Unsupervised Learning in Financial Applications

CSCA 5632 Final

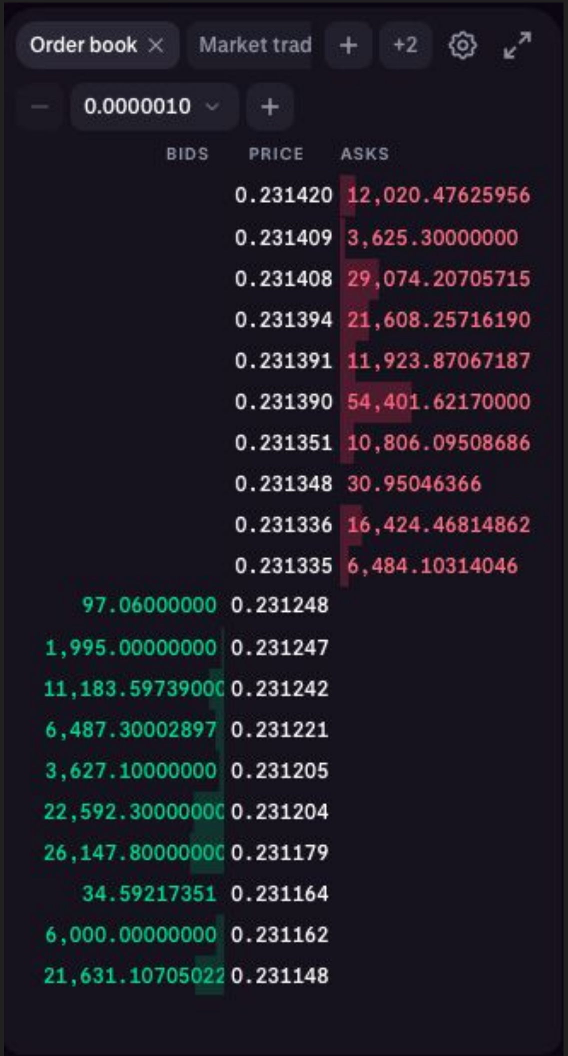
Problem Statement

This project investigates the application of Principal Component Analysis (PCA) to high-frequency Level 2 order book data of TRON (TRX) on the Kraken cryptocurrency exchange, with the objective of forecasting short-term price movements. The project aims to address two core questions: First, can PCA effectively reduce the dimensionality of TRX's order book data while preserving predictive signals related to price dynamics? Second, do PCA-derived features, when integrated into a simple learning framework (in this case Linear Regression), outperform baseline models that use raw order book features?

Background

- What is a Limit Order Book?
    - A limit order is an order to buy or sell a specific instrument at a specific price or better.
    - Two types of orders “Bid” and “Ask” (also sometimes called an offer)

- Bid says I will buy X amount at Y price or better.
- Ask says I will sell X amount at Y price or better.



Kraken Exchange API

- Order book data (often called a level 2 feed) that shows the quantity to buy or sell at different price levels is prohibitively expensive for traditional U.S. equities.
- Luckily cryptocurrency exchanges are significantly more open with their data often providing order book data via a public websocket API.
- For this project I utilized the Kraken V2 websocket book feed and a simple Go lang script to gather the necessary data

https://docs.kraken.com/api/docs/websocket-v2/book

API Background


```json
{
    "channel": "book",
    "type": "update",
    "data": [
        {
            "symbol": "MATIC/USD",
            "bids": [
                {
                    "price": 0.5657,
                    "qty": 1098.3947558
                }
            ],
            "asks": [],
            "checksum": 2114181697,
            "timestamp": "2023-10-06T17:35:55.440295Z"
        }
    ]
}
```

```json
{
    "channel": "book",
    "type": "snapshot",
    "data": [
        {
            "symbol": "MATIC/USD",
            "bids": [
                {
                    "price": 0.5666,
                    "qty": 4831.75496356
                },
                {
                    "price": 0.5665,
                    "qty": 6658.22734739
                },
                {
                    "price": 0.5664,
                    "qty": 18724.91513344
                },
                {
                    "price": 0.5663,
                    "qty": 11563.92544914
                },
                {
                    "price": 0.5662,
                    "qty": 14006.65365711
                },
                {
                    "price": 0.5661,
                    "qty": 17454.85679807
                },
                {
                    "price": 0.566,
                    "qty": 18097.1547
                },
                {
                    "price": 0.5659,
                    "qty": 33644.89175666
                },
                {
                    "price": 0.5658,
                    "qty": 148.3464
                },
                {
                    "price": 0.5657,
                    "qty": 606.70854372
                }
            ],
            "asks": [
                {
                    "price": 0.5668,
```

