Asg_ash

February 21, 2024

Assignment 3

1 Author: Ashutosh Ekade

Question 1 Posted on absalon is a press release from the European Securities and Markets Authoriy regarding its decision to prohibit sales of binary options to retail investors. Read the article and answer the following questions:

1. What is the motivation behind this decision? How does this relate to the models we have seen in class?
2. What effect will this decision have on liquidity in the binary options market?
3. Measures announced in the press release differ between the binary option and CFD markets. How will the effects of the regulation be different across the two markets?

Be concise and to the point. Please try to keep your answer less than 100 words (but not just one sentense).

Solution 1

1. The motivation behind decision is for protecting the retail investors as the underlying products are quite complex and this in turn results in heavy losses in retail accounts (74-89%). Moreover, there is lack of transparency in these products. The models we have seen in class overlook irrational investor behaviors and assume that the markets are efficient, which is not the case always. Information assymetry results in the profits favouring the providers of these products over the retail investors.
2. This decision of prohibiting the distribution and marketing of these contracts will significantly reduce the liquidity of the binary options. Bid-ask spreads may become wider, leading to inefficient markets.
3. Complete ban in the binary option contracts will potential losses for retail investors, reduce liquidity and widen the bid-ask spreads. Stricter regulations like leverage limits, margin close-out rules, and negative balance protection in CFDs will aim to reduce losses and risks for retail investors. However, these restrictions may also reduce market participation and liquidity, increase costs for compliant providers, potentially impacting overall market size. Overall, the decision prioritizes investor protection over market efficiency in these specific cases due to the significant risks identified.

Question 2 Trade data In the following use the trade data in tqBAC.csv. Denote trade prices by p_t and mid-quotes by m_t. * Sign each trade based on Lee-ready algorithm * Calcualte Spread, Effective Spread and Realized Spread by EXCHANGES * Examine order correlation

Import the relevant modules

[1]:
```
### In this project, I will use datatable (quicker) rather pandas to manipulate␣
↪data. You can see which one is more intuitive for you
from datetime import datetime, timedelta
import datatable as dt
from datatable import dt, f, by, update, shift
import statsmodels.formula.api as smf
import numpy as np
import matplotlib.pyplot as plt
import os
```

```
<IPython.core.display.HTML object>
```

Lee and Ready algorithm In the database reported by exchanges, the buyer and seller identify is not revealed. This is very different from bond database you worked in the previous assignment, which was reported by dealers and records the trade direction of dealers. Given that every trade involves buyers and sellers, how do we know which side market maker (liquidity supplier) stands and customers (liquidity demander) stands? The simple answer is given by the Lee and Ready algorithm as follows. The idea is to assign trade directions based on wheter trades happen around ask or bid prices.

A typical classification is * buyer-initiated if pt > mt * buyer-initiated if pt = mt and pt > pt−1 (downtick) * seller-initiated if pt < mt * seller-initiated if pt = mt and pt < pt−1 (uptick)

