University of California Los Angeles

Master of Financial Engineering

AFP - Group 11

Improved Microprice Prediction using Detailed Order Book Data

Author:
Saransh Srivastava
Yash Verma
Alex Wu
Priyanshee Palriwala
Sai Kartheek

Faculty Supervisor:
Dr. Bernard Herskovic

December 10, 2022

Microprice Prediction

UCLA Anderson School of Management

Contents

1 Introduction **1**

2 Literature Review **2**

3 Data Description **6**

4 Results **7**

4.1 Stoikov's Micro-price estimator 7

4.2 Machine Learning Based Approach 12

4.3 Classifier Results 18

5 Conclusion and Future Extensions **19**

6 Appendix **20**

6.1 Data Import and Stoikov's Method Python codes 20

6.2 Random Forest Regressor + Classifier Python codes 27

6.3 XGBoost Regressor + Classifier Python codes 31

Microprice Prediction

1 Introduction

Trading Technologies is a financial technology company that develops algorithmic execution strategies and quantitative trading products for institution clients. They design and implement order execution strategies which helps their clients reduce their execution costs and slippage while doing trades in the market.

Trading algorithms incorporate market data to make informed trading decisions; however, there are many types of market data with fluctuating degrees of granularity. The most bird's eye image of the order book gives just a bid and ask price; however, there are numerous other pertinent bits of information in the Limit Order Book, like sizes at the bid and ask and the number of orders at those prices.

The straightforward way traders utilize this data for their algorithms is to compute a "microprice". The microprice can be considered a more precise cost than the simple mean of the bid and ask price. The microprice is commonly calculated as a weighted normal of the best bid and best ask prices, with the weights being the volumes at the best bid and ask. We, in our project, are trying to make a microprice predictor which at the very least, beats the weighted average mid price as a microprice

The goal of this project is to test out various microprice calculation methodologies that incorporate Level-2 and Level-3 order book data that exists in current research literature and see if they can predict future traded prices more accurately than the weighted mean of the best bid and ask price.(1)

2 Literature Review

The Micro-Price: A High Frequency Estimator of Future Prices (Sasha Stoikov)

The micro-price is defined to be the limit of a sequence of expected mid-prices and provide conditions for this limit to exist. The micro-price is a martingale by construction and can be considered to be the 'fair' price of an asset, conditional on the information in the order book. The micro-price may be expressed as an adjustment to the mid-price that takes into account the bid-ask spread and the imbalance. The micro-price can be estimated using high frequency data. It is shown empirically that it is a better predictor of short term prices than the mid-price or the weighted mid-price.

Definitions

The bid, ask, bid size, ask size: P^b, P^a, Q^b, Q^a

The mid-price:

$$M = \frac{P^b + P^a}{2}$$

The weighted mid-price:

$$M = P^b(1 - I) + P^a I$$

The generic imbalance:

$$I_t = \frac{Q^b}{Q^b + Q^a}$$

The exponentially weighted imbalance:

$$I_t = \frac{Q_{b1} * e^0 + Q_{b2} * e^{-1} + Q_{b3} * e^{-2}}{Q_{b1} * e^0 + Q_{b2} * e^{-1} + Q_{b3} * e^{-2} + Q_{a1} * e^0 + Q_{a2} * e^{-1} + Q_{a3} * e^{-2}}$$

where Q_{b1}, Q_{b2}, Q_{b3} are the bid size at best level, level 2, and level 3 of the order book respectively, and
Q_{a1}, Q_{a2}, Q_{a3} are the ask size at best level, level 2 and level of the order book respectively.
The bid-ask spread:

$$S = (P^a - P^b)$$

The mid vs. the weighted mid

The mid-price:

- Not a martingale (Bid-ask bounce)
- Medium frequency signal
- Doesn't use volume at the best bid and ask prices.

Microprice Prediction

The weighted mid-price:

- Uses the volume at the best bid and ask prices.
- High frequency signal
- Is quite noisy, particularly when the spread widens to two ticks

Desirable features of the Micro-Price

$P_t^{micro} = F(M_t, I_t, S_t) = M_t + G(I_t, S_t)$

- Martingale
- Computationally fast
- Better short term price predictions than the midprice or weighted midprice
- Should work for large tick stocks (like BAC) or small tick stocks (like CVX)

Micro-price definition

Define

$$P_t^{micro} = \lim_{n\to\infty} P_t^n$$

where the approximating sequence of martingale prices is given by

$$P_t^n = \mathbb{E}\left[M_{\tau_n} | I_t, S_t\right]$$

$\tau_1, ..., \tau_n$ are (random) times when the mid-price M_t changes

The micro-price is the expected mid-price in the distant future

In practice, the distant future is well captured by P_t^6, the expected mid price after 6 price moves.

Main result

The i-th approximation to the micro-price can be written as

$$P_t^n = M_t + \sum_{k=1}^{n} g^k(I_t, S_t)$$

where

$$g^1(I_t, S_t) = \mathbb{E}\left[M_{\tau_1} - M_t | I_t, S_t\right]$$

and

$$g^{n+1}(I_t, S_t) = \mathbb{E}\left[g^n(I_{\tau_1}, S_{\tau_1}) | I_t, S_t\right], \forall j \geq 0$$

can be computed recursively.

Finite state Markov chain

Microprice Prediction

- The imbalance takes discrete values $1 \leq i_I \leq n$,
- The spread takes discrete values $1 \leq i_S \leq m$
- The mid-price changes takes values in $K = [-0.01 \quad -0.005 \quad 0.005 \quad 0.01]$.
- Define the state $X_t = (I_t, S_t)$ with discrete values $1 \leq i \leq nm$

Computing g^1

The first step approximation to the micro-price

$$\begin{aligned} g^1(i) =& \mathbb{E}\left[M_{\tau_1} - M_t | X_t = i\right] \\ =& \left(1 - Q\right)^{-1} R^1 \underline{k} \end{aligned}$$

Where

$$Q_{ij} := \mathbb{P}(M_{t+1} - M_t = 0 \wedge X_{t+1} = j | X_t = i)$$

are the transition probabilities for transient states (mid price does not move)

$$R^1_{ik} := \mathbb{P}(M_{t+1} - M_t = k | X_t = i)$$

are the transition probabilities into absorbing states (mid price does move)
and $\underline{k} = [-0.01 \quad -0.005 \quad 0.005 \quad 0.01]^T$

Computing g^{i+1}

We can compute recursively

$$g^{n+1} = B g^n$$

where $B := \left(1 - Q\right)^{-1} R^2$
and R^2 is a new matrix of absorbing states

$$R^2_{ik} := \mathbb{P}(M_{t+1} - M_t \neq 0 \wedge I_{t+1} = k | I_t = i)$$

Estimation

1. On every quote, compute $I_t, S_t, (M_{t+1} - M_t)$, after having discretized the state space
2. Symmetrize the data, by making a copy where $I^2_t = n - I_t, S^2_t = S_t, (M^2_{t+1} - M^2_t) = -(M_{t+1} - M_t)$
3. Estimate transition probability matrices Q, R^1, R^2

Microprice Prediction

Computation

Compute the first micro-price adjustment:

$$p^1 - M = g^1 = \left(1 - Q\right)^{-1} R^1 \underline{k}$$

Use our recursive formula to compute the 6th micro-price adjustment:

$$p^6 - M = g^1 + g^2 + \ldots + g^6 = g^1 + Bg^1 + \ldots + B^5 g^1$$

In practice this converges after 6 price moves.

Summary

1. The micro-price is the expected mid-price in the distant future
2. In practice, the distant future is adequately approximated by τ^6 the time of the 6th mid price move
3. Paper provides evidence that the micro-price is a good predictor of future mid prices
4. Micro-price can fit very different microstructures
5. Micro-price is horizon independent
6. Micro-price seems to live between the bid and the ask

3 Data Description

- Order Book Data: This data is provided by Trading Technologies, and is sourced from OneTick, their market level order book data provider. This data will be requested through One Tick's API and will be imported as JSON files in Jupyter Notebook using python.
- Tick by Tick trade execution data, and order book updates provided by OneTick updated every trade.
- Data containing order details. Example: Order Sent/Order Received/Order Executed/Order Placed.
- There are a total of 58 parameters we can use for the microprice prediction. Some of them are: Level 1,2,3 Bid Ask Prices and Bid Ask Volume; Implied Volume; Trade Execution Details
- Previously, we were working on a month's worth of microsecond data/tick data for Interest Rate futures
- For further testing of the micro-price predictor, we have tick by tick data of US Treasury 2-year, 10-year Notes futures and options and SOFR futures. Following table depicts the tick by tick data we have tested on till now:

Bond	Code
SOFR Futures	SR1
US Treasury 10Y Futures	ZNU2
US Treasury 5Y Futures	ZFU2
US Treasury 2Y Futures	ZTU2
US Treasury 30Y Bond Futures	ZBU2

Table 1: Summary of Bond Futures and Options Data Used

4 Results

4.1 Stoikov's Micro-price estimator

We have replicated Stoikov's method of calculating the microprice on SOFR and US Treasury near Future data, and our findings are below :

Thus, our Microprice prediction will be:

- the spread(1-tick) adjustment to the mid-price if the long run average spread between bid and ask is around 1-tick, i.e., it is liquid
- it will be spread(2-tick) adjustment to the mid price if the long run average spread between bid and ask is around 2-ticks, i.e., it is less liquid

Further, we included all three levels of data in defining imbalance. We replicated the Stoikov's method for US Treasury options and have the following analysis for adjustments and stationary distribution of the weighted price:

