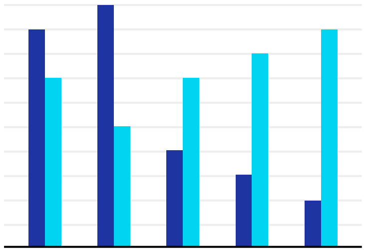
Predict US stocks closing movements

Ke Zhang
Data Science Institute
2023.10.17

Github: https://github.com/kzhangaz/DATA1030---Predict-US-stocks-closing-movements

Intro



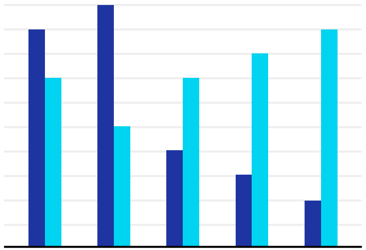
- **The problem?**

  To predict the future price movements of stocks using historic data for the daily ten minute closing auction.

- **Why this is important?**

  The closing auction is considered a crucial period for price discovery, as it reflects the collective sentiment and information of market participants at the end of the trading day. By analyzing the closing movements, investors can gain insights into the market's perception of a stock's value at that specific time. It could help with investment decision making, risk management and market efficiency.

Intro



- **Regression or classification?**

  Regression.

- **Where does the data come from?**

  Kaggle, provided by Optiver.

- **How it was collected?**

  It was collected from the order book and the closing auction of stocks listed on Nasdaq, one of the largest electronic equity platforms in the world.

EDA

5,237,980 * 17

stock_id
date_id
seconds_in_bucket
time_id
row_id

imbalance_size
matched_size
imbalance_buy_sell_flag

reference_price
far_price
near_price
[bid/ask]_price
[bid/ask]_size

Wap: weighted average price
target

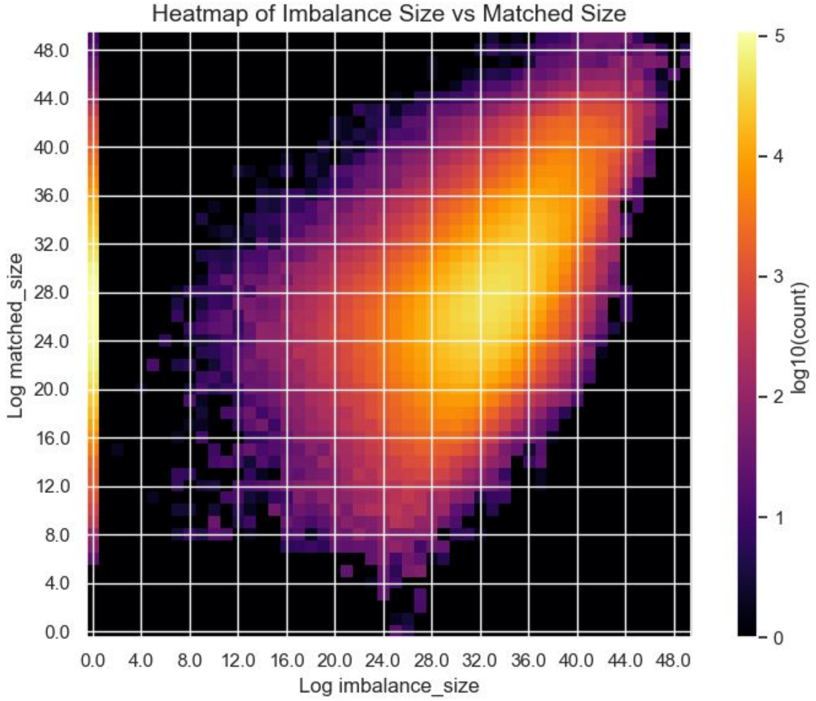
Not i.i.d. !