[2]:
```
trade_df = dt.fread('./tqBAC.csv')
trade_df = trade_df[:, f[:].extend({"pt_1": dt.shift(f.PRICE, n=1)})]
print(trade_df.shape)
print(trade_df.head())
```

```
(24460, 11)
   | date                      SYMBOL  EX       PRICE    SIZE  COND      BID
BIDSIZ       OFR  OFRSIZ     pt_1
   | str32                     str32   str32  float64   int32  void  float64
int32  float64   int32  float64
-- + ------------------------  ------  -----  -------  ------  ----  -------
------  -------  ------  -------
 0 | 2017-09-19T14:30:00.009Z  BAC     P        24.7     20000    NA    24.72
6     24.74      100    NA
 1 | 2017-09-19T14:30:01.361Z  BAC     T        24.71      300    NA    24.72
6     24.74      100    24.7
 2 | 2017-09-19T14:30:01.525Z  BAC     P        24.7       100    NA    24.72
6     24.74      100    24.71
 3 | 2017-09-19T14:30:02.990Z  BAC     N        24.73   429950    NA    24.72
6     24.74      100    24.7
 4 | 2017-09-19T14:30:02.996Z  BAC     N        24.72      100    NA    24.72
```

```
35     24.74      420    24.73
  5 | 2017-09-19T14:30:03.006Z  BAC     T       24.725     200    NA    24.72
9     24.73        4    24.72
  6 | 2017-09-19T14:30:03.009Z  BAC     B       24.73      100    NA    24.72
9     24.73        4    24.725
  7 | 2017-09-19T14:30:03.026Z  BAC     K       24.73     3500    NA    24.72
55     24.74      525    24.73
  8 | 2017-09-19T14:30:03.032Z  BAC     N       24.73      778    NA    24.72
51     24.74      424    24.73
  9 | 2017-09-19T14:30:03.049Z  BAC     T       24.72     1645    NA    24.71
68     24.72      137    24.73
[10 rows x 11 columns]
```

[3]:

```
# Extract BID and OFR values
bid_values = trade_df[:, 'BID'].to_numpy().flatten()
ofr_values = trade_df[:, 'OFR'].to_numpy().flatten()

# Calculate mt (average of BID and OFR)
mt = (bid_values + ofr_values) / 2
trade_df = trade_df[:, f[:].extend({"mt": mt})]
# Calculate pt-1
pt_1 = dt.shift(trade_df[:, 'PRICE'])

# Extract PRICE values
pt_1_values = pt_1[:, 0].to_numpy().flatten()

# Calculate pt
pt = trade_df[:, 'PRICE'].to_numpy().flatten()

# Calculate classification based on provided logic
buyer_initiated = (pt > mt) | ((pt == mt) & (pt > pt_1_values))
seller_initiated = (pt < mt) | ((pt == mt) & (pt < pt_1_values))

# Update COND column based on classification
trade_df[:, dt.update(COND=dt.ifelse(buyer_initiated, 1, dt.
 ↪ifelse(seller_initiated, -1, 0)))]

# Print the updated datatable
print(trade_df)
```

```
      | date                      SYMBOL  EX       PRICE    SIZE   COND      BID
BIDSIZ      OFR   OFRSIZ      pt_1       mt
      | str32                     str32   str32  float64   int32  int32  float64
int32  float64   int32  float64  float64
----- + ------------------------  ------  -----  -------  ------  -----  -------
------  -------  ------  -------  -------
    0 | 2017-09-19T14:30:00.009Z  BAC     P      24.7      20000     -1    24.72
```

```
6   24.74      100  NA       24.73
    1 | 2017-09-19T14:30:01.361Z  BAC     T      24.71       300     -1    24.72
6   24.74      100  24.7     24.73
    2 | 2017-09-19T14:30:01.525Z  BAC     P      24.7        100     -1    24.72
6   24.74      100  24.71    24.73
    3 | 2017-09-19T14:30:02.990Z  BAC     N      24.73    429950      1    24.72
6   24.74      100  24.7     24.73
    4 | 2017-09-19T14:30:02.996Z  BAC     N      24.72       100     -1    24.72
35   24.74      420  24.73    24.73
    5 | 2017-09-19T14:30:03.006Z  BAC     T      24.725      200      1    24.72
9   24.73        4  24.72    24.725
    6 | 2017-09-19T14:30:03.009Z  BAC     B      24.73       100      1    24.72
9   24.73        4  24.725   24.725
    7 | 2017-09-19T14:30:03.026Z  BAC     K      24.73      3500      0    24.72
55   24.74      525  24.73    24.73
    8 | 2017-09-19T14:30:03.032Z  BAC     N      24.73       778      0    24.72
51   24.74      424  24.73    24.73
    9 | 2017-09-19T14:30:03.049Z  BAC     T      24.72      1645      1    24.71
68   24.72      137  24.73    24.715
   10 | 2017-09-19T14:30:03.051Z  BAC     N      24.71       800     -1    24.7
481   24.72       31  24.72    24.71
   11 | 2017-09-19T14:30:03.052Z  BAC     N      24.71       100     -1    24.71
31   24.72       29  24.71    24.715
   12 | 2017-09-19T14:30:03.069Z  BAC     P      24.71        67     -1    24.71
7   24.72        3  24.71    24.715
   13 | 2017-09-19T14:30:03.095Z  BAC     Z      24.7125     300     -1    24.71
18   24.725       7  24.71    24.7175
   14 | 2017-09-19T14:30:03.111Z  BAC     K      24.71       300     -1    24.71
39   24.72       44  24.7125  24.715
    … | …                        …       …            …        …       …        …
…        …        …        …        …
24455 | 2017-09-19T20:59:59.993Z  BAC     B      24.87       137      1    24.86
3020   24.87     2071  24.8625  24.865
24456 | 2017-09-19T21:00:00.009Z  BAC     P      24.87     28400      1    24.86
3020   24.87     1956  24.87    24.865
24457 | 2017-09-19T21:00:00.013Z  BAC     A      24.86       100     -1    24.86
3020   24.87     1956  24.87    24.865
24458 | 2017-09-19T21:00:00.213Z  BAC     P      24.87     28400      1    24.86
3020   24.87     1956  24.86    24.865
24459 | 2017-09-19T21:00:00.344Z  BAC     T      24.865       99     -1    24.86
3020   24.87     1956  24.87    24.865
[24460 rows x 12 columns]
```