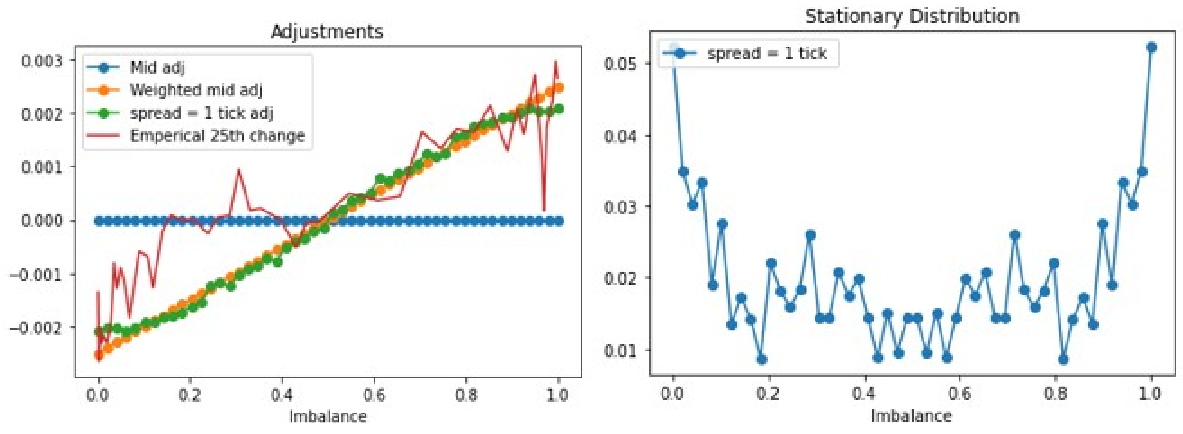

Figure 1: Observations for SR1 with Level 1 Imbalance

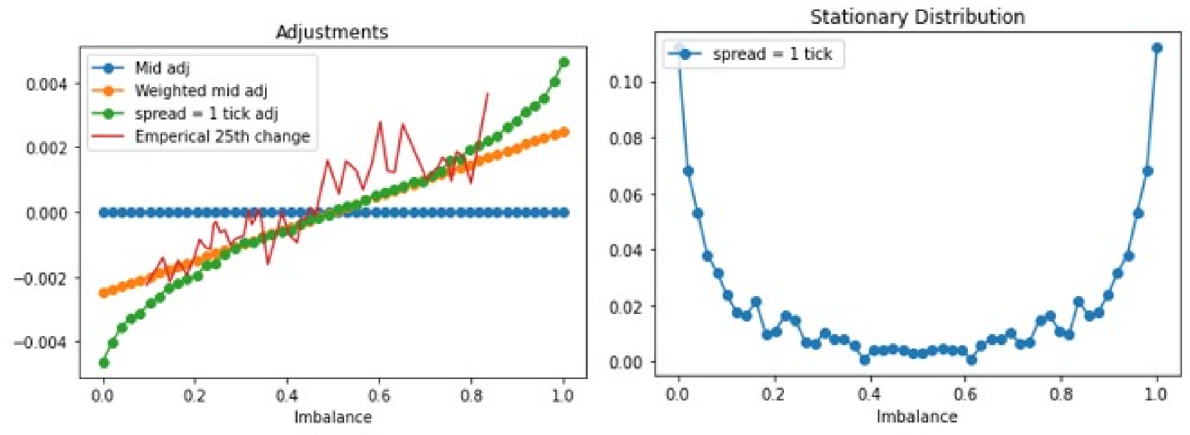

Figure 2: Observations for SR1 with Level 3 Imbalance

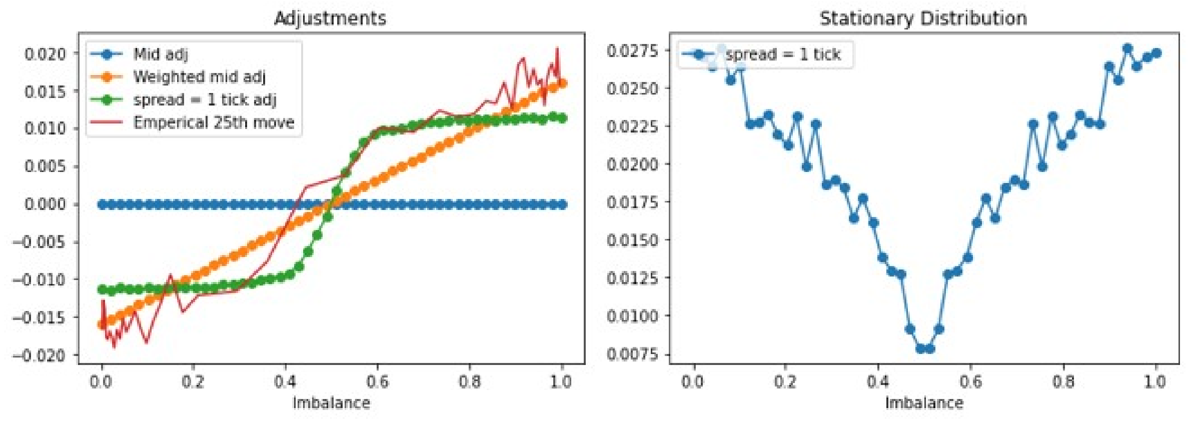

Figure 3: Observations for ZBU2 with Level 1 Imbalance

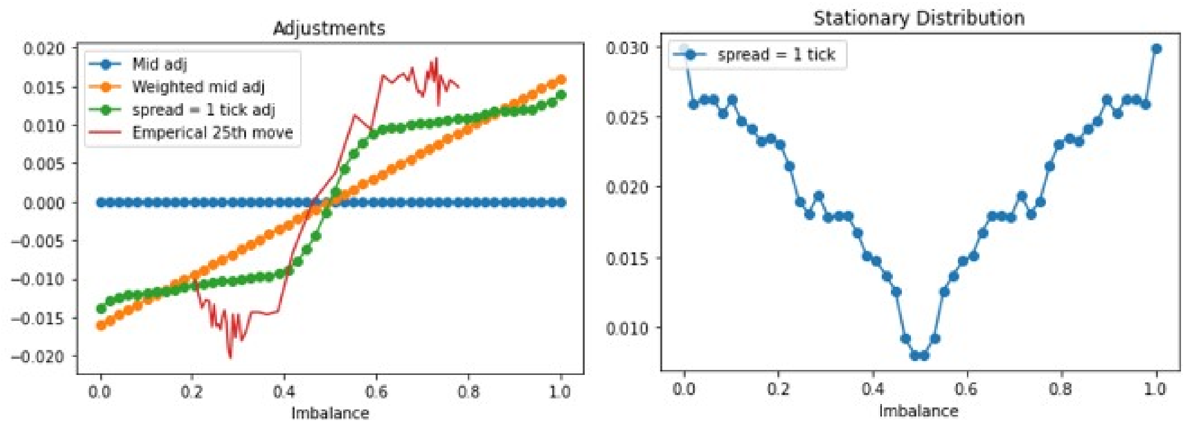

Figure 4: Observations for ZBU2 with Level 3 Imbalance

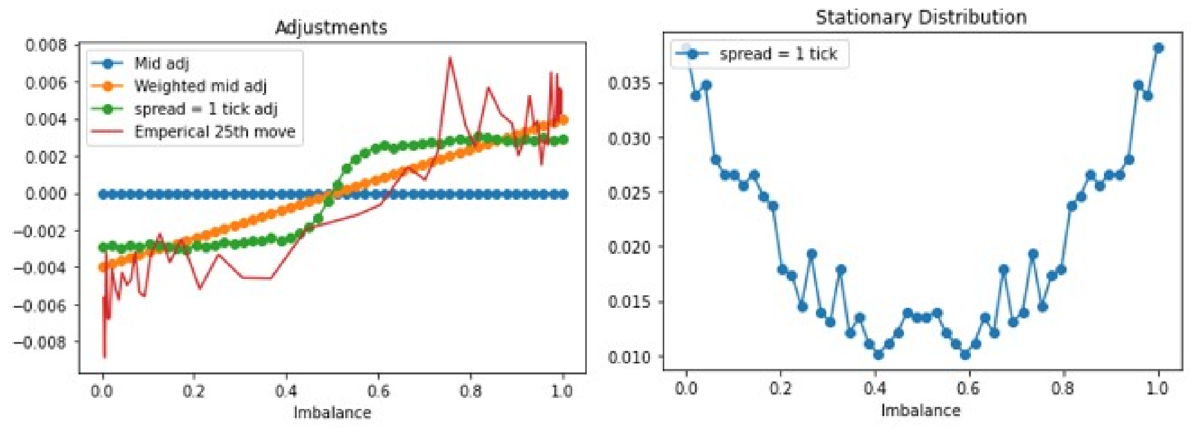

Figure 5: Observations for ZFU2 with Level 1 Imbalance

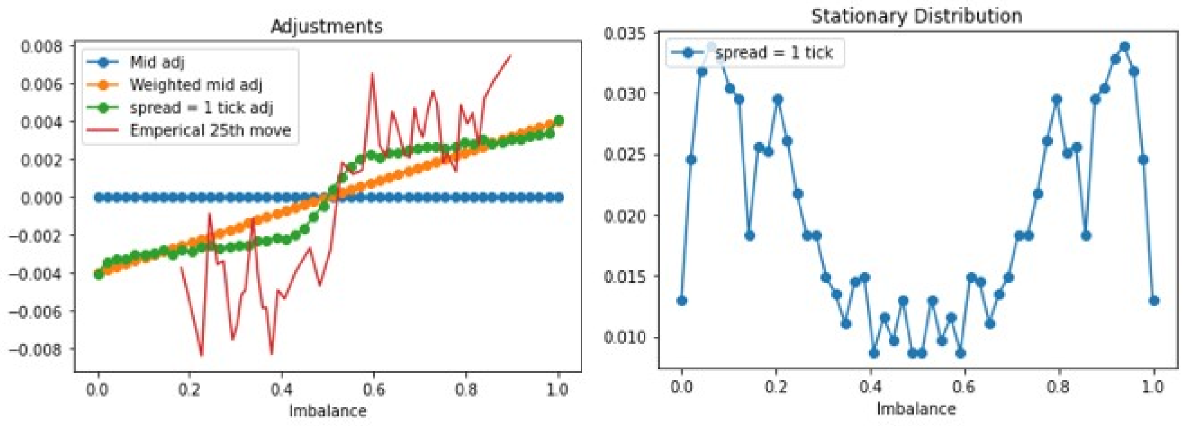

Figure 6: Observations for ZFU2 with Level 3 Imbalance

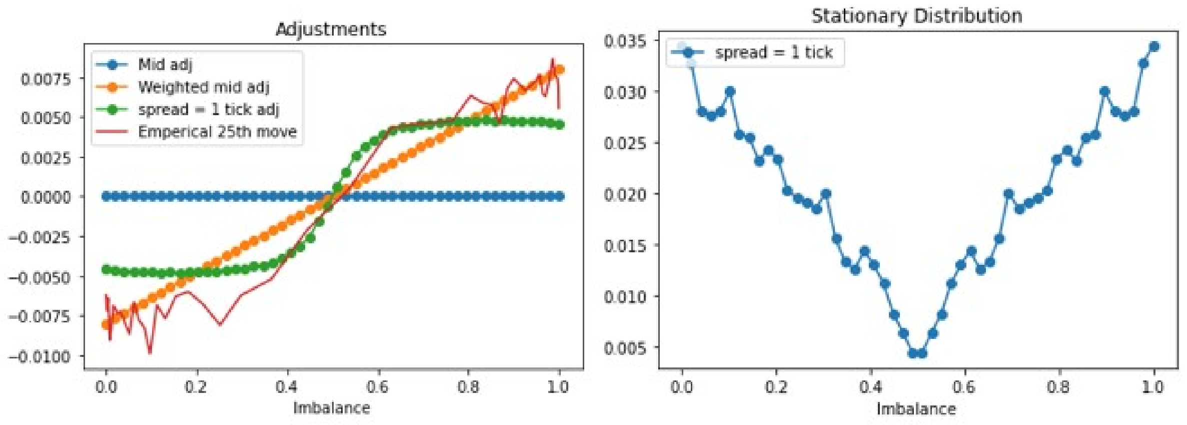

Figure 7: Observations for ZNU2 with Level 1 Imbalance

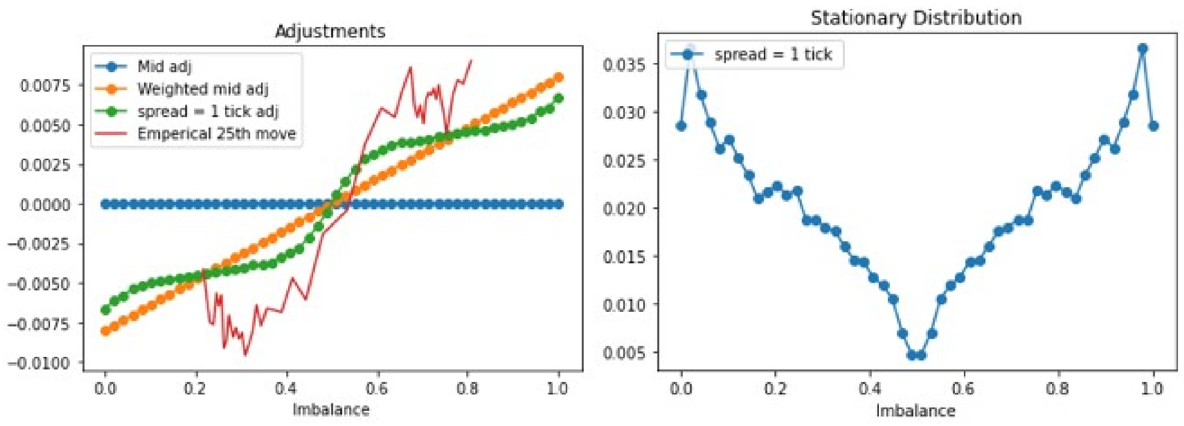

Figure 8: Observations for ZNU2 with Level 3 Imbalance

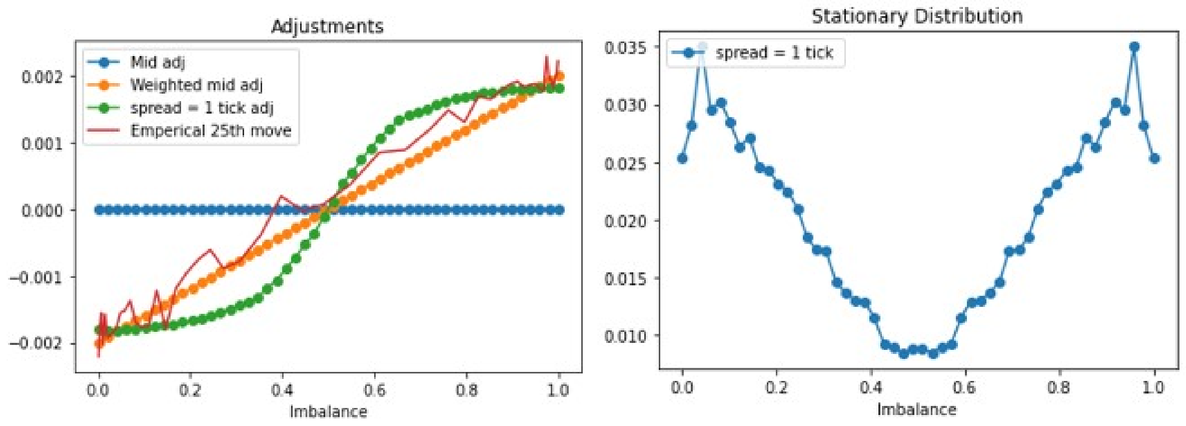

Figure 9: Observations for ZTU2 with Level 1 Imbalance

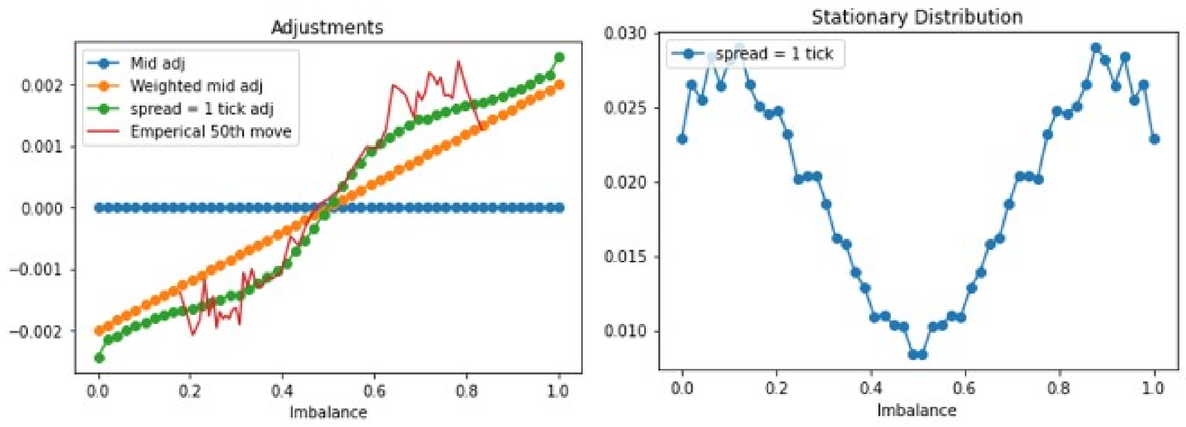

Figure 10: Observations for ZTU2 with Level 3 Imbalance

We can see from the above graph, the Stationary Distribution of the imbalances through our period of study and we can say by looking at the graph that it is stationary.

Looking at the graph above of Adjustments vs Imbalance, we can see that the adjustments fall roughly close to or outside VWAP and mid price, unlike the equity data that was used in Stoikov's paper. This difference has come up probably due to the fact that the microstructure of SOFR and UST futures are different vs that of equities, as the spread between bid and ask in our data is equal to the tick size almost all the time, while in the equities data the bid ask spread exist from 2 - 3 tick size depending on the security that is being studied.

4.2 Machine Learning Based Approach

Random Forest Model