group structure & time series data

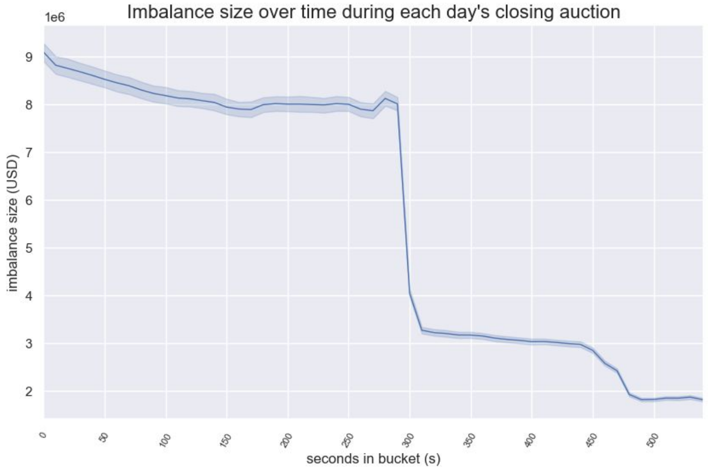
EDA

imbalance_size

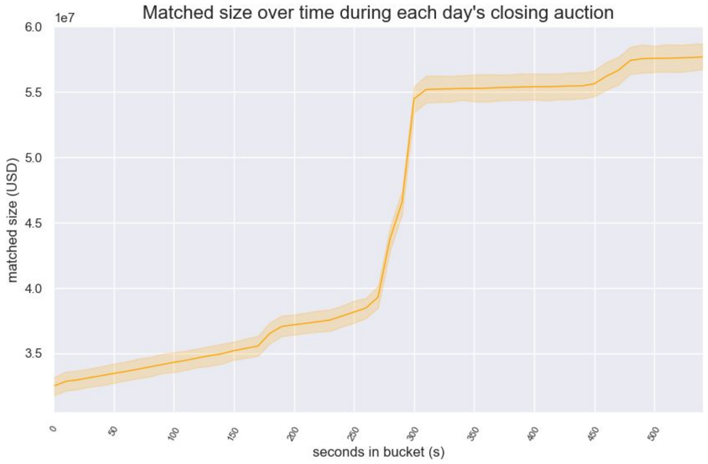
matched_size



Strong positive correlation between imbalance size and matched size.



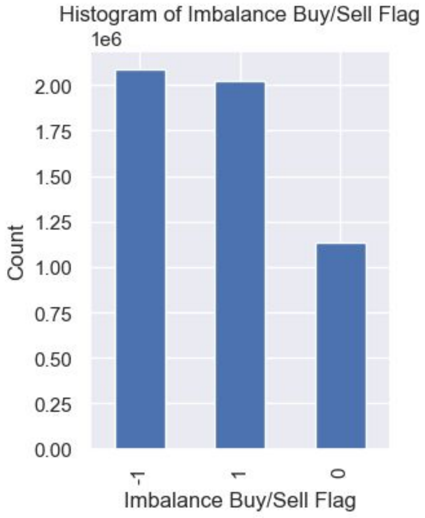
Decreasing trend during each day's closing auction