Calcualte Spread, Effective Spread and Realized Spread by EXCHANGES There are multiple ways to measure spreads in realty for different purposes. Moreoever, in order to compare spreads across stocks, it is common to normalized spread based on prices. For example, the spread

of Bitcoin is larger than the spread of AMC, simply because Bitcoin trades at \$50000 per unit whereas AMC trades at a few dollars per unit.
1. Quoted spread: $S_t = \frac{a_t - b_t}{m_t}$, where $m_t = \frac{a_t + b_t}{2}$. This is simple bid-ask spread telling you about the potential cost of trading. 2. Effective spread: $S_t = d_t(p_t - m_t)$, where d_t is the trade direction (1 for buyer-initiated and -1 for seller initiaed). In reality, because of high-frequency traders (remind HFs can cancell orders and post another one before your orders arrive at exhcnages), the actually transaction price can differ from bid and ask prices you see. This effective spread is the actual transaction cost one pays. 3. Realized spread: $S_t = d_t(p_t - m_{t+\Delta})$. Imagine that you bought some shares at t, then the price moves to a new level $m_{t+\Delta}$ because of realization of information or other things, then the actual spread paid can be negative if the news are good and larger if the news are bad. This is the measure more relavent to market makers as this measures how much a market maker for providing liqudity over time t to $t + \Delta$.

In this exercise, please calcuate different spreads, and check some summary statistics for these spreads. Note that for realized spread, using mid-quote in 10 mins 1. calcate correlation of three spreads 2. plot time series of three spreads by hour 3. calcuate mean spreads at the Exchange level