First, We have utilised Random Forest for predicting price by exploiting Limit Order Book based features. Random forest applies bagging to decision trees in order to avoid over-fitting problems that occur with deep decision trees. Bagging techniques involve training multiple decision trees on subsets of the full data set and averaging the predictions in the case of regression trees. Additionally, to ensure low correlation amongst our multiple regression trees, each tree is assigned a random subset of possible features. Hyper-parameters for random forest predictors include number of trees, number of features, depth of the trees, etc.

In our model, response variable is the empirical change between the mid-price and the weighted mid price over the next 25 ticks. After carefully studying the Limit Order Book data, we experimented with various features to build the price prediction model.

Features used in the Random Forest model (base model) are:

- up to 3 levels of order book data (ask, bid, size, etc.)
- order imbalance
- volume weighted mid price
- bid ask spread

The residual plot (residuals vs predicted value) graph looks like this for this base model:

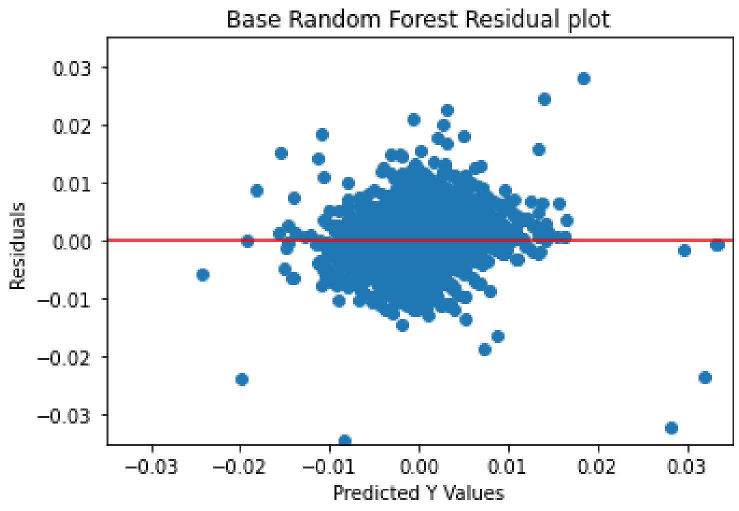

Figure 11: Base Random Forest Residual Plot

We further engineered more advanced time insensitive features to improvise the predictions (2). These include:

- Moneyness
- Order size and order fills
- Rolling mean of 20 periods for important base features
- Rolling skewness and kurtosis of 20 periods of important base features

The SHAP based feature importance method computes the contribution of a feature to the prediction. The more important the feature, the higher the absolute sum of SHAP values. Following is a SHAP summary plot that helps us understand the relative impact on model predictions between the important features:

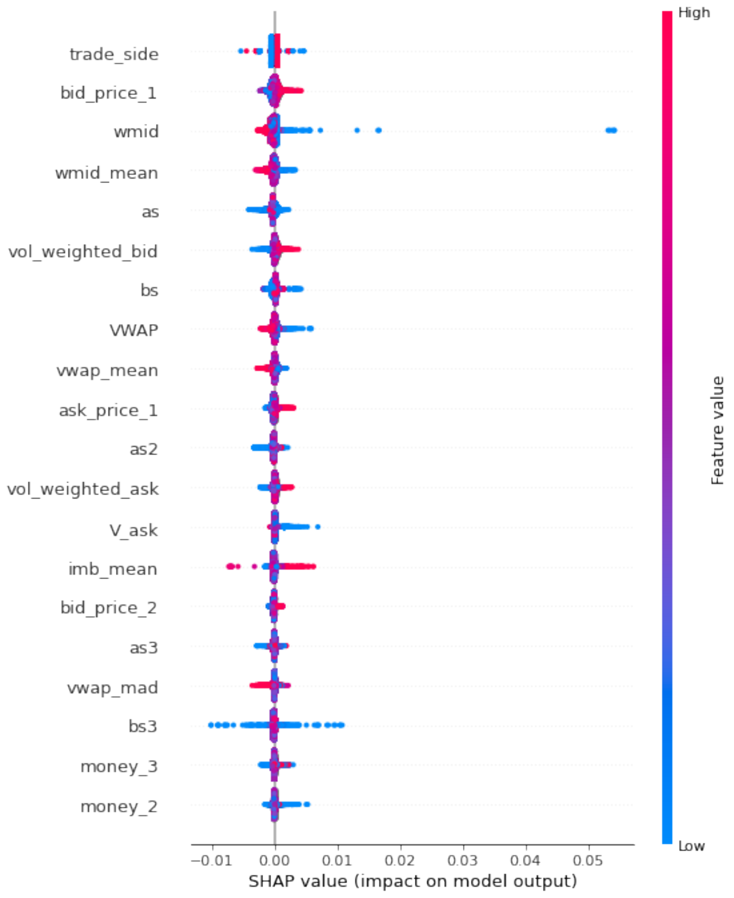

Figure 12: Feature Importance Plot

The residual plot (residuals vs predicted value) graph looks like this for the model with these advanced features:

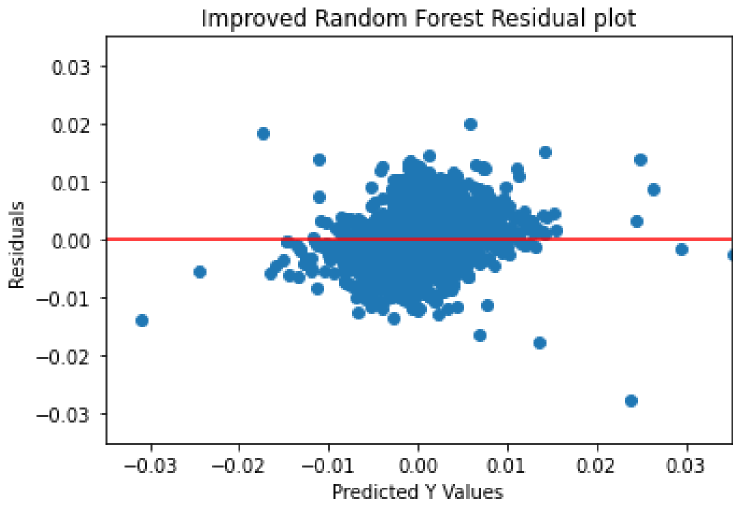

Figure 13: Improvised Random Forest Residual Plot

The entire data has been split into train and test samples in 70:30 ratio. The model has been trained using train sample, hyper-parameters are tuned using the performance on sub-sample of train data. Finally, the test sample is used to evaluate the models. The following table summarizes these two models:

	Base RF Model	Improvised RF Model
R^2	59.53%	62.63%
MAE	0.0017	0.0018
RMSE	0.0026	0.0025

Table 2: Comparison of Base Random Forest Model and Improvised Random Forest Model

We can clearly see that the improvised model is outperforming the base random forest model.

XGBoost Model

We have also utilised XGBoost model for predicting price by exploiting Limit Order Book based features. Boosting is quite different from bagging (as in Random Forest model). Bagging involved overfitting of each individual tree, but then averaging to get rid of noise in samples with low correlation. Boosting fits smaller trees and instead learns slowly by sequentially adding small trees fit to the prediction errors of the existing 'ensemble' of trees. Thus, each tree added depends on the trees already grown. It creates a strong predictor based on many weak predictors. Hyper-parameters for xgboost predictors include number of trees, shrinkage parameter, depth of the trees, etc.

In our model, response variable is the empirical change between the mid-price and the weighted mid price over the next 25 ticks. After carefully studying the Limit Order Book data, we experimented with various features to build the price prediction model.

Features used in the XGBoost model (base model) are:

- up to 3 levels of order book data (ask, bid, size, etc.)
- order imbalance
- volume weighted mid price
- bid ask spread

The residual plot (residuals vs predicted value) graph looks like this for this XGBoost base model:

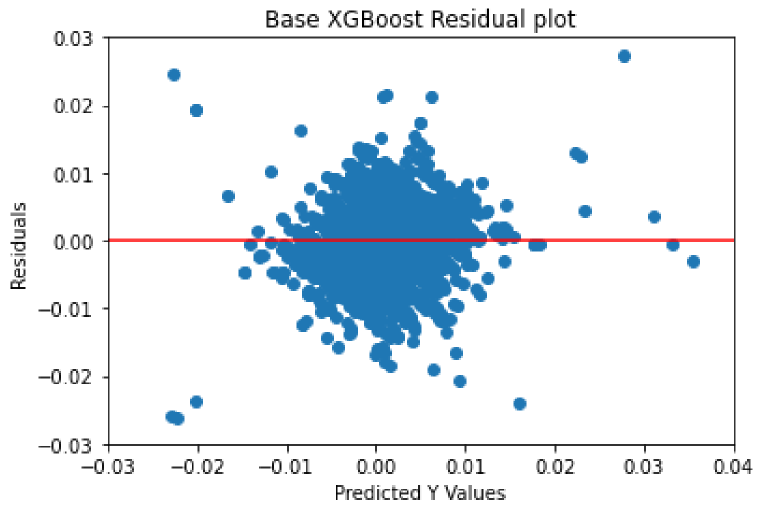

Figure 14: Base XGBoost Residual Plot

We further engineered more advanced time insensitive features to improvise the predictions (2). These include:

- Moneyness
- Order size and order fills
- Rolling mean of 20 periods for important base features
- Rolling skewness and kurtosis of 20 periods of important base features

The SHAP based feature importance method computes the contribution of a feature to the prediction. The more important the feature, the higher the absolute sum of SHAP values. Following is a SHAP summary plot that helps us understand the relative impact on model predictions between the important features:

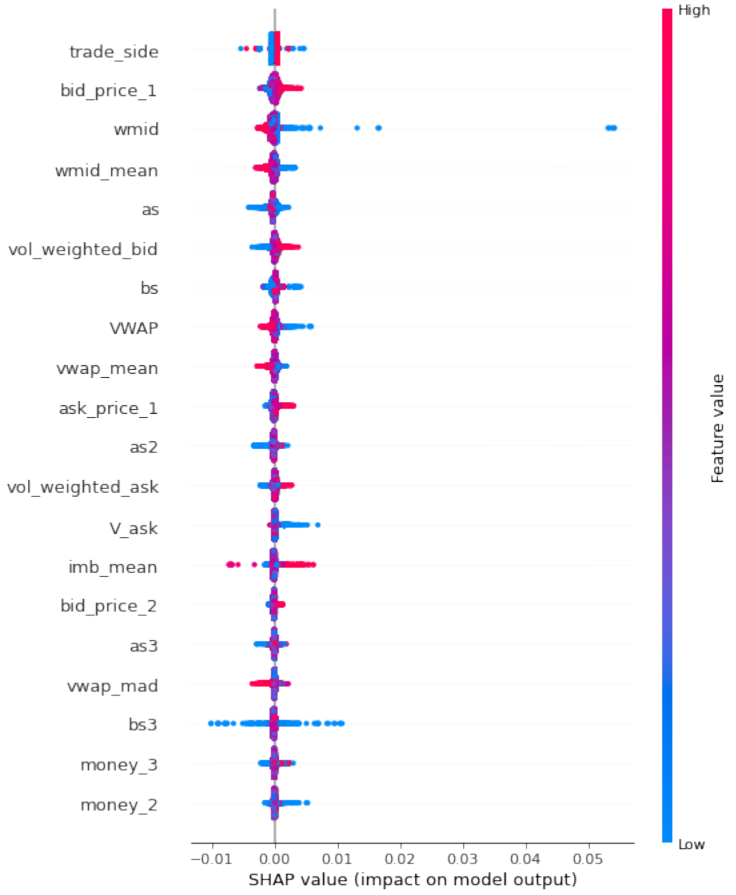

Figure 15: Shap Graph - Feature Importance

The residual plot (residuals vs predicted value) graph looks like this for the XGBoost model with these advanced features:

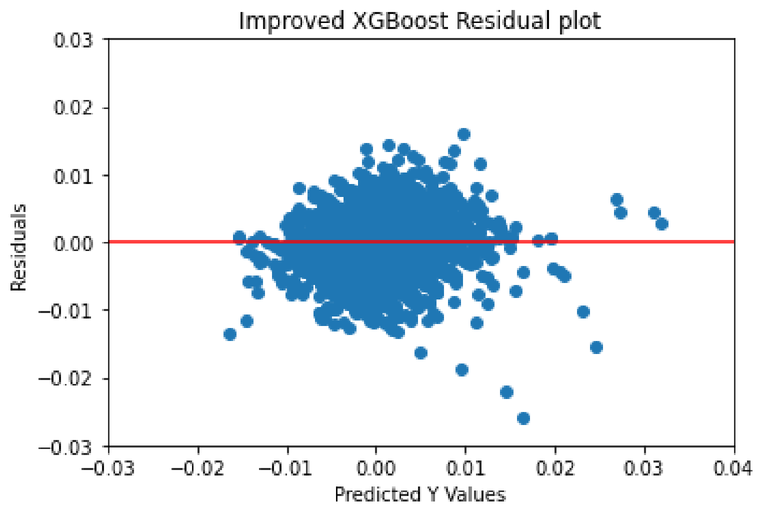

Figure 16: Improvised XGBoost Residual Plot

The entire data has been split into train and test samples in 70:30 ratio. The model has been trained using train sample, hyper-parameters are tuned using the performance on sub-sample of train data. Finally, the test sample is used to evaluate the models. The following table summarizes these two models:

	Base XGB Model	Improvised XGB Model
R^2	50.3%	60.5%
MAE	0.0021	0.0019
RMSE	0.0029	0.0025

Table 3: Comparison of Base XGBoost Model and Improvised XGBoost Model

We can clearly see that the improvised model is outperforming the base XGBoost model.