Go + JSONL

```go
func main() {
    // Kraken V2 WebSocket endpoint.
    wsURL := "wss://ws.kraken.com/v2"

    // Open a WebSocket connection.
    conn, _, err := websocket.DefaultDialer.Dial(wsURL, nil)
    if err != nil {
        log.Fatalf("Error connecting to Kraken WebSocket: %v", err)
    }
    defer conn.Close()

    // Open (or create) a file for appending JSON lines.
    file, err := os.OpenFile("orderbook.jsonl", os.O_APPEND|os.O_CREATE|os.O_WRONLY, (
    if err != nil {
        log.Fatalf("Error opening file: %v", err)
    }
    defer file.Close()

    // Create a buffered channel to store update messages.
    updatesChan := make(chan []byte, 100)

    // Start a goroutine to write messages from the channel to the file.
    go func() {
        for msg := range updatesChan {
            // Write the raw JSON message followed by a newline.
            if _, err := file.Write(append(msg, '\n')); err != nil {
                log.Printf("Error writing to file: %v", err)
            }
        }
    }()
```

```go
    // Build the subscription message for TRX/USD with a depth of 10.
    subReq := SubscribeRequest{Method: "subscribe"}
    subReq.Params.Channel = "book"
    subReq.Params.Symbol = []string{"TRX/USD"}
    subReq.Params.Depth = 10
    subReq.Params.Snapshot = true

    msg, err := json.Marshal(subReq)
    if err != nil {
        log.Fatalf("Error marshalling subscription request: %v", err)
    }

    // Send the subscription message.
    if err := conn.WriteMessage(websocket.TextMessage, msg); err != nil {
        log.Fatalf("Error sending subscription request: %v", err)
    }
    log.Printf("Subscription sent: %s", msg)

    // Optionally, set a read deadline.
    conn.SetReadDeadline(time.Now().Add(60 * 60 * time.Second))

    // Read messages from the WebSocket.
    for {
        _, message, err := conn.ReadMessage()
        if err != nil {
            log.Fatalf("Error reading message: %v", err)
        }
        log.Printf("Received message: %s", message)

        // Send the message to the buffered channel.
        updatesChan <- message
    }
}
```

