_buy = trade_log['buy_sell']
        idx_sell = ~idx_buy

        t0 = log['timestamp'].iloc[0]+pd.DateOffset(-1)
        t1 = log['timestamp'].iloc[-1]+pd.DateOffset(1)
        q = ifp_liquid[id]['short_title']
        mkt_agg = ifp_liquid[id]['market']['trade_agg']

        trade_cnt = ifp_liquid[id]['market']['trade_cnt']
        usr_ids = trade_cnt.sort_values().index[-5:]

        pnl = {'timestamp': mkt_agg.index}

        for usr in usr_ids:
            usr_log = trade_log[trade_log['user.ID']==usr]

            cash = list() # cash
            posn = list() # contract position
            mktv = list() # contract market value
            port = list() # portfolio value

            for t in mkt_agg.index:
                usr_log_t = usr_log[usr_log['timestamp']<=t]
                cash_t = -(usr_log_t['price']*usr_log_t['signed_qty']).sum()
                posn_t = usr_log_t['signed_qty'].sum()
                mktv_t = posn_t*mkt_agg.loc[t]['price']
                port_t = cash_t+mktv_t
                cash.append(cash_t)
                posn.append(posn_t)
                mktv.append(mktv_t)
```

```
            port.append(port_t)

        pnl[usr] = port

    pnl = pd.DataFrame(pnl)
    # print(pnl)

    pnl_T = dict()

    for usr in ifp_liquid[id]['market']['usr_ids']:
        usr_log = trade_log[trade_log['user.ID']==usr]
        cash_T = -(usr_log['price']*usr_log['signed_qty']).sum()
        posn_T = usr_log['signed_qty'].sum()
        mktv_T = posn_T*mkt_agg.iloc[-1]['price']
        port_T = cash_T+mktv_T
        pnl_T[usr] = port_T

    pnl_T = pd.Series(pnl_T)
    # print(pnl_T)

    print(len(pnl_T), pnl_T.mean(), pnl_T.std())

    print('-'*50)
    print('id:           ', id)
    print('q_text:       ', ifp_liquid[id]['q_text'])
    print('date_start:   ', ifp_liquid[id]['date_start'])
    print('date_suspend:', ifp_liquid[id]['date_suspend'])
    print('outcome:      ', ifp_liquid[id]['outcome'])
    print('n_traders:    ', len(ifp_liquid[id]['market']['usr_ids']))
    print('n_orders:     ', ifp_liquid[id]['market']['n_orders'])
    print('n_trades:     ', ifp_liquid[id]['market']['n_trades'])
    print('n_buys:       ', sum(idx_buy))
    print('n_sells:      ', sum(idx_sell))
    print('-'*50)

    fig, ax1 = plt.subplots(figsize=(16,4))
    ax1.scatter(trade_log[idx_buy]['timestamp'], trade_log[idx_buy]['price'], s=15, c='g', marker='^', label='buy')
    ax1.scatter(trade_log[idx_sell]['timestamp'], trade_log[idx_sell]['price'], s=15, c='r', marker='v', label='sell')
    ax1.plot(mkt_agg.index, mkt_agg['price'], 'k', lw=1, label='avg price')
    ax2 = ax1.twinx()
    ax2.bar(mkt_agg.index, mkt_agg['qty'], color='k', width=0.5, alpha=0.2, label='volume')
    ax1.set_title(f'Prediction market {id} on "{q}"\nTrades from {t0.date()} to {t1.date()}')
    ax1.set_xlim(t0,t1)
    ax1.set_ylim(0,100)
    ax1.set_ylabel('trade price')
    ax2.set_ylabel('volume')
    ax1.legend()
    plt.show()

    fig = plt.figure(figsize=(16,4))
    for usr in usr_ids:
        plt.plot(pnl['timestamp'], pnl[usr], lw=1, label=f'id={usr} ({trade_cnt[usr]})')
    plt.title(f'Prediction market {id} on "{q}"\nP&Ls of top {len(usr_ids)} frequent traders')
    # plt.xticks(rotation=30)
    plt.xlim(t0,t1)
```