4.3 Classifier Results

After achieving a good R^2 when predicting mid price using the Machine Learning Approach, we further got intrigued by the predictive power of these features in identifying the direction of price change. Trading technologies often assist their clients with estimating the direction of price change and they wanted to assess whether the Machine Learning model could prove helpful. So, we built a model similar to the Machine Learning models mentioned in previous section. The response variable, however, in this case, is the direction of the mid-price change in future 25 ticks. We utilised the base as well as advanced features mentioned previously and trained both Random Forest and XGBoost models. We evaluated the model on the test sample and following are the results:

	XGBoost	Random Forest
Accuracy	82.78%	86.96%
AUC	0.88	0.94

Table 4: Results of the Classifier models

We can see that the Random Forest model is outperforming the XGBoost model. Following figure shows the ROC (receiver operating characteristic) curve for the Random Forest model:

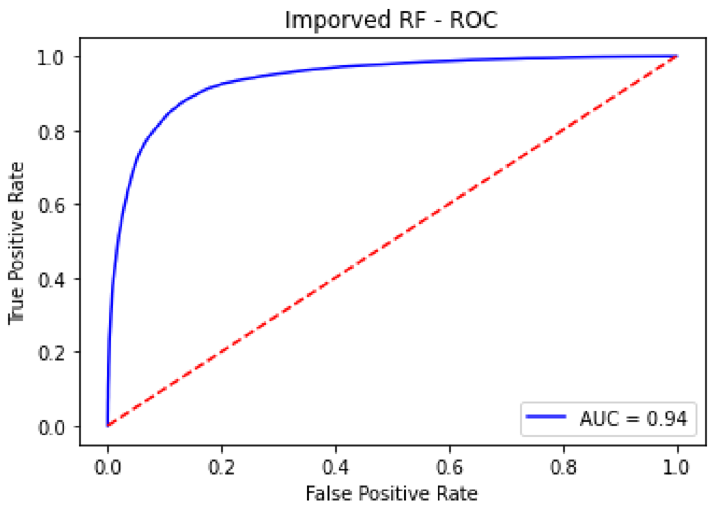

Figure 17: Classifier (RF) ROC Curve

5 Conclusion and Future Extensions

Conclusions

We have predicted the mid price adjustment using both the Markov-model based micro-price estimator as well as machine learning approach. Amongst the four machine learning regression models, we saw improvised Random Forest Model performs the best in terms of R^2. The most important features predicting price are: trade side, bid price, weighted mid, rolling mean of weighted mid and volume. We also compared both the methods and checked the accuracy of predictions using root mean squared error (RMSE). The micro-price estimator based on Stoikov's paper has a RMSE of 0.0028, the Improvised Random Forest model has RMSE of 0.0025 and the improvised XGBoost model has RMSE of 0.0025. Even though the RMSE of Stoikov's method is a little bit on the higher side, we still believe it is a better approach to estimating micro-price adjustments based on our experience with the project. This is because Stoikov's method is more intuitive, can be justified and overall makes more sense. On the other hand, machine learning methods are still a black-box.

Future extensions

Possible future extensions can be:

1. Relax the Finite-Chain Markov Assumption in the modelling of transition states
2. Improvise the machine learning model utilising more advanced features from the LOB data
3. Implement iceberg detection to get a better idea about incoming orders

References

[1] S. Stoikov, "The micro-price: A high frequency estimator of future prices," 2018.

[2] F. Qureshi, "Investigating limit order book features for short-term price prediction: A machine learning approach," 2018.