JSONL

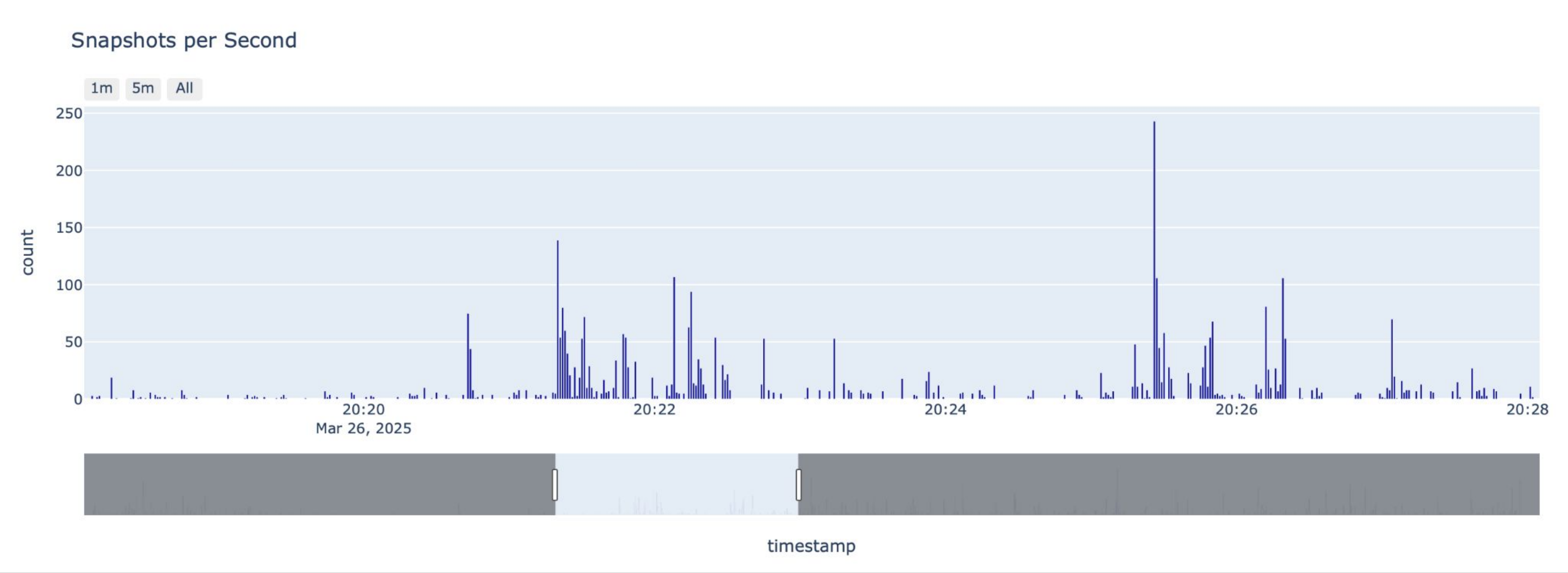

```
{"channel":"status","type":"update","data":[{"version":"2.0.9","system":"online","api_version":"v2","connection_id":6276481220026731114}]
{"method":"subscribe","result":{"channel":"book","depth":10,"snapshot":true,"symbol":"TRX/USD"},"success":true,"time_in":"2025-03-26T19:5
{"channel":"book","type":"snapshot","data":[{"symbol":"TRX/USD","bids":[{"price":0.229762,"qty":336.12031220},{"price":0.229761,"qty":652
{"channel":"heartbeat"}
{"channel":"book","type":"update","data":[{"symbol":"TRX/USD","bids":[],"asks":[{"price":0.229824,"qty":0.00000000},{"price":0.229891,"qt
{"channel":"book","type":"update","data":[{"symbol":"TRX/USD","bids":[],"asks":[{"price":0.229818,"qty":2331.08071764}],"checksum":290809
{"channel":"book","type":"update","data":[{"symbol":"TRX/USD","bids":[],"asks":[{"price":0.229818,"qty":1995.00000000},{"price":0.229817,
{"channel":"book","type":"update","data":[{"symbol":"TRX/USD","bids":[{"price":0.229761,"qty":0.00000000},{"price":0.229664,"qty":36.4136
{"channel":"book","type":"update","data":[{"symbol":"TRX/USD","bids":[],"asks":[{"price":0.229819,"qty":0.00000000},{"price":0.229891,"qt
{"channel":"book","type":"update","data":[{"symbol":"TRX/USD","bids":[{"price":0.229762,"qty":0.00000000},{"price":0.229585,"qty":45487.5
{"channel":"book","type":"update","data":[{"symbol":"TRX/USD","bids":[],"asks":[{"price":0.229817,"qty":6863.30464017}],"checksum":349531
{"channel":"book","type":"update","data":[{"symbol":"TRX/USD","bids":[{"price":0.229763,"qty":6528.46628917}],"asks":[],"checksum":388570
{"channel":"book","type":"update","data":[{"symbol":"TRX/USD","bids":[],"asks":[{"price":0.229817,"qty":6526.93229830},{"price":0.229816,
{"channel":"book","type":"update","data":[{"symbol":"TRX/USD","bids":[{"price":0.229756,"qty":0.00000000},{"price":0.229664,"qty":36.4136
{"channel":"heartbeat"}
{"channel":"heartbeat"}
{"channel":"heartbeat"}
{"channel":"heartbeat"}
{"channel":"heartbeat"}
{"channel":"heartbeat"}
{"channel":"heartbeat"}
{"channel":"heartbeat"}
{"channel":"heartbeat"}
{"channel":"book","type":"update","data":[{"symbol":"TRX/USD","bids":[{"price":0.229674,"qty":0.00000000},{"price":0.229664,"qty":36.4136
{"channel":"book","type":"update","data":[{"symbol":"TRX/USD","bids":[{"price":0.229680,"qty":24803.30000000}],"asks":[],"checksum":33661
{"channel":"book","type":"update","data":[{"symbol":"TRX/USD","bids":[],"asks":[{"price":0.229817,"qty":0.00000000},{"price":0.229891,"qt
{"channel":"book","type":"update","data":[{"symbol":"TRX/USD","bids":[{"price":0.229763,"qty":0.00000000},{"price":0.229664,"qty":36.4136
{"channel":"book","type":"update","data":[{"symbol":"TRX/USD","bids":[],"asks":[{"price":0.229816,"qty":0.00000000},{"price":0.229894,"qt
```