[4]:
```
# Extract BID and OFR values
bid_values = trade_df[:, 'BID'].to_numpy().flatten()
ofr_values = trade_df[:, 'OFR'].to_numpy().flatten()

# Calculate mt (average of BID and OFR)
mt = (bid_values + ofr_values) / 2

# Calculate pt-1
pt_1 = dt.shift(trade_df[:, 'PRICE'])
# Extract PRICE values
pt = trade_df[:, 'PRICE'].to_numpy().flatten()

# Calculate classification based on provided logic
buyer_initiated = (pt > mt) | ((pt == mt) & (pt > pt_1.to_numpy().flatten()))
seller_initiated = (pt < mt) | ((pt == mt) & (pt < pt_1.to_numpy().flatten()))

# Update COND column based on classification
trade_df[:, dt.update(COND=dt.ifelse(buyer_initiated, 1, dt.
 ↪ifelse(seller_initiated, -1, 0)))]

# Calculate Quoted Spread
quoted_spread = (ofr_values - bid_values) / mt

# Calculate Effective Spread
trade_direction = dt.ifelse(buyer_initiated, 1, -1)
effective_spread = trade_direction * (pt - mt)
print(quoted_spread.shape)

# Calculate Realized Spread
trade_df[:, 'date'] = dt.Type.time64
m_t_plus_delta_values = []
```

```python
for i in range(trade_df.nrows):
    temp_trade_df = trade_df.copy()
    m_t_plus_delta_values.append(temp_trade_df[f.date < dt.
 ↪Frame([temp_trade_df[i, 'date'] + timedelta(minutes=10)]), :][-1, 'mt'])
m_t_plus_delta_values = np.array(m_t_plus_delta_values)

realized_spread = trade_direction * (pt - m_t_plus_delta_values)
trade_df[:, dt.update(m_t_plus_delta_values=m_t_plus_delta_values)]

# Add new columns to datatable
trade_df[:, dt.update(quoted_spread=quoted_spread,␣
 ↪effective_spread=effective_spread, realized_spread=realized_spread)]

# Print the updated datatable
print(trade_df)
```

```
(24460,)
      | date                     SYMBOL  EX       PRICE    SIZE   COND      BID
BIDSIZ      OFR  OFRSIZ  …       mt  m_t_plus_delta_values  quoted_spread
effective_spread  realized_spread
      | time64                   str32   str32  float64   int32  int32  float64
int32  float64   int32    float64                float64        float64
float64          float64
----- + -----------------------  ------  -----  -------  ------  -----  -------
------  -------  ------    -------  ---------------------  -------------
----------------  ---------------
    0 | 2017-09-19T14:30:00.009  BAC     P      24.7      20000     -1    24.72
6   24.74      100  …  24.73                 24.745    0.000808734
0.03             0.045
    1 | 2017-09-19T14:30:01.361  BAC     T      24.71       300     -1    24.72
6   24.74      100  …  24.73                 24.745    0.000808734
0.02             0.035
    2 | 2017-09-19T14:30:01.525  BAC     P      24.7        100     -1    24.72
6   24.74      100  …  24.73                 24.745    0.000808734
0.03             0.045
    3 | 2017-09-19T14:30:02.99   BAC     N      24.73    429950      1    24.72
6   24.74      100  …  24.73                 24.74     0.000808734
0               -0.01
    4 | 2017-09-19T14:30:02.996  BAC     N      24.72       100     -1    24.72
35   24.74      420  …  24.73                 24.74     0.000808734
0.01             0.02
    5 | 2017-09-19T14:30:03.006  BAC     T      24.725      200      1    24.72
9   24.73        4  …  24.725                24.74     0.000404449
0               -0.015
    6 | 2017-09-19T14:30:03.009  BAC     B      24.73       100      1    24.72
9   24.73        4  …  24.725                24.74     0.000404449
0.005           -0.01
```

```
    7 | 2017-09-19T14:30:03.026  BAC     K       24.73      3500      0     24.72
55   24.74      525   …  24.73                 24.74     0.000808734
-0                0.01
    8 | 2017-09-19T14:30:03.032  BAC     N       24.73       778      0     24.72
51   24.74      424   …  24.73                 24.74     0.000808734
-0                0.01
    9 | 2017-09-19T14:30:03.049  BAC     T       24.72      1645      1     24.71
68   24.72      137   …  24.715                24.74     0.000404613
0.005          -0.02
   10 | 2017-09-19T14:30:03.051  BAC     N       24.71       800     -1     24.7
481   24.72       31   …  24.71                 24.74     0.000809389
-0                0.03
   11 | 2017-09-19T14:30:03.052  BAC     N       24.71       100     -1     24.71
31   24.72       29   …  24.715                24.74     0.000404613
0.005            0.03
   12 | 2017-09-19T14:30:03.069  BAC     P       24.71        67     -1     24.71
7   24.72        3  …  24.715                24.74     0.000404613
0.005            0.03
   13 | 2017-09-19T14:30:03.095  BAC     Z       24.7125     300     -1     24.71
18   24.725       7   …  24.7175               24.74     0.000606857
0.005            0.0275
   14 | 2017-09-19T14:30:03.111  BAC     K       24.71       300     -1     24.71
39   24.72       44   …  24.715                24.74     0.000404613
0.005            0.03
    … | …                       …       …            …       …      …        …
…        …       …  …       …                 …              …
…                …
24455 | 2017-09-19T20:59:59.993  BAC     B       24.87       137      1     24.86
3020   24.87     2071   …  24.865                24.865    0.000402172
0.005            0.005
24456 | 2017-09-19T21:00:00.009  BAC     P       24.87     28400      1     24.86
3020   24.87     1956   …  24.865                24.865    0.000402172
0.005            0.005
24457 | 2017-09-19T21:00:00.013  BAC     A       24.86       100     -1     24.86
3020   24.87     1956   …  24.865                24.865    0.000402172
0.005            0.005
24458 | 2017-09-19T21:00:00.213  BAC     P       24.87     28400      1     24.86
3020   24.87     1956   …  24.865                24.865    0.000402172
0.005            0.005
24459 | 2017-09-19T21:00:00.344  BAC     T       24.865      99     -1     24.86
3020   24.87     1956   …  24.865                24.865    0.000402172
-0               -0
[24460 rows x 16 columns]
```