```
        plt.ylabel('P&L')
        plt.legend()
        plt.show()

        fig = plt.figure(figsize=(6,4))
        plt.hist(pnl_T, bins=50, color='k')
        plt.title(f'Prediction market {id} on "{q}"\nP&L histogram of all traders')
        plt.xlabel('P&L')
        plt.ylabel('trader count')
        plt.show()
```

5.3 Trading Strategies

```
def belief(price, demand, W, G):
    p = price
    n = demand
    b = (1+n/(W-n*p))**G
    q = p*b/(1+p*(b-1))
    return q

def demand(price, belief, W, G):
    p = price
    q = belief
    a = ((q*(1-p))/(p*(1-q)))**(1/G)
    n = W*(a-1)/(1+p*(a-1))
    return n
```

```
def trade_market_cwm(id, weight, W, G, C, K=0.5, T=0, P=None, plot=False):
    # simulate trading in market id based on contribution weighted model
    if not P: P = np.Inf

    log = ifp_liquid[id]['market']['log']
    order_log = ifp_liquid[id]['market']['order_log'].copy()
    trade_log = ifp_liquid[id]['market']['trade_log']
    mkt_agg = ifp_liquid[id]['market']['trade_agg']

    t0 = log['timestamp'].iloc[0]+pd.DateOffset(-1)
    t1 = log['timestamp'].iloc[-1]+pd.DateOffset(1)
    q = ifp_liquid[id]['short_title']

    idx_buy = trade_log['buy_sell']
    idx_sell = ~idx_buy

    # construct time series of belief based on submit order flow
    order_log['belief'] = belief(order_log['price']/100, order_log['signed_qty'], W, G)
    order_log['weight'] = order_log['user.ID'].apply(lambda x: weight[x] if x in weight
     else 0)

    a = list() # alpha of Beta-distributed belief
    b = list() # beta of Beta-distributed belief
    for i, row in order_log.reset_index().iterrows():
        w = C * row['weight']
        if i == 0:
            a.append(w*row['belief'])
            b.append(w*(1-row['belief']))
        else:
            a.append((1-w)*a[i-1]+w*row['belief'])
            b.append((1-w)*b[i-1]+w*(1-row['belief']))
    order_log['alpha'] = a
    order_log['beta'] = b

```

```
    order_log['belief_mean'] = order_log['alpha']/(order_log['alpha']+order_log['beta'])
    order_log['belief_sd'] = np.sqrt((order_log['alpha']*order_log['beta'])/(order_log['alpha']+order_log['beta']+1))/(order_log['alpha']+order_log['beta'])
    # print(order_log)

    # simulate trading strategy
    cash = 0
    posn = 0
    mktv = 0
    port = 0
    pnl_ts = dict()

    for t, row in mkt_agg.iterrows():
        order_log_t = order_log[order_log['timestamp']<=t]

        p_t = row['price']/100
        v_t = row['qty']
        q_t = order_log_t['belief_mean'].iloc[-1]
        sd_t = order_log_t['belief_sd'].iloc[-1]
        s_t = (p_t-q_t)/sd_t

        n_t = demand(p_t, q_t, W/100, C)
        n_t = np.round(np.sign(n_t) * min(np.abs(n_t), v_t/2))
        n_t = max(min(n_t, P-posn), -P-posn)

        if s_t < -K or s_t > K:
            cash -= n_t*p_t + np.abs(n_t*p_t)*T
            posn += n_t
            mktv = posn*p_t
            port = cash+mktv
        else:
            n_t = 0

        pnl_ts[t] = {'n': n_t, 'cash': cash, 'posn': posn, 'mktv': mktv, 'port': port}

    pnl_ts = pd.DataFrame(pnl_ts).T

    sim = {
        'info': {
            'strat': 'cwm',
            'id': id,
            'W': W,
            'G': G,
            'C': C,
            'K': K,
            'T': T,
            'P': P,
        },
        'pnl': {
            'cash': cash,
            'posn': posn,
            'mktv': mktv,
            'port': port,
        },
        'pnl_ts': pnl_ts,
    }

    if plot:
        fig, ax1 = plt.subplots(figsize=(16,4))
        ax1.scatter(trade_log[idx_buy]['timestamp'], trade_log[idx_buy]['price'], s=15,
```

