

## 03\_normalize\_tick\_data

September 29, 2021

# 1 Analyze Order Book Data

## 1.1 Imports & Settings

```
[1]: import pandas as pd
      from pathlib import Path
      import numpy as np
      import seaborn as sns
      import matplotlib as mpl
      import matplotlib.pyplot as plt
      from matplotlib.ticker import FuncFormatter
      from math import pi
      from bokeh.plotting import figure, show
      from scipy.stats import normaltest
```

```
[2]: %matplotlib inline
      pd.set_option('display.float_format', lambda x: '%.2f' % x)
      sns.set_style('whitegrid')
```

```
[3]: data_path = Path('data')
      itch_store = str(data_path / 'itch.h5')
      order_book_store = str(data_path / 'order_book.h5')
      stock = 'AAPL'
      date = '20191030'
      title = '{} | {}'.format(stock, pd.to_datetime(date).date())
```

## 1.2 Load system event data

```
[4]: with pd.HDFStore(itch_store) as store:
      sys_events = store['S'].set_index('event_code').drop_duplicates()
      sys_events.timestamp = sys_events.timestamp.add(pd.to_datetime(date)).dt.
      ↪time
      market_open = sys_events.loc['Q', 'timestamp']
      market_close = sys_events.loc['M', 'timestamp']
```

### 1.3 Trade Summary

We will combine the messages that refer to actual trades to learn about the volumes for each asset.

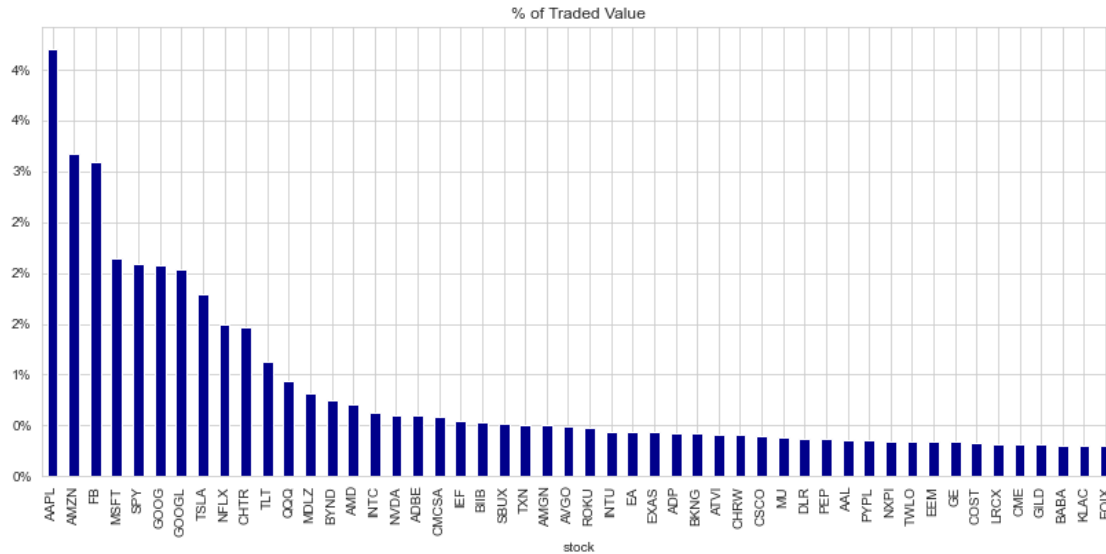
```
[5]: with pd.HDFStore(itch_store) as store:
      stocks = store['R']
      stocks.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 8887 entries, 0 to 8886
Data columns (total 17 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   stock_locate                          8887 non-null   int64
1   tracking_number                       8887 non-null   int64
2   timestamp                             8887 non-null   timedelta64[ns]
3   stock                                8887 non-null   object
4   market_category                      8887 non-null   object
5   financial_status_indicator           8887 non-null   object
6   round_lot_size                       8887 non-null   int64
7   round_lots_only                      8887 non-null   object
8   issue_classification                 8887 non-null   object
9   issue_sub_type                       8887 non-null   object
10  authenticity                          8887 non-null   object
11  short_sale_threshold_indicator        8887 non-null   object
12  ipo_flag                             8887 non-null   object
13  luld_reference_price_tier             8887 non-null   object
14  etp_flag                             8887 non-null   object
15  etp_leverage_factor                  8887 non-null   int64
16  inverse_indicator                    8887 non-null   object
dtypes: int64(4), object(12), timedelta64[ns](1)
memory usage: 1.2+ MB
```

As expected, a small number of the over 8,500 equity securities traded on this day account for most trades

```
[6]: with pd.HDFStore(itch_store) as store:
      stocks = store['R'].loc[:, ['stock_locate', 'stock']]
      trades = store['P'].append(store['Q'].rename(columns={'cross_price': 'price'}), sort=False).merge(stocks)

      trades['value'] = trades.shares.mul(trades.price)
      trades['value_share'] = trades.value.div(trades.value.sum())
      trade_summary = trades.groupby('stock').value_share.sum().
        ↪sort_values(ascending=False)
      trade_summary.iloc[:50].plot.bar(figsize=(14, 6), color='darkblue', title='% of ↪
        ↪Traded Value')
      plt.gca().yaxis.set_major_formatter(FuncFormatter(lambda y, _: '{:.0%}'.
        ↪format(y)))
```