Increasing trend during each day's closing auction

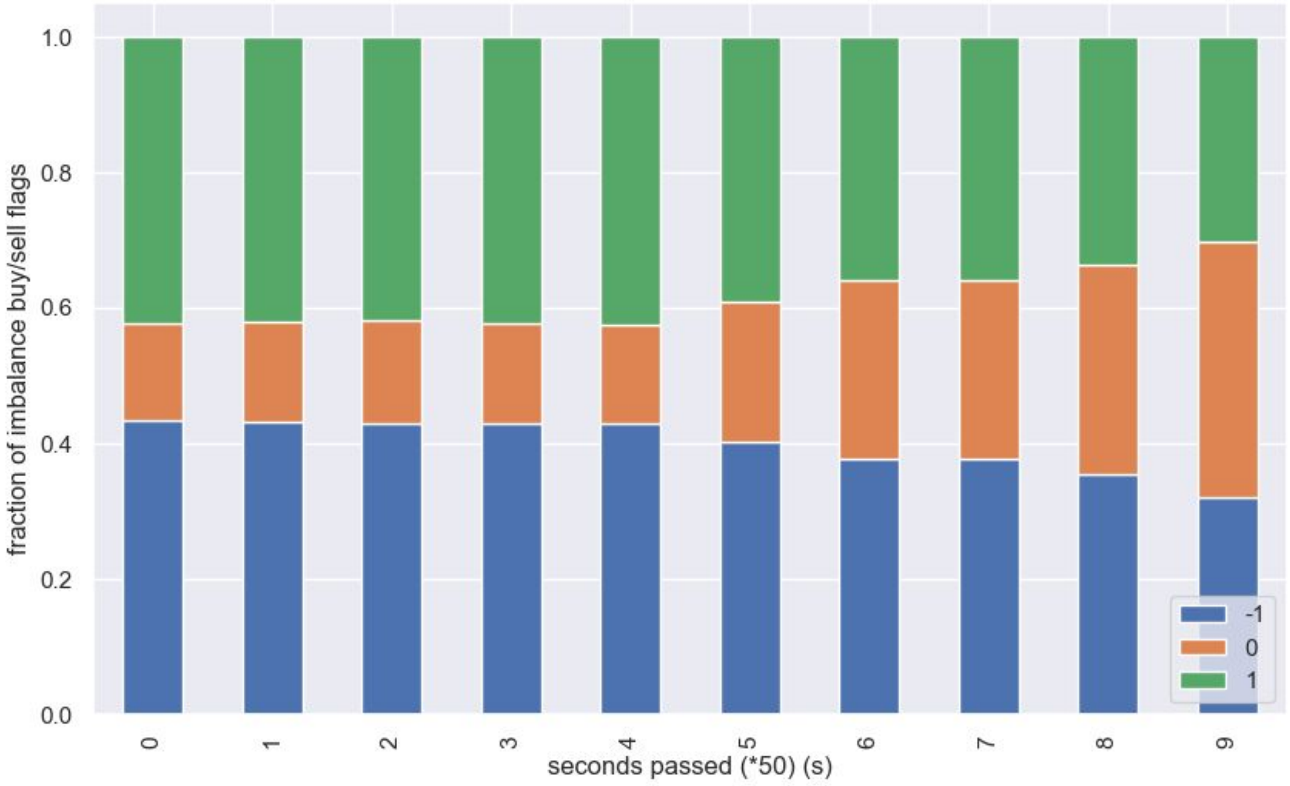
EDA

imbalance_buy_sell_flag



the imbalanced flags for buy/sell are approximately the same, and there are less balanced flags in general

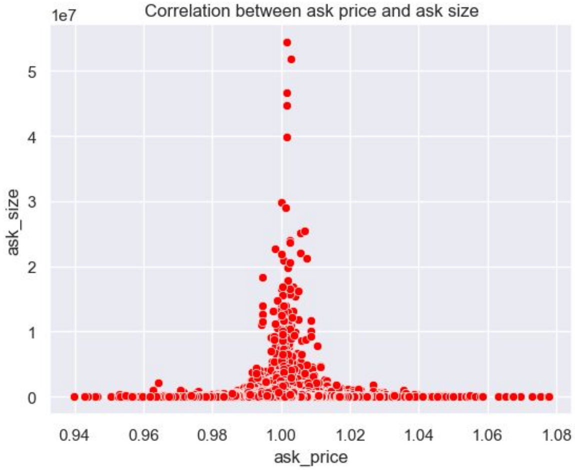
balanced flags increases during the closing period, yet the imbalanced flags for buy/sell remain approximately equal.



EDA

[bid/ask]_price

[bid/ask]_size



The behavior between bid price & bid size is very similar to ask price & ask size. The high bid/ask size occurs around the mean bid/ask price.

Strong correlation between bid price and ask price

Splitting

Since the data is time series data with a group structure, it was a bit tricky to split. There are 200 stocks in total, each has around 20000+ data points. So I took one stock out for testing. As for training and validation data, I use GroupKFold to split the remaining data so that the same stocks goes to the same group.

```
# train-test split
'''
take one stock out for testing (stock 199)
'''
df_other=df[df['stock_id']!=199]
df_test=df[df['stock_id']==199]
X_other=df_other.loc[:, df.columns != 'target']
y_other=df_other['target']
X_test=df_test.loc[:, df.columns != 'target']
y_test=df_test['target']
```
[7] ✓ 1.3s

```
print(f'X_other shape: {X_other.shape}, y_other shape: {y_other.shape}')
print(f'X_test shape: {X_test.shape}, y_test shape: {y_test.shape}')
```
[8] ✓ 0.2s

```
X_other shape: (5216365, 16), y_other shape: (5216365,)
X_test shape: (21615, 16), y_test shape: (21615,)
```