```python
    c='g', marker='^', label='buy')
        ax1.scatter(trade_log[idx_sell]['timestamp'], trade_log[idx_sell]['price'], s
    =15, c='r', marker='v', label='sell')
        ax1.plot(order_log['timestamp'], 100*order_log['belief_mean'], 'b--', lw=2,
    label='cwm mean')
        plt.fill_between(order_log['timestamp'], 100*(order_log['belief_mean']-K*
    order_log['belief_sd']), 100*(order_log['belief_mean']+K*order_log['belief_sd']),
    color='b', alpha=0.2)
        ax1.plot(mkt_agg.index, mkt_agg['price'], 'k', lw=1, label='avg price')
        ax2 = ax1.twinx()
        ax2.bar(mkt_agg.index, mkt_agg['qty'], color='k', width=0.5, alpha=0.2)
        ax3 = ax1.twinx()
        ax3.plot(pnl_ts.index, pnl_ts['port'], 'k--', lw=2)
        ax3.scatter(pnl_ts[pnl_ts['n']>0].index, pnl_ts[pnl_ts['n']>0]['port'], s=40, c
    ='k', marker='^')
        ax3.scatter(pnl_ts[pnl_ts['n']<0].index, pnl_ts[pnl_ts['n']<0]['port'], s=40, c
    ='k', marker='v')
        ax1.set_title(f'Prediction market {id} on "{q}"\nContribution weighted model
    trading strategy')
        ax1.set_xlim(t0,t1)
        ax1.set_ylim(0,100)
        ax1.set_ylabel('trade price')
        ax3.set_ylabel('P&L')
        ax2.set_yticks([])
        ax1.legend()
        plt.show()

    return sim
```

```python
def trade_market_vol(id, W, G, C, K=0.5, T=0, P=None, wmax=0.5, plot=False):
    # simulate trading in market id based on volume weighting of all submit order flows
    if not P: P = np.Inf

    log = ifp_liquid[id]['market']['log']
    order_log = ifp_liquid[id]['market']['order_log'].copy()
    trade_log = ifp_liquid[id]['market']['trade_log']
    mkt_agg = ifp_liquid[id]['market']['trade_agg']

    t0 = log['timestamp'].iloc[0]+pd.DateOffset(-1)
    t1 = log['timestamp'].iloc[-1]+pd.DateOffset(1)
    q = ifp_liquid[id]['short_title']

    idx_buy = trade_log['buy_sell']
    idx_sell = ~idx_buy

    # construct time series of belief based on submit order flow
    order_log['belief'] = belief(order_log['price']/100, order_log['signed_qty'], W, G)
    order_log['weight'] = np.minimum(order_log['qty']/order_log['qty'].cummax(), wmax)

    a = list() # alpha of Beta-distributed belief
    b = list() # beta of Beta-distributed belief
    for i, row in order_log.reset_index().iterrows():
        w = C * row['weight']
        if i == 0:
            a.append(w*row['belief'])
            b.append(w*(1-row['belief']))
        else:
            a.append((1-w)*a[i-1]+w*row['belief'])
            b.append((1-w)*b[i-1]+w*(1-row['belief']))
    order_log['alpha'] = a
    order_log['beta'] = b

```

```
    order_log['belief_mean'] = order_log['alpha']/(order_log['alpha']+order_log['beta'
    ])
    order_log['belief_sd'] = np.sqrt((order_log['alpha']*order_log['beta'])/(order_log[
    'alpha']+order_log['beta']+1))/(order_log['alpha']+order_log['beta'])
    # print(order_log)