6 Appendix

6.1 Data Import and Stoikov's Method Python codes

```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import os
from matplotlib import animation

from scipy.linalg import block_diag
import matplotlib.pyplot as plt

pd.set_option("display.max_columns", None)

 ticksize =0.004

def read_data():

    # read data
    data_directory = './mbo_data/'
     df_list  = []

    directory = os.fsencode(data_directory)
    for file in os. listdir (directory):
        filename = os.fsdecode(file)
        if filename.startswith('ZTU2') & filename.endswith('.csv'):
            temp = pd.read_csv(data_directory + filename, header=0, quotechar='\'',
                on_bad_lines='skip',engine='python')
            temp['instrument'] = filename. split ('.') [0]
            temp['product_group'] = filename[0:2]
            temp['front_month'] = filename[2:4]
             df_list .append(temp)

    df = pd.concat( df_list , axis=0).reset_index()

    # handle time information

    df. loc[df.timestamp.str.len() < 21, 'timestamp'] = df.loc[df.timestamp.str.len() <
        21, 'timestamp'] + '.0'   # pad timestamps that do not contain fractional seconds
    df['time'] = pd.to_datetime(df['timestamp'], format='%Y-%b-%d %H:%M:%S.%f',
        utc=True).dt.tz_convert('America/Chicago')
    df['time_of_day'] = df.time.dt.time
    df['trade_date'] = (df.time + pd.Timedelta(hours=7, minutes=30)).dt.strftime('%Y%
        m%d')
    df['hour'] = (df.time + pd.Timedelta(hours=7)).dt.hour # map hour of day to hours
        since the start of the trading day
    df['weeks_until_expiration'] = (df.days_until_expiration/7).astype('int')

    df['next_tick'] = 0  # 1 means quote move was up for buys/down for sells or down
        for buys/up for sells
    df.loc[df.same_time_until_next_quote_move_through_ms < 0, 'next_tick'] = 1
    df.loc[df.same_time_until_next_quote_move_through_ms >= 0, 'next_tick'] = -1

    return df
df = read_data()

def get_ask(y) :
    if y.trade_side == -1 :
        return y.price_contra_1
    else :
        return y.price_same_1

def get_ask_2(y) :
    if y.trade_side == -1 :
        return y.price_contra_2
    else :
        return y.price_same_2

def get_ask_3(y) :
    if y.trade_side == -1 :
        return y.price_contra_3
    else :
        return y.price_same_3

def get_bid(y) :
    if y.trade_side == 1 :
        return y.price_contra_1
    else :
        return y.price_same_1

def get_bid_2(y) :
    if y.trade_side == 1 :
        return y.price_contra_2
    else :
        return y.price_same_2

def get_bid_3(y) :
    if y.trade_side == 1 :
        return y.price_contra_3
    else :
        return y.price_same_3

def get_ask_size(y) :
    if y.trade_side == -1 :
        return y.size_contra_1
    else :
        return y.size_same_1

def get_ask_size_2(y) :
    if y.trade_side == -1 :
        return y.size_contra_2
    else :
        return y.size_same_2

def get_ask_size_3(y) :
    if y.trade_side == -1 :
        return y.size_contra_3
    else :
        return y.size_same_3

def get_bid_size(y) :
    if y.trade_side == 1 :
        return y.size_contra_1
    else :
        return y.size_same_1

def get_bid_size_2(y) :
    if y.trade_side == 1 :
        return y.size_contra_2
    else :
        return y.size_same_2

def get_bid_size_3(y) :
    if y.trade_side == 1 :
        return y.size_contra_3
    else :
        return y.size_same_3

df1 = df
df1 = df1[["timestamp","trade_side","price_same_1","price_same_2","price_same_3","
    price_contra_1","price_contra_2","price_contra_3","direct_size_same_1","
    direct_size_same_2","direct_size_same_3","implied_size_same_1","implied_size_same_2",
    "implied_size_same_3"," direct_size_contra_1 "," direct_size_contra_2 ","
    direct_size_contra_3 "," implied_size_contra_1"," implied_size_contra_2","
    implied_size_contra_3","time"," fills " ]]

df1["size_same_1"] = df1["direct_size_same_1"] + df1["implied_size_same_1"]
df1["size_same_2"] = df1["direct_size_same_2"] + df1["implied_size_same_2"]
df1["size_same_3"] = df1["direct_size_same_3"] + df1["implied_size_same_3"]

df1[" size_contra_1"] = df1[" direct_size_contra_1 "] + df1[" implied_size_contra_1"]
df1[" size_contra_2"] = df1[" direct_size_contra_2 "] + df1[" implied_size_contra_2"]
df1[" size_contra_3"] = df1[" direct_size_contra_3 "] + df1[" implied_size_contra_3"]

df1["VWAP"] = (df1["price_same_1"]*df1["size_contra_1"] + df1["price_same_2"] * df1["
    size_contra_2"] + df1["price_same_3"]
                * df1[" size_contra_3"] + df1["price_contra_1"] * df1["size_same_1"] + df1["
                    price_contra_2"] *
                df1["size_same_2"] + df1["price_contra_3"] * df1["size_same_3"])/(df1["
                    size_same_1"]+
                df1["size_same_2"] +df1["size_same_3"]+ df1[" size_contra_1"]+df1["
                    size_contra_2"]+df1[" size_contra_3"])

df1['ask'] = df1.apply(lambda x : get_ask(x),axis=1)

df1['bid'] = df1.apply(lambda x : get_bid(x),axis=1)

df1['bs'] = df1.apply(lambda x : get_bid_size(x),axis=1)
df1['bs2'] = df1.apply(lambda x : get_bid_size_2(x),axis=1)
df1['bs3'] = df1.apply(lambda x : get_bid_size_3(x),axis=1)

df1 = df1[df1['bs'] + df1['bs2'] + df1['bs3'] > 100]

df1['as'] = df1.apply(lambda x : get_ask_size(x),axis=1)
df1['as2'] = df1.apply(lambda x : get_ask_size_2(x),axis=1)
```

```
df1['as3'] = df1.apply(lambda x : get_ask_size_3(x),axis=1)

df1 = df1[df1['as'] + df1['as2'] + df1['as3'] > 100]
df1['mid']=(df1['bid'].astype(float)+df1['ask'].astype(float))/2

df1['V_bid'] = df1['bs'].astype(float) + df1['bs2'].astype(float) * np.exp(-0.5) + df1['
    bs3'].astype(float) * np.exp(-1)
df1['V_ask'] = df1['as'].astype(float) + df1['as2'].astype(float) * np.exp(-0.5) + df1['
    as3'].astype(float) * np.exp(-1)

# df1['imb']=(df1['bs'].astype(float))/(df1['bs'].astype(float)+df1['as'].astype(float))
df1['imb']=(df1['V_bid'])/(df1['V_bid']+df1['V_ask'])

df1['wmid']=df1['ask'].astype(float)*df1['imb']+df1['bid'].astype(float)*(1-df1['imb'])
df1['spread'] = df1['ask']- df1['bid']

df1["timestamp"] = pd.to_datetime(df1["timestamp"])
df2 = df1[(df1["timestamp"].dt.hour >= 10) & (df1["timestamp"].dt.hour <= 19)]

def prep_data_sym(T,n_imb,dt,n_spread):

    # adds the spread and mid prices
    T['spread']=np.round((T['ask']-T['bid'])/ticksize)*ticksize
    T['mid']=(T['bid']+T['ask'])/2

    #filter out spreads >= n_spread
    T = T.loc[(T.spread == n_spread*ticksize) & (T.spread>0)]
    T['imb']=T['bs'] /(T['bs'] + T['as'])

#    T['imb']=(T['bs']*np.exp(0) + T['bs2'] * np.exp(-1) + T['bs3']* np.exp(-2))/((T['
    bs']*np.exp(0) + T['bs2']* np.exp(-1) + T['bs3']* np.exp(-2) + T['as']*np.exp(0) +
    T['as2']* np.exp(-1) + T['as3']* np.exp(-2)))
    #discretize imbalance into percentiles
    T['imb_bucket'] = pd.qcut(T['imb'], n_imb, labels=False, duplicates='drop')
    T['next_mid']=T['mid'].shift(-dt)
    #step ahead state variables
    T['next_spread']=T['spread'].shift(-dt)
    T['next_time']=T['time'].shift(-dt)
    T['next_imb_bucket']=T['imb_bucket'].shift(-dt)
    # step ahead change in price
    T['dM']=np.round((T['next_mid']-T['mid'])/ticksize*4)*ticksize/4
    T = T.loc[(abs(T.dM) == ticksize) | (T.dM == 0)]
    print(np.unique(T['dM'], return_counts=True))
    # symetrize data
    T2 = T.copy(deep=True)
    T2['imb_bucket']=n_imb-1-T2['imb_bucket']
    T2['next_imb_bucket']=n_imb-1-T2['next_imb_bucket']
    T2['dM']=-T2['dM']
    T2['mid']=-T2['mid']
    T3=pd.concat([T,T2])
    T3.index = pd.RangeIndex(len(T3.index))
    return T3,ticksize

def estimate(T):
    no_move=T[T['dM']==0]
    no_move_counts=no_move.pivot_table(index=[ 'next_imb_bucket'],
                                 columns=['spread', 'imb_bucket'],
                                 values='time',
                                  fill_value =0,
                                 aggfunc='count').unstack()
    Q_counts=np.resize(np.array(no_move_counts[0:(n_imb*n_imb)]),(n_imb,n_imb))
    # loop over all spreads and add block matrices
    for i in range(1,n_spread):
        Qi=np.resize(np.array(no_move_counts[(i*n_imb*n_imb):(i+1)*(n_imb*n_imb)]),(
            n_imb,n_imb))
        Q_counts=block_diag(Q_counts,Qi)
    #print Q_counts
    move_counts=T[(T['dM']!=0)].pivot_table(index=['dM'],
                                     columns=['spread', 'imb_bucket'],
                                     values='time',
                                      fill_value =0,
                                     aggfunc='count').unstack()
    R_counts=np.resize(np.array(move_counts),(n_imb*n_spread,2))
    T1=np.concatenate((Q_counts,R_counts),axis=1).astype(float)
    for i in range(0,n_imb*n_spread):
        T1[i]=T1[i]/T1[i].sum()
    Q=T1[:,0:(n_imb*n_spread)]
    R1=T1[:,(n_imb*n_spread):]

    K=np.array([-ticksize, ticksize ])
    move_counts=T[(T['dM']!=0)].pivot_table(index=['spread','imb_bucket'],
                                 columns=['next_spread', 'next_imb_bucket'],
                                 values='time',
                                  fill_value =0,
                        aggfunc='count') #.unstack()

    R2_counts=np.resize(np.array(move_counts),(n_imb*n_spread,n_imb*n_spread))
    T2=np.concatenate((Q_counts,R2_counts),axis=1).astype(float)

    for i in range(0,n_imb*n_spread):
        T2[i]=T2[i]/T2[i].sum()
    R2=T2[:,(n_imb*n_spread):]
    Q2=T2[:,0:(n_imb*n_spread)]
    G1=np.dot(np.dot(np.linalg.inv(np.eye(n_imb*n_spread)-Q),R1),K)
    B=np.dot(np.linalg.inv(np.eye(n_imb*n_spread)-Q),R2)


    return G1,B,Q,Q2,R1,R2,K

def plot_Gstar(G1,B,T):
    G2=np.dot(B,G1)+G1
    G3=G2+np.dot(np.dot(B,B),G1)
    G4=G3+np.dot(np.dot(np.dot(B,B),B),G1)
    G5=G4+np.dot(np.dot(np.dot(np.dot(B,B),B),B),G1)
    G6=G5+np.dot(np.dot(np.dot(np.dot(np.dot(B,B),B),B),B),G1)
    plt.plot(imb,np.linspace(-0.005,0.005,n_imb)*0,label='Mid adj',marker='o')
    plt.plot(imb,np.linspace(-ticksize/2, ticksize /2,n_imb),label='Weighted mid adj',
        marker='o')
    for i in range(0,n_spread):
        plt.plot(imb,G6[(0+i*n_imb):(n_imb+i*n_imb)],label="spread = "+str(i+1)+"
            tick adj",marker='o')
#    plt.ylim(-0.01,0.01)
    plt.legend(loc='upper left')
    plt.title('Adjustments')
    plt.xlabel('Imbalance')
    return G6

n_imb=50
n_spread=1
dt=1
pd.set_option('mode.chained_assignment', None)
import warnings
warnings.simplefilter(action='ignore', category=UserWarning)
T, ticksize =prep_data_sym(df2,n_imb,dt,n_spread)

T.sort_values(by=['timestamp'],inplace=True)
```

```
D = T.copy().tail(210444 −175000)
D["MidMove"] = np.abs(D["mid"]).shift(−50) − np.abs(D["mid"])
tst  = D[["imb","MidMove"]]

imb=np.linspace(0,1,n_imb)
G1,B,Q,Q2,R1,R2,K=estimate(T.head(175000 ))
G6=plot_Gstar(G1,B,T)
tst ['imb_bucket'] = pd.qcut(tst['imb'],  n_imb, labels=False, duplicates='drop')
tst ['adjustment'] = G6[tst['imb_bucket']]
tst ['error']  = np.abs(tst['adjustment'] − tst["MidMove"])
k = tst.groupby(['imb_bucket']).mean()
plt.plot(k["imb"],k["MidMove"],label="Emperical 25th move")
plt.legend()

plt.imshow(Q2 + R2, cmap='viridis', interpolation='nearest')
plt.xlabel("Next imbalance bucket")
plt.ylabel("Current imbalance bucket")
plt. title ("Heat map of actual transition  matrix")
plt.show()

plt.imshow(R2, cmap='viridis', interpolation='nearest')
plt.xlabel("Next imbalance bucket")
plt.ylabel("Current imbalance bucket")
plt. title ("Heat map of transition matrix in case  of  a mid−price changes")
plt.show()

W=np.linalg.matrix_power(B,100)
for  i  in range(0,n_spread):
    plt.plot(imb,W[0][(0+i∗n_imb):(n_imb+i∗n_imb)],label="spread = "+str(i+1)+" tick
        ",marker='o')

plt.legend(loc='upper left')
plt. title ('Stationary  Distribution')
plt.xlabel('Imbalance')
```