test data split

```
'''
use GroupKFold for train-validation split (preserve the order by default)
'''
group_kfold = GroupKFold(n_splits=3)
stock_group = X_other['stock_id']
i=1
for train_index, val_index in group_kfold.split(X_other, y_other, groups=stock_group):
    X_train, X_val = X_other.iloc[train_index], X_other.iloc[val_index]
    y_train, y_val = y_other.iloc[train_index], y_other.iloc[val_index]
    print(f'{i}-th Fold:')
    print(f'X_train shape: {X_train.shape}, y_train shape: {y_train.shape}')
    print(f'X_val shape: {X_val.shape}, y_val shape: {y_val.shape}')
    i+=1
```
✓ 2.1s

```
1-th Fold:
X_train shape: (3472700, 16), y_train shape: (3472700,)
X_val shape: (1743665, 16), y_val shape: (1743665,)
2-th Fold:
X_train shape: (3477485, 16), y_train shape: (3477485,)
X_val shape: (1738880, 16), y_val shape: (1738880,)
3-th Fold:
X_train shape: (3482545, 16), y_train shape: (3482545,)
X_val shape: (1733820, 16), y_val shape: (1733820,)
```

train/val data split

Preprocessing - Missing Values

```
data dimensions: (5237980, 17)
fraction of missing values in features:
imbalance_size      0.000042
reference_price     0.000042
matched_size        0.000042
far_price           0.552568
near_price          0.545474
bid_price           0.000042
ask_price           0.000042
wap                 0.000042
target              0.000017
fraction of points with missing values: 0.5525683565038431
```

All features with missing values are continuous.

Features with a very small fraction of missing values: Forward Fill & Backward Fill. (Since it's a time series data and I don't want the time gap to be different by dropping rows with missing value)

Far price & near price: the missing values are caused by fo Nasdaq only records these variables from 3:55 pm to 4 pm. Impute all values before 3:55 with 0, use Forward Fill & Backward Fill for missing values in the later half.

Preprocessing

- **Categorical data:** apply one-hot encoding to imbalance_buy_sell_flag since it's categorical
- **Continuous data:** apply StandardScaler to the continuous features for each stock on each day since the data is continuous and time series

Before preprocessing: 5,237,980 * 17
After: 5,237,980 * 19

```python
def preprocessing(X,y):

    # apply one-hot encoding to imbalance_buy_sell_flag
    one_hot_ftrs = ['imbalance_buy_sell_flag']

    # initialize the encoder
    enc = OneHotEncoder(sparse=False)
    enc.fit(X)
    # transform X
    X_ohe = enc.fit(X[one_hot_ftrs])
    X_ohe = enc.transform(X[one_hot_ftrs])
    # print(X_ohe)
    print('X transformed')

    # apply StandardScaler to the continuous features for each stock on each day
    cont_ftrs = ['imbalance_size','reference_price','matched_size','far_price','near_price',\
                 'bid_price','bid_size','ask_price','ask_size','wap']
    scaler_x = StandardScaler()
    X_scaled = X.groupby(['stock_id', 'date_id']).apply(lambda group: pd.DataFrame(scaler_x.fit_transform(group[cont_ftrs]), cc
    X_scaled.reset_index(inplace=True, drop=True)
    print(X_scaled.head())
```

```python
    # apply StandardScaler to target
    scaler_y = StandardScaler()
    y_output = scaler_y.fit_transform(y.to_numpy().reshape(-1, 1))
    print(y_output)

    X_output = pd.concat([pd.DataFrame(X_ohe), X_scaled], axis=1, ignor
    X_output.columns = ['bal_flag_1', 'bal_flag_0', 'bal_flag_m1'] + li
    X_output = X_output.drop(columns=['imbalance_buy_sell_flag','target
    print(f'shape of X after preprocessing: {X_output.shape}')

    return X_output, y_output
```

Thank you!