Convert Raw Updates to Snapshots

```python
with open('orderbook.jsonl', 'r') as f:
    for line in f:
        msg = json.loads(line)
        if msg.get("channel") == "book":
            msg_type = msg.get("type")
            data = msg.get("data", [])
            if msg_type == "snapshot":
                orderbook.process_snapshot(data)
            elif msg_type == "update":
                orderbook.process_update(data)

print(orderbook) # This is just showing the final state of the orderbook after all updates were processed to help understand the data structure
```

```
Bids:
[(0.230085, 29723.40346245), (0.230063, 333.43900823), (0.230062, 1995.0), (0.230059, 3242.5), (0.230058, 10866.79443695), (0.230036, 3477.0137), (0.23003, 7434.6736

Asks:
[(0.230086, 44645.40859149), (0.230135, 6517.92019009), (0.230153, 333.10527318), (0.230154, 6000.0), (0.230171, 3241.8), (0.230174, 10861.38298493), (0.230202, 1192
```

Visualize Update Frequency



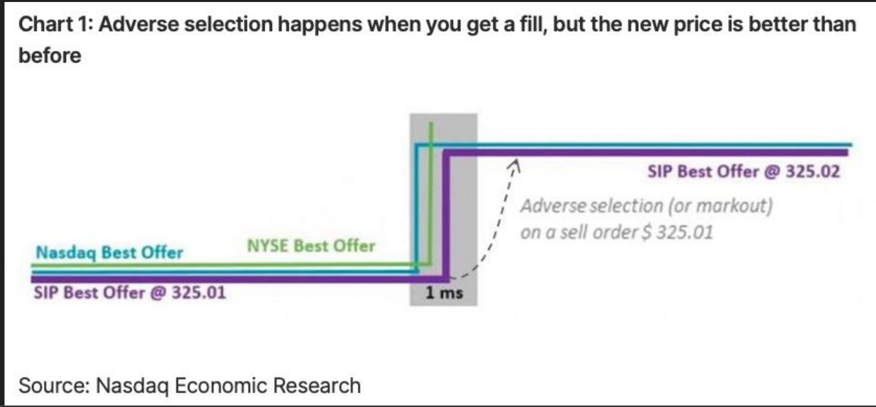
Convert to Dataframe for Analysis

	bids_0_price	bids_0_qty	bids_1_price	bids_1_qty	bids_2_price	bids_2_qty	bids_3_price	bids_3_qty	bids_4_price	bids_4_qty	...	asks_5_price
timestamp												
NaT	0.229762	336.120312	0.229761	6528.523117	0.229755	6528.693608	0.229742	3375.80000	0.229727	11183.597390	...	0.229834
2025-03-26 19:58:40.407519+00:00	0.229762	336.120312	0.229761	6528.523117	0.229755	6528.693608	0.229742	3375.80000	0.229727	11183.597390	...	0.229847
2025-03-26 19:58:40.407572+00:00	0.229762	336.120312	0.229761	6528.523117	0.229755	6528.693608	0.229742	3375.80000	0.229727	11183.597390	...	0.229847
2025-03-26 19:58:40.410248+00:00	0.229762	336.120312	0.229761	6528.523117	0.229755	6528.693608	0.229742	3375.80000	0.229727	11183.597390	...	0.229834
2025-03-26 19:58:40.411155+00:00	0.229762	336.120312	0.229755	6528.693608	0.229742	3375.800000	0.229727	11183.59739	0.229721	4352.244600	...	0.229834
...	...	...	...	...	...	...	...	...	...	...	...	...
2025-03-26 20:58:36.607108+00:00	0.230085	29723.403462	0.230063	333.439008	0.230062	1995.000000	0.230059	3242.50000	0.230058	10866.794437	...	0.230202
2025-03-26 20:58:36.629076+00:00	0.230085	29723.403462	0.230063	333.439008	0.230062	1995.000000	0.230059	3242.50000	0.230058	10866.794437	...	0.230178
2025-03-26 20:58:37.418444+00:00	0.230085	29723.403462	0.230063	333.439008	0.230062	1995.000000	0.230059	3242.50000	0.230058	10866.794437	...	0.230202
2025-03-26 20:58:37.418530+00:00	0.230085	29723.403462	0.230063	333.439008	0.230062	1995.000000	0.230059	3242.50000	0.230058	10866.794437	...	0.230174
2025-03-26 20:58:37.421857+00:00	0.230085	29723.403462	0.230063	333.439008	0.230062	1995.000000	0.230059	3242.50000	0.230058	10866.794437	...	0.230174

23214 rows × 40 columns

Understanding Markout

- A common metric when talking about price is “mid market” or just “mid.” This is the average price between the best bid and best ask.
- Markout usually described as price change after an event of interest




```
df['mid'] = (df['bids_0_price'] + df['asks_0_price']) / 2
df['ret_100'] = df['mid'].shift(-100) - df['mid']
df.dropna(inplace=True) # Need to drop last rows that don't have 100 entries after them
```

Model Features

- Skew represents the log difference between the quantity on the best bid and best ask
- Imbalance is similar to skew but instead of just looking at best bid and ask (often called top-of-book) we include all 10 levels on each side

```
df['skew'] = np.log(df.bids_0_qty) - np.log(df.asks_0_qty)
df['imbalance'] = np.log(df[list(df.filter(regex='bids_[0-9]_qty'))].sum(axis=1)) - \
    np.log(df[list(df.filter(regex='asks_[0-9]_qty'))].sum(axis=1))

# Pos skew means more top of book bids than asks
# Neg skew means more top of book asks than bids
# Pos imbalance means higher cumulative bid qty than ask qty
# Neg imbalance means higher cumulative ask qty than bid qty
```

Baseline Model Without PCA

[ ]

```python
# Create in sample and out of sample datasets
split = int(0.66 * len(df))
df_in = df.iloc[:split]
df_out = df.iloc[split:]
```

[ ]

```python
# As a simple first step check correlation. One thing to note is that order books are incredibly high noise
# environments so don't expect to see particularly high values like you might in other domains.
corr = df_in[['skew', 'imbalance', 'ret_100']].corr()
print(corr)
```