6.2 Random Forest Regressor + Classifier Python codes

```
df  = pd.read_csv("dataframe.csv")
```

```
df[' fills '] = pd.Series([[item.split(',') for item in elem.strip("][").split('],[')] for
    elem in df[' fills ']])

fills_changed = {}
for i in df.index:
    if df[' fills '][i] != [['']]:
        fills_changed[i] = [[eval(i) for i in elem] for elem in df[' fills '][i]]
    else:
        fills_changed[i] = [['']]

orders_size = []
orders_filled = []
for i in df.index:
    if df[' fills '][i]!= [['']]:
        os_val = np.sum(np.array(fills_changed[i]), 0)[1]
        of_val = np.sum(np.array(fills_changed[i]), 0)[2]
        orders_size.append(os_val)
        orders_filled.append(of_val)
    else:
        os_val = np.nan
        of_val = np.nan
        orders_size.append(os_val)
        orders_filled.append(of_val)

df['orders_size'] = pd.Series(orders_size)
df[' orders_filled '] = pd.Series( orders_filled )

def clean_data_for_rf(df, tick):
    df['DM'] = df['mid'].shift(-1*tick) - df['wmid']
    df['dummy']=0
    for i in range(1,len(df)):
        if ((df.loc[i,'count']-df.loc[(i-1),'count']) == 1):
            for j in range( 1, (tick+1)):
                df.loc[(i-j),'dummy']=1

    df1 = df[df['dummy']==0]

    return df1

tick=25 ## Predictions for
df_final = clean_data_for_rf(df, tick)
```

```
def RF_func(df,tick):

    df = df.drop(columns=['timestamp'])
    del df['time']
    del df[' fills ']
    del df['dM']

    del df[' orders_size ']
    del df[' orders_filled ']

    df['bid_price_1'] = pd.Series([df['price_same_1'][t] if df['trade_side'][t] == 1
                                   else df['price_contra_1'][t] for t in df.index])

    df['ask_price_1'] = pd.Series([df['price_contra_1'][t] if df['trade_side'][t] == 1
                                   else df['price_same_1'][t] for t in df.index])

    df['bid_price_2'] = pd.Series([df['price_same_2'][t] if df['trade_side'][t] == 1
                                   else df['price_contra_2'][t] for t in df.index])

    df['ask_price_2'] = pd.Series([df['price_contra_2'][t] if df['trade_side'][t] == 1
                                   else df['price_same_2'][t] for t in df.index])

    df['bid_price_3'] = pd.Series([df['price_same_3'][t] if df['trade_side'][t] == 1
                                   else df['price_contra_3'][t] for t in df.index])

    df['ask_price_3'] = pd.Series([df['price_contra_3'][t] if df['trade_side'][t] == 1
                                   else df['price_same_3'][t] for t in df.index])

    df['bid_spread_12'] = pd.Series([df['bid_price_1'][t] - df['bid_price_2'][t] for t in
        df.index])

    df['bid_spread_23'] = pd.Series([df['bid_price_2'][t] - df['bid_price_3'][t] for t in
        df.index])

    df['ask_spread_12'] = pd.Series([df['ask_price_2'][t] - df['ask_price_1'][t] for t in
        df.index])

    df['ask_spread_23'] = pd.Series([df['ask_price_3'][t] - df['ask_price_2'][t] for t in
        df.index])

    df['vol_weighted_bid'] = pd.Series([(df['bid_price_1'][t]*df['bs'][t]+df['bid_price_2
        '][t]*df['bs2'][t]+
                                        df['bid_price_3'][t]*df['bs3'][t])
                                       /(df['bs'][t]+df['bs2'][t]+ df['bs3'][t])  for t
                                          in df.index])

df['vol_weighted_ask'] = pd.Series([(df['ask_price_1'][t]*df['as'][t]+df['ask_price_2
   '][t]*df['as2'][t]+
                                        df['ask_price_3'][t]*df['as3'][t])
                                       /(df['as'][t]+df['as2'][t]+ df['as3'][t])  for t
                                          in df.index])
#money
df['money_1'] = pd.Series([(df['bid_price_1'][t]*df['bs'][t]+df['ask_price_1'][t]*df[
   'as'][t])
                                for t in df.index])
df['money_2'] = pd.Series([(df['bid_price_2'][t]*df['bs2'][t]+df['ask_price_2'][t]*df
   ['as2'][t])
                                for t in df.index])
df['money_3'] = pd.Series([(df['bid_price_3'][t]*df['bs3'][t]+df['ask_price_3'][t]*df
   ['as3'][t])
                                for t in df.index])

#time series aggregates -> mean, std, mad, skewness, kurtosis
#weighted mid
df['wmid_mean'] = df['wmid'].rolling(20).mean().bfill()
df['wmid_std'] = df['wmid'].rolling(20).std().bfill()
df['wmid_mad'] = abs(df['wmid'] - df['wmid'].rolling(20).mean()).rolling(20).mean().
   bfill()
df['wmid_skew'] = df['wmid'].rolling(20).skew().bfill()
df['wmid_kurt'] = df['wmid'].rolling(20).kurt().bfill()
#VWAP
df['vwap_mean'] = df['VWAP'].rolling(20).mean().bfill()
df['vwap_std'] = df['VWAP'].rolling(20).std().bfill()
df['vwap_mad'] = abs(df['VWAP'] - df['VWAP'].rolling(20).mean()).rolling(20).mean
   ().bfill()
df['vwap_skew'] = df['VWAP'].rolling(20).skew().bfill()
df['vwap_kurt'] = df['VWAP'].rolling(20).kurt().bfill()
#imbalance
df['imb_mean'] = df['imb'].rolling(20).mean().bfill()
df['imb_std'] = df['imb'].rolling(20).std().bfill()
df['imb_mad'] = abs(df['imb'] - df['imb'].rolling(20).mean()).rolling(20).mean().
   bfill()
df['imb_skew'] = df['imb'].rolling(20).skew().bfill()
df['imb_kurt'] = df['imb'].rolling(20).kurt().bfill()
```

```
    df.dropna(inplace = True)

    y = df['DM'].ravel()
    x = df.drop(columns=['DM']) #df[['imb', 'spread']]
    x_train, x_test, y_train, y_test = train_test_split (x, y, test_size =0.30,
        random_state=42)
    model = RandomForestRegressor()
    model.fit(x_train, y_train)
    y_pred = model.predict(x_test)
    plt.figure( figsize =(10,10))
    plt.scatter(y_test, y_pred)
    p1 = max(max(y_pred), max(y_test))
    p2 = min(min(y_pred), min(y_test))
    plt.plot([p1, p2], [p1, p2], 'b-')
    plt.xlabel('True Values', fontsize=15)
    plt.ylabel('Predictions', fontsize=15)
    plt.axis('equal')
    plt.show()

    return y_test, y_pred, model, x_train, x_test
```