    # simulate trading strategy
    cash = 0
    posn = 0
    mktv = 0
    port = 0
    pnl_ts = dict()

    for t, row in mkt_agg.iterrows():
        order_log_t = order_log[order_log['timestamp']<=t]

        p_t = row['price']/100
        v_t = row['qty']
        q_t = order_log_t['belief_mean'].iloc[-1]
        sd_t = order_log_t['belief_sd'].iloc[-1]
        s_t = (p_t-q_t)/sd_t

        n_t = demand(p_t, q_t, W/100, C)
        n_t = np.round(np.sign(n_t) * min(np.abs(n_t), v_t/2))
        n_t = max(min(n_t, P-posn), -P-posn)

        if s_t < -K or s_t > K:
            cash -= n_t*p_t + np.abs(n_t*p_t)*T
            posn += n_t
            mktv = posn*p_t
            port = cash+mktv
        else:
            n_t = 0

        pnl_ts[t] = {'n': n_t, 'cash': cash, 'posn': posn, 'mktv': mktv, 'port': port}

    pnl_ts = pd.DataFrame(pnl_ts).T

    sim = {
        'info': {
            'strat': 'cwm',
            'id': id,
            'W': W,
            'G': G,
            'C': C,
            'K': K,
            'T': T,
            'P': P,
            'wmax': wmax,
        },
        'pnl': {
            'cash': cash,
            'posn': posn,
            'mktv': mktv,
            'port': port,
        },
        'pnl_ts': pnl_ts,
    }

    if plot:
        fig, ax1 = plt.subplots(figsize=(16,4))
```

```
        ax1.scatter(trade_log[idx_buy]['timestamp'], trade_log[idx_buy]['price'], s=15, c='g', marker='^', label='buy')
        ax1.scatter(trade_log[idx_sell]['timestamp'], trade_log[idx_sell]['price'], s=15, c='r', marker='v', label='sell')
        ax1.plot(order_log['timestamp'], 100*order_log['belief_mean'], 'b--', lw=2, label='cwm mean')
        plt.fill_between(order_log['timestamp'], 100*(order_log['belief_mean']-K*order_log['belief_sd']), 100*(order_log['belief_mean']+K*order_log['belief_sd']), color='b', alpha=0.2)
        ax1.plot(mkt_agg.index, mkt_agg['price'], 'k', lw=1, label='avg price')
        ax2 = ax1.twinx()
        ax2.bar(mkt_agg.index, mkt_agg['qty'], color='k', width=0.5, alpha=0.2)
        ax3 = ax1.twinx()
        ax3.plot(pnl_ts.index, pnl_ts['port'], 'k--', lw=2)
        ax3.scatter(pnl_ts[pnl_ts['n']>0].index, pnl_ts[pnl_ts['n']>0]['port'], s=40, c='k', marker='^')
        ax3.scatter(pnl_ts[pnl_ts['n']<0].index, pnl_ts[pnl_ts['n']<0]['port'], s=40, c='k', marker='v')
        ax1.set_title(f'Prediction market {id} on "{q}"\nVolume weighted model trading strategy')
        ax1.set_xlim(t0,t1)
        ax1.set_ylim(0,100)
        ax1.set_ylabel('trade price')
        ax3.set_ylabel('P&L')
        ax2.set_yticks([])
        ax1.legend()
        plt.show()

    return sim
```

```
def trade_market_eq(id, W, G, C, K=0.5, T=0, P=None, w0=0.05, plot=False):
    # simulate trading in market id based on equal weighting of all submit order flows
    if not P: P = np.Inf

    log = ifp_liquid[id]['market']['log']
    order_log = ifp_liquid[id]['market']['order_log'].copy()
    trade_log = ifp_liquid[id]['market']['trade_log']
    mkt_agg = ifp_liquid[id]['market']['trade_agg']

    t0 = log['timestamp'].iloc[0]+pd.DateOffset(-1)
    t1 = log['timestamp'].iloc[-1]+pd.DateOffset(1)
    q = ifp_liquid[id]['short_title']

    idx_buy = trade_log['buy_sell']
    idx_sell = ~idx_buy

    # construct time series of belief based on submit order flow
    order_log['belief'] = belief(order_log['price']/100, order_log['signed_qty'], W, G)
    order_log['weight'] = w0

    a = list() # alpha of Beta-distributed belief
    b = list() # beta of Beta-distributed belief
    for i, row in order_log.reset_index().iterrows():
        w = C * row['weight']
        if i == 0:
            a.append(w*row['belief'])
            b.append(w*(1-row['belief']))
        else:
            a.append((1-w)*a[i-1]+w*row['belief'])
            b.append((1-w)*b[i-1]+w*(1-row['belief']))
    order_log['alpha'] = a
    order_log['beta'] = b
```