## 1.4 AAPL Trade Summary

```
[7]: with pd.HDFStore(order_book_store) as store:
      trades = store['{}/trades'.format(stock)]
```

```
[8]: trades.price = trades.price.mul(1e-4) # format price
      trades = trades[trades.cross == 0]
      trades = trades.between_time(market_open, market_close).drop('cross', axis=1)
      trades.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 58713 entries, 2019-10-30 09:30:00.010384780 to 2019-10-30
15:59:59.979015439
Data columns (total 2 columns):
#   Column  Non-Null Count  Dtype
---  -
0   shares  58713 non-null        int64
1   price   58713 non-null        float64
dtypes: float64(1), int64(1)
memory usage: 1.3 MB
```

## 1.5 Tick Bars

The trade data is indexed by nanoseconds and is very noisy. The bid-ask bounce, for instance, causes the price to oscillate between the bid and ask prices when trade initiation alternates between buy and sell market orders. To improve the noise-signal ratio and improve the statistical properties, we need to resample and regularize the tick data by aggregating the trading activity.

We typically collect the open (first), low, high, and closing (last) price for the aggregated pe-

riod, alongside the volume-weighted average price (VWAP), the number of shares traded, and the timestamp associated with the data.

```
[9]: tickBars = trades.copy()
tickBars.index = tickBars.index.time
tickBars.price.plot(figsize=(10, 5),
                    title='Tick Bars | {} | {}'.format(stock, pd.
                    ↳to_datetime(date).date()), lw=1)
plt.xlabel('')
plt.tight_layout();
```



### 1.5.1 Test for Normality of tick returns

```
[10]: normaltest(tickBars.price.pct_change().dropna())
```

```
[10]: NormaltestResult(statistic=20684.40456159484, pvalue=0.0)
```

## 1.6 Regularizing Tick Data

### 1.6.1 Price-Volume Chart

We will use the `price_volume` function to compare the price-volume relation for various regularization methods.

```
[11]: def price_volume(df, price='vwap', vol='vol', subtitle=title, fname=None):

    fig, axes = plt.subplots(nrows=2, sharex=True, figsize=(15,8))
    axes[0].plot(df.index, df[price])
    axes[1].bar(df.index, df[vol], width=1/(5*len(df.index)), color='r')
```

```

# formatting
xfmt = mpl.dates.DateFormatter('%H:%M')
axes[1].xaxis.set_major_locator(mpl.dates.HourLocator(interval=3))
axes[1].xaxis.set_major_formatter(xfmt)
axes[1].get_xaxis().set_tick_params(which='major', pad=25)
axes[0].set_title('Price', fontsize=14)
axes[1].set_title('Volume', fontsize=14)
fig.autofmt_xdate()
fig.suptitle(suptitle)
fig.tight_layout()
plt.subplots_adjust(top=0.9);

```

## 1.6.2 Time Bars

Time bars involve trade aggregation by period.

```

[12]: def get_bar_stats(agg_trades):
        vwap = agg_trades.apply(lambda x: np.average(x.price, weights=x.shares)).
        ↳to_frame('vwap')
        ohlc = agg_trades.price.ohlc()
        vol = agg_trades.shares.sum().to_frame('vol')
        txn = agg_trades.shares.size().to_frame('txn')
        return pd.concat([ohlc, vwap, vol, txn], axis=1)

```

We create time bars using the `.resample()` method with the desired period.

```

[13]: resampled = trades.groupby(pd.Grouper(freq='1Min'))
        time_bars = get_bar_stats(resampled)
        normaltest(time_bars.vwap.pct_change().dropna())

```

```

[13]: NormaltestResult(statistic=65.70387182967823, pvalue=5.402384769537968e-15)

```

```

[14]: price_volume(time_bars,
        suptitle=f'Time Bars | {stock} | {pd.to_datetime(date).date()}',
        fname='time_bars')

```



### 1.6.3 Bokeh Candlestick Chart

Alternative visualization using the [bokeh](#) library:

```
[15]: resampled = trades.groupby(pd.Grouper(freq='5Min')) # 5 Min bars for better
      ↪ print
      df = get_bar_stats(resampled)

      increase = df.close > df.open
      decrease = df.open > df.close
      w = 2.5 * 60 * 1000 # 2.5 min in ms

      WIDGETS = "pan, wheel_zoom, box_zoom, reset, save"

      p = figure(x_axis_type='datetime', tools=WIDGETS, plot_width=1500, title =
      ↪ "AAPL Candlestick")
      p.xaxis.major_label_orientation = pi/4
      p.grid.grid_line_alpha=0.4

      p.segment(df.index, df.high, df.index, df.low, color="black")
      p.vbar(df.index[increase], w, df.open[increase], df.close[increase],
      ↪ fill_color="#D5E1DD", line_color="black")
      p.vbar(df.index[decrease], w, df.open[decrease], df.close[decrease],
      ↪ fill_color="#F2583E", line_color="black")
      show(p)
```

### 1.6.4 Volume Bars

Time bars smooth some of the noise contained in the raw tick data but may fail to account for the fragmentation of orders. Execution-focused algorithmic trading may aim to match the volume weighted average price (VWAP) over a given period, and will divide a single order into multiple trades and place orders according to historical patterns. Time bars would treat the same order differently, even though no new information has arrived in the market.