6.3 XGBoost Regressor + Classifier Python codes

```
import numpy as np
import pandas as pd
import random
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score,mean_squared_error, mean_absolute_error,
    accuracy_score, roc_curve, roc_auc_score
import seaborn as sns
from sklearn.model_selection import RandomizedSearchCV
import matplotlib.pyplot as plt
import xgboost as xgb
import shap
from sklearn.preprocessing import LabelEncoder

def clean_data_for_rf (df, tick):
    df['DM'] = df['mid'].shift(-1*tick) - df['wmid'] ### PS: its wmid
    df['dummy']=0
    #print(df.shape)
    for i in range(1,len(df)):
        if ((df.loc[i,'count']-df.loc[(i-1),'count']) == 1):
            for j in range( 1, (tick+1)):
                df.loc[(i-j),'dummy']=1

    df1 = df[df['dummy']==0]

    return df1

df = pd.read_csv("dataframe.csv")
df[' fills '] = pd.Series([[item.split(',') for item in elem.strip("][").split('],[')] for
    elem in df[' fills ']])

fills_changed = {}
for i in df.index:
    if df[' fills '][i] != [['']]:
        fills_changed[i] = [[eval(i) for i in elem] for elem in df[' fills '][i]]
    else:
        fills_changed[i] = [['']]


orders_size = []
orders_filled = []
for i in df.index:
    if df[' fills '][i]!= [['']]:
        os_val = np.sum(np.array(fills_changed[i]), 0)[1]
        of_val = np.sum(np.array(fills_changed[i]), 0)[2]
        orders_size.append(os_val)
        orders_filled.append(of_val)
    else:
        os_val = np.nan
        of_val = np.nan
        orders_size.append(os_val)
        orders_filled.append(of_val)

df['orders_size'] = pd.Series(orders_size)
df[' orders_filled '] = pd.Series( orders_filled )

tick=25 ## Predictions for
df_final = clean_data_for_rf(df, tick)
```

```
df_final['direction'] = np.where(df_final['DM']<0, 0, 1)

def XGBoost_func_Regressor(df_final):
    y = df_final.DM.ravel()
    y[np.isnan(y)] = 0
    df = df_final.drop(['DM','direction','timestamp','time',' fills ','dM','dummy'], axis
        =1)

    df['bid_price_1'] = pd.Series([df['price_same_1'][t] if df['trade_side'][t] == 1
                                   else df['price_contra_1'][t] for t in df.index])

    df['ask_price_1'] = pd.Series([df['price_contra_1'][t] if df['trade_side'][t] == 1
                                   else df['price_same_1'][t] for t in df.index])

    df['bid_price_2'] = pd.Series([df['price_same_2'][t] if df['trade_side'][t] == 1
                                   else df['price_contra_2'][t] for t in df.index])

    df['ask_price_2'] = pd.Series([df['price_contra_2'][t] if df['trade_side'][t] == 1
                                   else df['price_same_2'][t] for t in df.index])

    df['bid_price_3'] = pd.Series([df['price_same_3'][t] if df['trade_side'][t] == 1
                                   else df['price_contra_3'][t] for t in df.index])

    df['ask_price_3'] = pd.Series([df['price_contra_3'][t] if df['trade_side'][t] == 1
                                   else df['price_same_3'][t] for t in df.index])
    #spread between bid and ask levels
    df['bid_spread_12'] = pd.Series([df['bid_price_1'][t] - df['bid_price_2'][t] for t in
         df.index])

    df['bid_spread_23'] = pd.Series([df['bid_price_2'][t] - df['bid_price_3'][t] for t in
         df.index])

    df['ask_spread_12'] = pd.Series([df['ask_price_2'][t] - df['ask_price_1'][t] for t in
         df.index])

    df['ask_spread_23'] = pd.Series([df['ask_price_3'][t] - df['ask_price_2'][t] for t in
         df.index])

    #volume weighted bid and ask
    df['vol_weighted_bid'] = pd.Series([(df['bid_price_1'][t]*df['bs'][t]+df['bid_price_2
        '][t]*df['bs2'][t]+
                                        df['bid_price_3'][t]*df['bs3'][t])
                                       /(df['bs'][t]+df['bs2'][t]+ df['bs3'][t]) for t in df
                                          .index])

    df['vol_weighted_ask'] = pd.Series([(df['ask_price_1'][t]*df['as'][t]+df['ask_price_2
        '][t]*df['as2'][t]+
                                        df['ask_price_3'][t]*df['as3'][t])
                                       /(df['as'][t]+df['as2'][t]+ df['as3'][t]) for t in df
                                          .index])
    #money indicators
    df['money_1'] = pd.Series([(df['bid_price_1'][t]*df['bs'][t]+df['ask_price_1'][t]*df[
        'as'][t])
                           for t in df.index])
    df['money_2'] = pd.Series([(df['bid_price_2'][t]*df['bs2'][t]+df['ask_price_2'][t]*df
        ['as2'][t])
                           for t in df.index])
    df['money_3'] = pd.Series([(df['bid_price_3'][t]*df['bs3'][t]+df['ask_price_3'][t]*df
        ['as3'][t])
                           for t in df.index])

    #time series aggregates -> mean, std, mad, skewness, kurtosis
    #weighted mid
    df['wmid_mean'] = df['wmid'].rolling(20).mean().bfill()
    df['wmid_std'] = df['wmid'].rolling(20).std().bfill()
    df['wmid_mad'] = abs(df['wmid'] - df['wmid'].rolling(20).mean()).rolling(20).mean().
        bfill()
    df['wmid_skew'] = df['wmid'].rolling(20).skew().bfill()
    df['wmid_kurt'] = df['wmid'].rolling(20).kurt().bfill()
    #VWAP
    df['vwap_mean'] = df['VWAP'].rolling(20).mean().bfill()
    df['vwap_std'] = df['VWAP'].rolling(20).std().bfill()
    df['vwap_mad'] = abs(df['VWAP'] - df['VWAP'].rolling(20).mean()).rolling(20).mean
        ().bfill()
    df['vwap_skew'] = df['VWAP'].rolling(20).skew().bfill()
    df['vwap_kurt'] = df['VWAP'].rolling(20).kurt().bfill()
    #imbalance
    df['imb_mean'] = df['imb'].rolling(20).mean().bfill()
    df['imb_std'] = df['imb'].rolling(20).std().bfill()
    df['imb_mad'] = abs(df['imb'] - df['imb'].rolling(20).mean()).rolling(20).mean().
        bfill()
    df['imb_skew'] = df['imb'].rolling(20).skew().bfill()
    df['imb_kurt'] = df['imb'].rolling(20).kurt().bfill()
```

```
    x = df

    # define the train set and test set
    x_train, x_val, y_train, y_val = train_test_split (x, y, test_size =0.3)
    xgb_regressor = xgb.XGBRegressor(
    booster = "gbtree",               # Which booster to use
    objective = "reg:squarederror",   # Specify the learning task
    reg_lambda = 10,                  # L2 regularization term
    gamma = 0,                        # Minimum loss reduction
    max_depth = 20,                   # Maximum tree depth
    learning_rate = 0.35              # Learning rate, eta
    )

    xgb_train = xgb.DMatrix(x_train,label = y_train)
    xgb_parm = xgb_regressor.get_xgb_params()
    xgb_cvresult = xgb.cv(xgb_parm, xgb_train,
                          num_boost_round = 20,
                          metrics = "rmse",
                          nfold = 10,
                           stratified =False,
                          seed=20,
                          early_stopping_rounds=3)

    # Update parameters (# of trees)
    xgb_regressor.set_params(n_estimators = xgb_cvresult.shape[0])

    xgb_a = xgb_regressor.fit(x_train,y_train)
    y_pred_a = xgb_regressor.predict(x_val, ntree_limit = xgb_cvresult.shape[0])
    mse_a = mean_squared_error(y_val, y_pred_a)

    residuals = y_val-y_pred_a
    plt.scatter(y_pred_a, residuals)
    plt.title("Improved XGBoost Residual plot")
    plt.xlabel("Predicted Y Values")
    plt.ylabel("Residuals")
    plt.axhline(y=0, color='r', linestyle='--')
    plt.show()

    return (np.sqrt(mse_a))

def XGBoost_func_Classifier(df_final):
```

```
y = df_final . direction . ravel ()
y[np.isnan(y)] = 0
df = df_final .drop(['DM','direction' , 'timestamp','time', ' fills ' , 'dM','dummy'], axis
    =1)

df[ ' bid_price_1 ' ] = pd.Series([df[ 'price_same_1' ][ t ] if df[ ' trade_side ' ][ t ] == 1
                            else df[ ' price_contra_1 ' ][ t ] for t in df.index])

df[ ' ask_price_1 ' ] = pd.Series([df[ ' price_contra_1 ' ][ t ] if df[ ' trade_side ' ][ t ] == 1
                            else df[ 'price_same_1' ][ t ] for t in df.index])

df[ ' bid_price_2 ' ] = pd.Series([df[ 'price_same_2' ][ t ] if df[ ' trade_side ' ][ t ] == 1
                            else df[ ' price_contra_2 ' ][ t ] for t in df.index])

df[ ' ask_price_2 ' ] = pd.Series([df[ ' price_contra_2 ' ][ t ] if df[ ' trade_side ' ][ t ] == 1
                            else df[ 'price_same_2' ][ t ] for t in df.index])

df[ ' bid_price_3 ' ] = pd.Series([df[ 'price_same_3' ][ t ] if df[ ' trade_side ' ][ t ] == 1
                            else df[ ' price_contra_3 ' ][ t ] for t in df.index])

df[ ' ask_price_3 ' ] = pd.Series([df[ ' price_contra_3 ' ][ t ] if df[ ' trade_side ' ][ t ] == 1
                            else df[ 'price_same_3' ][ t ] for t in df.index])
#spread between bid and ask levels
df[ 'bid_spread_12' ] = pd.Series([df[ ' bid_price_1 ' ][ t ] − df[' bid_price_2 ' ][ t ] for t in
     df.index])

df[ 'bid_spread_23' ] = pd.Series([df[ ' bid_price_2 ' ][ t ] − df[' bid_price_3 ' ][ t ] for t in
     df.index])

df[ 'ask_spread_12' ] = pd.Series([df[ ' ask_price_2 ' ][ t ] − df[' ask_price_1 ' ][ t ] for t in
     df.index])

df[ 'ask_spread_23' ] = pd.Series([df[ ' ask_price_3 ' ][ t ] − df[' ask_price_2 ' ][ t ] for t in
     df.index])

#volume weighted bid and ask
df[ 'vol_weighted_bid' ] = pd.Series([(df[ ' bid_price_1 ' ][ t]∗df[ 'bs' ][ t]+df[' bid_price_2
    ' ][ t]∗df[ 'bs2' ][ t]+
                                   df[ ' bid_price_3 ' ][ t]∗df[ 'bs3' ][ t ])
                                 /(df[ 'bs' ][ t]+df['bs2' ][ t]+ df[ 'bs3' ][ t ]) for t in df
                                     .index])
```

```
df['vol_weighted_ask'] = pd.Series([(df['ask_price_1'][t]*df['as'][t]+df['ask_price_2'][t]*df['as2'][t]+
                                     df['ask_price_3'][t]*df['as3'][t])
                                    /(df['as'][t]+df['as2'][t]+ df['as3'][t]) for t in df.index])
#money indicators
df['money_1'] = pd.Series([(df['bid_price_1'][t]*df['bs'][t]+df['ask_price_1'][t]*df['as'][t])
                          for t in df.index])
df['money_2'] = pd.Series([(df['bid_price_2'][t]*df['bs2'][t]+df['ask_price_2'][t]*df['as2'][t])
                          for t in df.index])
df['money_3'] = pd.Series([(df['bid_price_3'][t]*df['bs3'][t]+df['ask_price_3'][t]*df['as3'][t])
                          for t in df.index])

#time series aggregates -> mean, std, mad, skewness, kurtosis
#weighted mid
df['wmid_mean'] = df['wmid'].rolling(20).mean().bfill()
df['wmid_std'] = df['wmid'].rolling(20).std().bfill()
df['wmid_mad'] = abs(df['wmid'] - df['wmid'].rolling(20).mean()).rolling(20).mean().bfill()
df['wmid_skew'] = df['wmid'].rolling(20).skew().bfill()
df['wmid_kurt'] = df['wmid'].rolling(20).kurt().bfill()
#VWAP
df['vwap_mean'] = df['VWAP'].rolling(20).mean().bfill()
df['vwap_std'] = df['VWAP'].rolling(20).std().bfill()
df['vwap_mad'] = abs(df['VWAP'] - df['VWAP'].rolling(20).mean()).rolling(20).mean().bfill()
df['vwap_skew'] = df['VWAP'].rolling(20).skew().bfill()
df['vwap_kurt'] = df['VWAP'].rolling(20).kurt().bfill()
#imbalance
df['imb_mean'] = df['imb'].rolling(20).mean().bfill()
df['imb_std'] = df['imb'].rolling(20).std().bfill()
df['imb_mad'] = abs(df['imb'] - df['imb'].rolling(20).mean()).rolling(20).mean().bfill()
df['imb_skew'] = df['imb'].rolling(20).skew().bfill()
df['imb_kurt'] = df['imb'].rolling(20).kurt().bfill()

x = df
```

```
    # define the train set and test set
    x_train, x_val, y_train, y_val = train_test_split (x, y, test_size =0.3)
    xgb_regressor = xgb.XGBClassifier(
    booster = "gbtree",               # Which booster to use
    objective = "reg:squarederror", # Specify the learning task
    reg_lambda = 10,                   # L2 regularization term
    gamma = 0,                        # Minimum loss reduction
    max_depth = 20,                    # Maximum tree depth
    learning_rate = 0.35               # Learning rate, eta
    )

    xgb_train = xgb.DMatrix(x_train,label = y_train)
    xgb_parm = xgb_regressor.get_xgb_params()
    xgb_cvresult = xgb.cv(xgb_parm, xgb_train,
                          num_boost_round = 20,
                          metrics = "rmse",
                          nfold = 10,
                           stratified =False,
                          seed=20,
                          early_stopping_rounds=3)

    # Update parameters (# of trees)
    xgb_regressor.set_params(n_estimators = xgb_cvresult.shape[0])

    ## To print shap uncomment this
#     explainer = shap.TreeExplainer(xgb_regressor)
#     shap_values = explainer.shap_values(x_train)
#.    shap.summary_plot(shap_values, x_train)
    xgb_a = xgb_regressor.fit (x_train, y_train)
    y_pred_a = xgb_regressor.predict(x_val, ntree_limit = xgb_cvresult.shape[0])
    mse_a = mean_squared_error(y_val, y_pred_a)

    return accuracy_score(y_val,y_pred_a)

rmse_reg = XGBoost_func_Regressor(df_final)
accuracySc_class = XGBoost_func_Classifier(df_final)
```