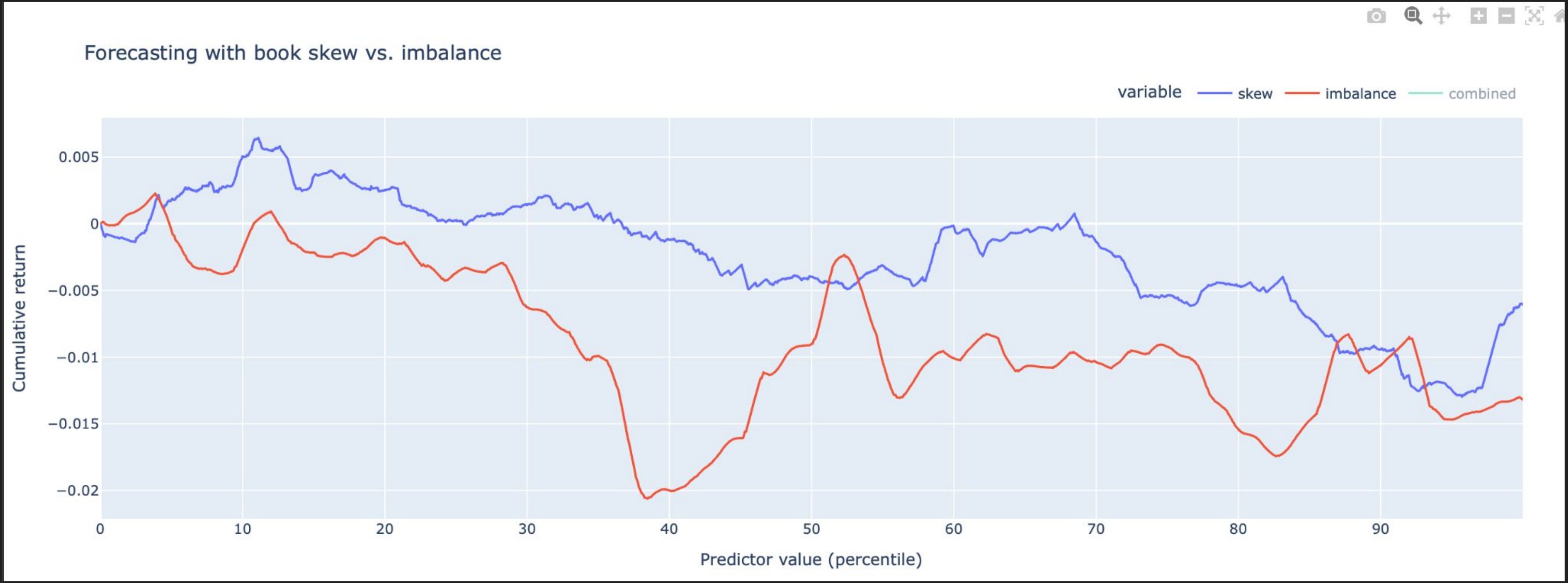
```
               skew  imbalance   ret_100
skew       1.000000   0.217988  0.119728
imbalance  0.217988   1.000000  0.058586
ret_100    0.119728   0.058586  1.000000
```

Now we can fit a linear regression model to each of the features individually and combinbed

[ ]

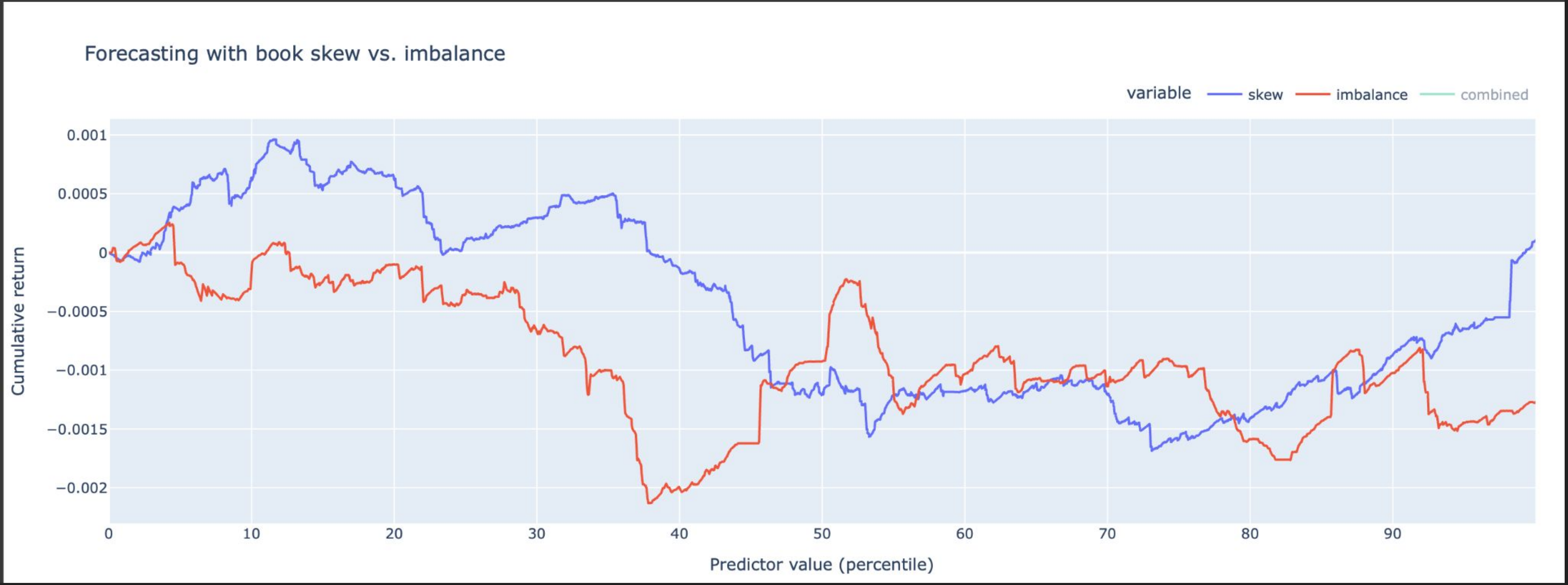
```python
reg = LinearRegression(fit_intercept=False, positive=True)

reg.fit(df_in[['skew']], df_in['ret_100'])
pred_skew = reg.predict(df_out[['skew']])

reg.fit(df_in[['imbalance']], df_in['ret_100'])
pred_imbalance = reg.predict(df_out[['imbalance']])

reg.fit(df_in[['skew', 'imbalance']], df_in['ret_100'])
pred_combined = reg.predict(df_out[['skew', 'imbalance']])
```