```python

    order_log['belief_mean'] = order_log['alpha']/(order_log['alpha']+order_log['beta'])
    order_log['belief_sd'] = np.sqrt((order_log['alpha']*order_log['beta'])/(order_log['alpha']+order_log['beta']+1))/(order_log['alpha']+order_log['beta'])
    # print(order_log)

    # simulate trading strategy
    cash = 0
    posn = 0
    mktv = 0
    port = 0
    pnl_ts = dict()

    for t, row in mkt_agg.iterrows():
        order_log_t = order_log[order_log['timestamp']<=t]

        p_t = row['price']/100
        v_t = row['qty']
        q_t = order_log_t['belief_mean'].iloc[-1]
        sd_t = order_log_t['belief_sd'].iloc[-1]
        s_t = (p_t-q_t)/sd_t

        n_t = demand(p_t, q_t, W/100, C)
        n_t = np.round(np.sign(n_t) * min(np.abs(n_t), v_t/2))
        n_t = max(min(n_t, P-posn), -P-posn)

        if s_t < -K or s_t > K:
            cash -= n_t*p_t + np.abs(n_t*p_t)*T
            posn += n_t
            mktv = posn*p_t
            port = cash+mktv
        else:
            n_t = 0

        pnl_ts[t] = {'n': n_t, 'cash': cash, 'posn': posn, 'mktv': mktv, 'port': port}

    pnl_ts = pd.DataFrame(pnl_ts).T

    sim = {
        'info': {
            'strat': 'cwm',
            'id': id,
            'W': W,
            'G': G,
            'C': C,
            'K': K,
            'T': T,
            'P': P,
            'w0': w0,
        },
        'pnl': {
            'cash': cash,
            'posn': posn,
            'mktv': mktv,
            'port': port,
        },
        'pnl_ts': pnl_ts,
    }

    if plot:
```

```python
        fig, ax1 = plt.subplots(figsize=(16,4))
        ax1.scatter(trade_log[idx_buy]['timestamp'], trade_log[idx_buy]['price'], s=15, c='g', marker='^', label='buy')
        ax1.scatter(trade_log[idx_sell]['timestamp'], trade_log[idx_sell]['price'], s=15, c='r', marker='v', label='sell')
        ax1.plot(order_log['timestamp'], 100*order_log['belief_mean'], 'b--', lw=2, label='cwm mean')
        plt.fill_between(order_log['timestamp'], 100*(order_log['belief_mean']-K*order_log['belief_sd']), 100*(order_log['belief_mean']+K*order_log['belief_sd']), color='b', alpha=0.2)
        ax1.plot(mkt_agg.index, mkt_agg['price'], 'k', lw=1, label='avg price')
        ax2 = ax1.twinx()
        ax2.bar(mkt_agg.index, mkt_agg['qty'], color='k', width=0.5, alpha=0.2)
        ax3 = ax1.twinx()
        ax3.plot(pnl_ts.index, pnl_ts['port'], 'k--', lw=2)
        ax3.scatter(pnl_ts[pnl_ts['n']>0].index, pnl_ts[pnl_ts['n']>0]['port'], s=40, c='k', marker='^')
        ax3.scatter(pnl_ts[pnl_ts['n']<0].index, pnl_ts[pnl_ts['n']<0]['port'], s=40, c='k', marker='v')
        ax1.set_title(f'Prediction market {id} on "{q}"\nEqually weighted model trading strategy')
        ax1.set_xlim(t0,t1)
        ax1.set_ylim(0,100)
        ax1.set_ylabel('trade price')
        ax3.set_ylabel('P&L')
        ax2.set_yticks([])
        ax1.legend()
        plt.show()

    return sim
```

```python
def trade_market_herd(id, weight, T=0, P=None, pmax=0.8, plot=False):
    # simulate trading in market id based on herd trading of contributing traders
    if not P: P = np.Inf

    log = ifp_liquid[id]['market']['log']
    order_log = ifp_liquid[id]['market']['order_log'].copy()
    trade_log = ifp_liquid[id]['market']['trade_log']
    mkt_agg = ifp_liquid[id]['market']['trade_agg']