[5]:
```
quoted_spread = trade_df[:, 'quoted_spread'].to_numpy().flatten()
effective_spread = trade_df[:, 'effective_spread'].to_numpy().flatten()
```

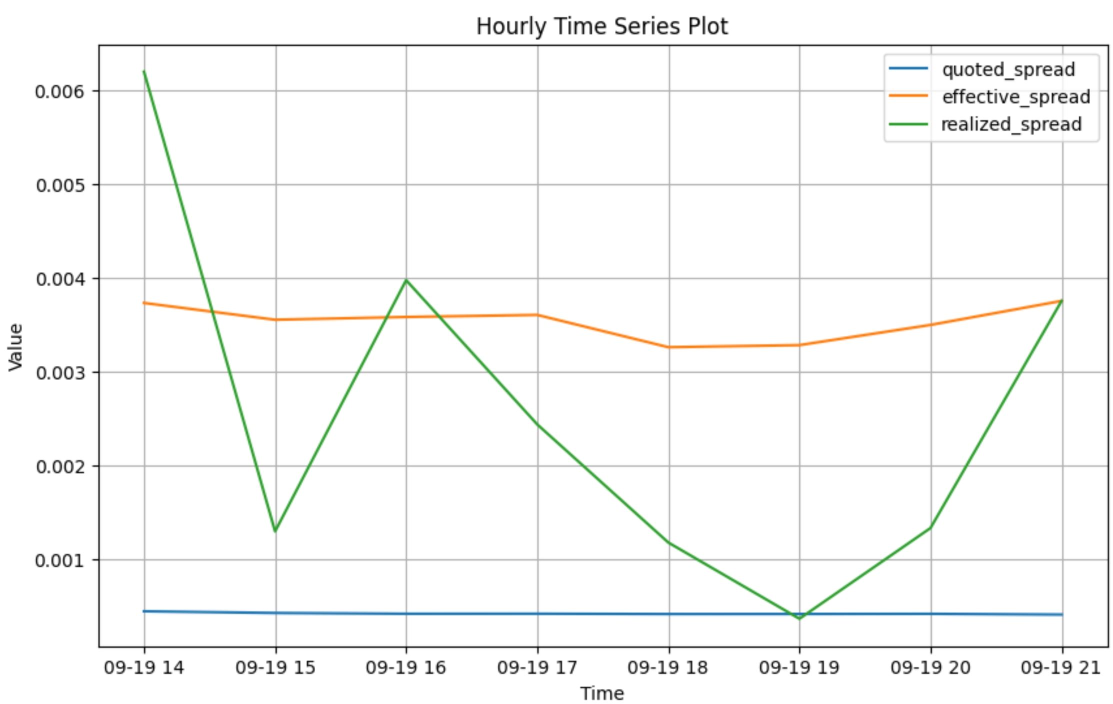
```
realized_spread = trade_df[:, 'realized_spread'].to_numpy().flatten()

# Calculate correlation of three spreads
corr_quoted_effective = np.corrcoef(quoted_spread, effective_spread)[0, 1]
corr_quoted_realized = np.corrcoef(quoted_spread, realized_spread)[0, 1]
corr_effective_realized = np.corrcoef(effective_spread, realized_spread)[0, 1]

print("Correlation of Quoted Spread and Effective Spread:",␣
 ↪corr_quoted_effective)
print("Correlation of Quoted Spread and Realized Spread:", corr_quoted_realized)
print("Correlation of Effective Spread and Realized Spread:",␣
 ↪corr_effective_realized)
```

```
Correlation of Quoted Spread and Effective Spread: -0.2493499795778548
Correlation of Quoted Spread and Realized Spread: 0.03545134380359204
Correlation of Effective Spread and Realized Spread: -0.00302885949409206
```