Model Results



Maybe the Markout was Too Far?

```
1 markout = 10
```



Unsupervised Learning to Improve Performance

- Similar preprocessing steps to reconstruct the snapshots and unpack the bids and asks into individual columns. Then standardize the features for Principal Component Analysis (PCA)

```python
# Define features for PCA
pca_feature_cols = []
for i in range(10):
    # Use log quantities to stabilize variance
    df[f'bids_{i}_log_qty'] = np.log(df[f'bids_{i}_qty'])
    df[f'asks_{i}_log_qty'] = np.log(df[f'asks_{i}_qty'])
    # Use price relative to mid
    df[f'bids_{i}_rel_price'] = (df[f'bids_{i}_price'] - df['mid'])
    df[f'asks_{i}_rel_price'] = (df[f'asks_{i}_price'] - df['mid'])
    pca_feature_cols.extend([
        f'bids_{i}_log_qty', f'asks_{i}_log_qty',
        f'bids_{i}_rel_price', f'asks_{i}_rel_price'
    ])
```

Apply PCA

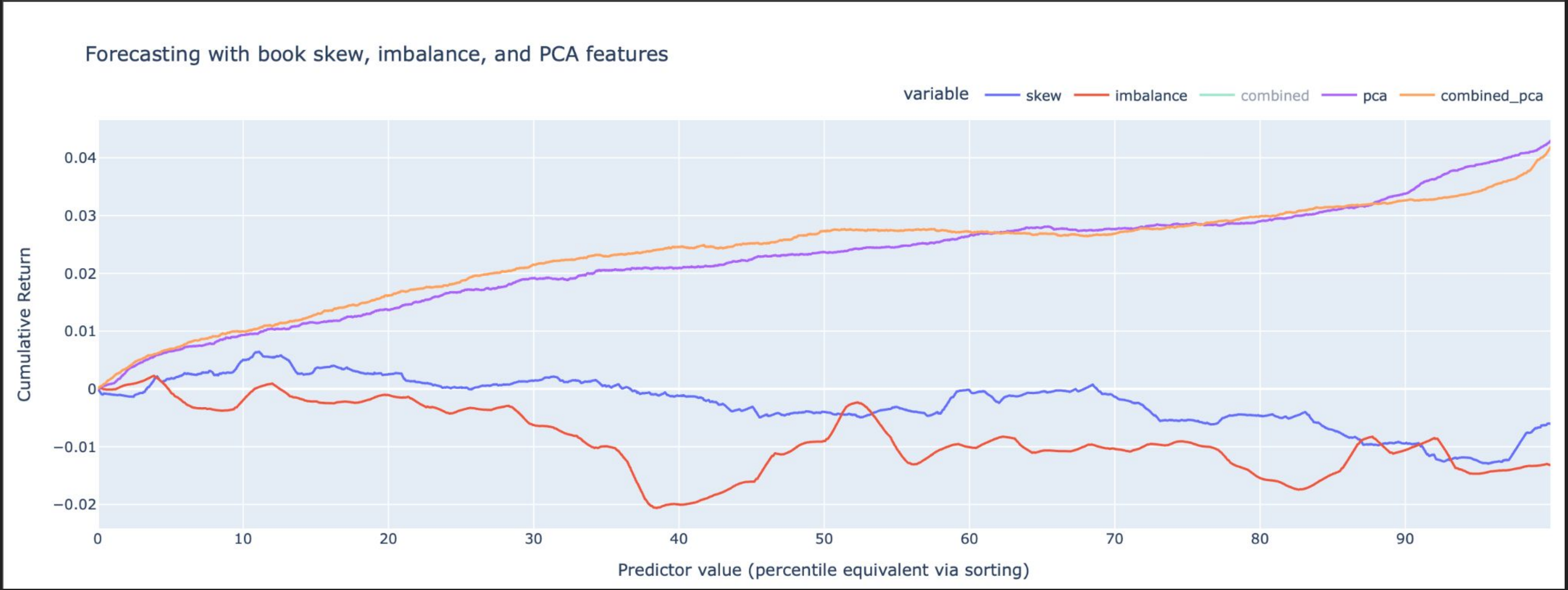
```python
# Unsupervised Step: PCA
# Standardize the selected features
scaler = StandardScaler()
df_in_pca_features_scaled = scaler.fit_transform(df_in[pca_feature_cols])
df_out_pca_features_scaled = scaler.transform(df_out[pca_feature_cols])

# Apply PCA
n_components = 5
pca = PCA(n_components=n_components)
pca_in = pca.fit_transform(df_in_pca_features_scaled)
pca_out = pca.transform(df_out_pca_features_scaled)

print(f"Explained variance ratio by {n_components} components: {pca.explained_variance_ratio_.sum():.3f}")
```

```
Explained variance ratio by 5 components: 0.546
```

Fit New Regression With PCA Columns

```python
# Add PCA components as features
pca_cols = [f'pca_{i}' for i in range(n_components)]
df_in[pca_cols] = pca_in
df_out[pca_cols] = pca_out