    t0 = log['timestamp'].iloc[0]+pd.DateOffset(-1)
    t1 = log['timestamp'].iloc[-1]+pd.DateOffset(1)
    q = ifp_liquid[id]['short_title']

    idx_buy = trade_log['buy_sell']
    idx_sell = ~idx_buy

    order_log['weight'] = order_log['user.ID'].apply(lambda x: weight[x] if x in weight else 0)
    order_log['trade_prob'] = np.minimum(order_log['weight']/order_log['weight'].max(), pmax)
    order_log['unif_rv'] = np.random.uniform(size=len(order_log))

    # simulate trading strategy
    cash = 0
    posn = 0
    mktv = 0
    port = 0
    pnl_ts = dict()

    for t, row in mkt_agg.iterrows():
        order_log_t = order_log[order_log['timestamp']<=t]
```

```
        p_t = row['price']/100
        v_t = row['qty']

        n_t = order_log_t.iloc[-1]['signed_qty']
        n_t = np.round(np.sign(n_t) * min(np.abs(n_t), v_t/2))
        n_t = max(min(n_t, P-posn), -P-posn)

        if order_log_t.iloc[-1]['unif_rv'] < order_log_t.iloc[-1]['trade_prob']:
            cash -= n_t*p_t + np.abs(n_t*p_t)*T
            posn += n_t
            mktv = posn*p_t
            port = cash+mktv
        else:
            n_t = 0

        pnl_ts[t] = {'n': n_t, 'cash': cash, 'posn': posn, 'mktv': mktv, 'port': port}

    pnl_ts = pd.DataFrame(pnl_ts).T

    sim = {
        'info': {
            'strat': 'cwm',
            'id': id,
            'T': T,
            'P': P,
            'pmax': pmax,
        },
        'pnl': {
            'cash': cash,
            'posn': posn,
            'mktv': mktv,
            'port': port,
        },
        'pnl_ts': pnl_ts,
    }

    if plot:
        fig, ax1 = plt.subplots(figsize=(16,4))
        ax1.scatter(trade_log[idx_buy]['timestamp'], trade_log[idx_buy]['price'], s=15, c='g', marker='^', label='buy')
        ax1.scatter(trade_log[idx_sell]['timestamp'], trade_log[idx_sell]['price'], s=15, c='r', marker='v', label='sell')
        ax1.plot(mkt_agg.index, mkt_agg['price'], 'k', lw=1, label='avg price')
        ax2 = ax1.twinx()
        ax2.bar(mkt_agg.index, mkt_agg['qty'], color='k', width=0.5, alpha=0.2)
        ax3 = ax1.twinx()
        ax3.plot(pnl_ts.index, pnl_ts['port'], 'k--', lw=2)
        ax3.scatter(pnl_ts[pnl_ts['n']>0].index, pnl_ts[pnl_ts['n']>0]['port'], s=40, c='k', marker='^')
        ax3.scatter(pnl_ts[pnl_ts['n']<0].index, pnl_ts[pnl_ts['n']<0]['port'], s=40, c='k', marker='v')
        ax1.set_title(f'Prediction market {id} on "{q}"\nHerd model trading strategy')
        ax1.set_xlim(t0,t1)
        ax1.set_ylim(0,100)
        ax1.set_ylabel('trade price')
        ax3.set_ylabel('P&L')
        ax2.set_yticks([])
        ax1.legend()
        plt.show()

```

```
    return sim
```

5.4 Model Training

```
def get_weight(ids):
    ifp_usr = {id: ifp_liquid[id]['market']['usr_ids'] for id in ifp_liquid}
    all_usr = np.unique(np.concatenate(list(ifp_usr.values())))
    cum_pnl = {usr: 0 for usr in all_usr}

    for id in tqdm(ids):
        trade_log = ifp_liquid[id]['market']['trade_log']
        mkt_agg = ifp_liquid[id]['market']['trade_agg']
        for usr in ifp_liquid[id]['market']['usr_ids']:
            usr_log = trade_log[trade_log['user.ID']==usr]
            cash_T = -(usr_log['price']*usr_log['signed_qty']).sum()
            posn_T = usr_log['signed_qty'].sum()
            mktv_T = posn_T*mkt_agg.iloc[-1]['price']
            port_T = cash_T+mktv_T
            cum_pnl[usr] += port_T