Volume bars offer an alternative by aggregating trade data according to volume. We can accomplish this as follows:

```
[16]: with pd.HDFStore(order_book_store) as store:
        trades = store['{}/trades'.format(stock)]

        trades.price = trades.price.mul(1e-4)
        trades = trades[trades.cross == 0]
        trades = trades.between_time(market_open, market_close).drop('cross', axis=1)
        trades.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 58713 entries, 2019-10-30 09:30:00.010384780 to 2019-10-30
15:59:59.979015439
Data columns (total 2 columns):
#   Column  Non-Null Count  Dtype
---  -
0   shares  58713 non-null      int64
1   price   58713 non-null      float64
dtypes: float64(1), int64(1)
memory usage: 1.3 MB
```

```
[17]: trades_per_min = trades.shares.sum()/(60*7.5) # min per trading day
        trades['cumul_vol'] = trades.shares.cumsum()
```

```
[18]: df = trades.reset_index()
        by_vol = df.groupby(df.cumul_vol.div(trades_per_min).round().astype(int))
        volBars = pd.concat([by_vol.timestamp.last().to_frame('timestamp'),
                               ↳get_bar_stats(by_vol)], axis=1)
        volBars.head()
```

```
[18]:
```

	timestamp	open	high	low	close	vwap	\
cumul_vol							
0	2019-10-30 09:30:01.658183871	244.83	244.94	244.72	244.82	244.77	
1	2019-10-30 09:30:02.124784227	244.82	244.85	244.76	244.77	244.79	
2	2019-10-30 09:30:04.690393971	244.77	244.97	244.66	244.97	244.80	
3	2019-10-30 09:30:13.816494419	244.98	245.00	244.88	245.00	244.98	
4	2019-10-30 09:30:17.522394656	245.00	245.25	245.00	245.10	245.10	

	vol	txn
cumul_vol		

0	5317	57
1	9127	44
2	12257	76
3	10675	124
4	10679	118

```
[19]: price_volume(volBars.set_index('timestamp'),
                subtitle=f'Volume Bars | {stock} | {pd.to_datetime(date).date()}',
                fname='volume_bars')
```



```
[20]: normaltest(volBars.vwap.dropna())
```

```
[20]: NormaltestResult(statistic=26.54669921949387, pvalue=1.7197190736584466e-06)
```

### 1.6.5 Dollar Bars

```
[21]: with pd.HDFStore(order_book_store) as store:
        trades = store['{}/trades'.format(stock)]

        trades.price = trades.price.mul(1e-4)
        trades = trades[trades.cross == 0]
        trades = trades.between_time(market_open, market_close).drop('cross', axis=1)
        trades.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 58713 entries, 2019-10-30 09:30:00.010384780 to 2019-10-30
15:59:59.979015439
Data columns (total 2 columns):
```



```

#    Column  Non-Null Count  Dtype
---  -
0    shares  58713 non-null    int64
1    price   58713 non-null    float64
dtypes: float64(1), int64(1)
memory usage: 1.3 MB

```

```

[22]: value_per_min = trades.shares.mul(trades.price).sum()/(60*7.5) # min per trading day
      trades['cumul_val'] = trades.shares.mul(trades.price).cumsum()

```

```

[23]: df = trades.reset_index()
      by_value = df.groupby(df.cumul_val.div(value_per_min).round().astype(int))
      dollarBars = pd.concat([by_value.timestamp.last().to_frame('timestamp'),
                              by_value.get_bar_stats(),
                              axis=1)
      dollarBars.head()

```

```

[23]:
      timestamp  open  high  low  close  vwap \
cumul_val
0    2019-10-30 09:30:01.658171058 244.83 244.94 244.72 244.82 244.77
1    2019-10-30 09:30:02.124784227 244.82 244.85 244.76 244.77 244.79
2    2019-10-30 09:30:04.095544194 244.77 244.96 244.66 244.87 244.80
3    2019-10-30 09:30:13.816494419 244.87 245.00 244.87 245.00 244.98
4    2019-10-30 09:30:17.522098123 245.00 245.25 245.00 245.13 245.09

      vol  txn
cumul_val
0         5287    56
1         9157    45
2        12047    71
3        10474   127
4        10690   116

```

```

[24]: price_volume(dollarBars.set_index('timestamp'),
                  subtitle=f'Dollar Bars | {stock} | {pd.to_datetime(date).date()}',
                  fname='dollarBars')

```

Dollar Bars | AAPL | 2019-10-30