reg = LinearRegression(fit_intercept=False, positive=True)

# Original models
reg.fit(df_in[['skew']], df_in[f'ret_{markout}'])
pred_skew = reg.predict(df_out[['skew']])

reg.fit(df_in[['imbalance']], df_in[f'ret_{markout}'])
pred_imbalance = reg.predict(df_out[['imbalance']])

reg.fit(df_in[['skew', 'imbalance']], df_in[f'ret_{markout}'])
pred_combined = reg.predict(df_out[['skew', 'imbalance']])

# New model using PCA features
reg.fit(df_in[pca_cols], df_in[f'ret_{markout}'])
pred_pca = reg.predict(df_out[pca_cols])

# New model combining original features and PCA features
combined_pca_cols = ['skew', 'imbalance'] + pca_cols
reg.fit(df_in[combined_pca_cols], df_in[f'ret_{markout}'])
pred_combined_pca = reg.predict(df_out[combined_pca_cols])
```

Updated Results



Conclusion

- So should you go and trade on this signal...no I can't say I recommend it. This model is missing one key dynamic from real-world order books, time priority. In most markets, order books follow a price-time priority model. That is to say higher bids (lower asks) have priority over lower bids (higher asks) so for example a bid for $100 would get filled before a bid for $99. However, there is a second axis time that determines who gets filled within a given price level. So if you want to buy at $100 but another person already has an order in to buy at $100 you would be behind them in the que.
- This project did however show that PCA is an effective tool for improving models in high-dimensional environments like financial data

Future Work

- Expand data sources and collection period
    - Use different exchanges
    - Different asset classes
    - Longer day+ periods
- Additional parameter tuning