    cum_pnl = pd.Series(cum_pnl)
    w = cum_pnl[cum_pnl>0]
    w /= w.sum()

    print()
    print('% contributing traders:', 100*len(w)/len(all_usr))

    return w

def backtest(ids, strat, **kwargs):
    if strat == 'cwm':
        trade = trade_market_cwm
    elif strat == 'vol':
        trade = trade_market_vol
    elif strat == 'eq':
        trade = trade_market_eq
    elif strat == 'herd':
        trade = trade_market_herd
    else:
        return
    sims = dict()
    for id in ids:
        sim = trade(id, **kwargs)
        sims[id] = sim
    return sims

def pnl_grid_over_CK(ids, strat, CC=None, KK=None, **kwargs):
    # simulate P&Ls over C,K-grid
    if CC is None or KK is None:
        CC = np.concatenate([np.arange(0.1,0.5,0.1),np.arange(0.5,3.5,0.5)])
        KK = np.arange(0.0,1.1,0.1)
    N = len(CC)*len(KK)
    pnls = dict()
    with tqdm(total=N, position=0, leave=True) as pbar:
        for C in CC:
            pnls[C] = dict()
            for K in KK:
                sims = backtest(ids, strat, C=C, K=K, **kwargs)
                pnls[C][K] = np.mean([sims[id]['pnl']['port'] for id in sims])
                pbar.update()
    pnls = pd.DataFrame(pnls)
```

```
    return pnls

def pnl_grid_over_WG(ids, strat, WW=None, GG=None, **kwargs):
    # simulate P&Ls over W,G-grid
    if WW is None or GG is None:
        WW = np.arange(1000,11000,1000)
        GG = np.arange(0.5,8.5,0.5)
    N = len(WW)*len(GG)
    pnls = dict()
    with tqdm(total=N, position=0, leave=True) as pbar:
        for W in WW:
            pnls[W] = dict()
            for G in GG:
                sims = backtest(ids, strat, W=W, G=G, **kwargs)
                pnls[W][G] = np.mean([sims[id]['pnl']['port'] for id in sims])
                pbar.update()
    pnls = pd.DataFrame(pnls)
    return pnls

def pnl_grid_over_P(ids, strat, param, PP, **kwargs):
    # simulate P&Ls over P-grid, where P is some user-specified param
    N = len(PP)
    pnls = dict()
    with tqdm(total=N, position=0, leave=True) as pbar:
        for P in PP:
            kwargs[param] = P
            sims = backtest(ids, strat, **kwargs)
            pnls[P] = np.mean([sims[id]['pnl']['port'] for id in sims])
            pbar.update()
    pnls = pd.Series(pnls)
    return pnls

def plot_3d_surf(x, y, df, xlabel=None, ylabel=None, zlabel=None, title=None, **kwargs)
    :
    X,Y = np.meshgrid(x,y)
    Z = df.to_numpy()
    D = pd.DataFrame(np.array([X,Y,Z]).reshape(3,-1).T,columns=['X','Y','Z'])

    fig = plt.figure(figsize=(10,6))
    ax = plt.axes(projection="3d")
    surf = ax.plot_trisurf(D['X'],D['Y'],D['Z'],cmap='copper')
    fig.subplots_adjust(left=0,right=0.9,bottom=0,top=1)
    if xlabel: ax.set_xlabel(xlabel)
    if ylabel: ax.set_ylabel(ylabel)
    if zlabel: ax.set_zlabel(zlabel)
    if title: ax.set_title(title)
    plt.show()
```

6 Statement of Contributions

All works in this project, including mathematical formulation, model development and report writing, are equally split between us. We thank Prof. Jia for his teaching and valuable insights over the semester.

References

[1] D. V. Budescu, E. Chen (2015). Identifying expertise to extract the wisdom of crowds. Management Science, 61(2), 267-280.

[2] J. Wolfers, E. Zitzewitz (2006). Interpreting prediction market prices as probabilities.