[6]:
```
# Convert datatable DataFrame to pandas DataFrame
df_pandas = trade_df.to_pandas()

import pandas as pd
# Convert index to datetime format
df_pandas['date'] = pd.to_datetime(df_pandas.date)
df_pandas.set_index('date', inplace=True)
df_pandas = df_pandas[['COND', 'quoted_spread', 'effective_spread',␣
 ↪'realized_spread']]
print(df_pandas.head(-10))
# Resample data by hour and calculate mean
df_hourly = df_pandas.resample('H').mean()

# Plot data
plt.figure(figsize=(10, 6))
for column in df_hourly.columns[-3:]:
    plt.plot(df_hourly.index, df_hourly[column], label=column)

plt.xlabel("Time")
plt.ylabel("Value")
plt.title("Hourly Time Series Plot")
plt.legend()
plt.grid(True)
plt.show()
```

```
                         COND  quoted_spread  effective_spread  \
date
2017-09-19 14:30:00.009    -1       0.000809             0.030
2017-09-19 14:30:01.361    -1       0.000809             0.020
2017-09-19 14:30:01.525    -1       0.000809             0.030
2017-09-19 14:30:02.990     1       0.000809             0.000
```

```
2017-09-19 14:30:02.996    -1        0.000809             0.010
…                        …            …               …
2017-09-19 20:59:58.769    -1        0.000402             0.005
2017-09-19 20:59:59.002    -1        0.000402             0.005
2017-09-19 20:59:59.005    -1        0.000402             0.005
2017-09-19 20:59:59.072    -1        0.000402             0.005
2017-09-19 20:59:59.243    -1        0.000402             0.005

                         realized_spread
date
2017-09-19 14:30:00.009            0.045
2017-09-19 14:30:01.361            0.035
2017-09-19 14:30:01.525            0.045
2017-09-19 14:30:02.990           -0.010
2017-09-19 14:30:02.996            0.020
…                                  …
2017-09-19 20:59:58.769            0.005
2017-09-19 20:59:59.002            0.005
2017-09-19 20:59:59.005            0.005
2017-09-19 20:59:59.072            0.005
2017-09-19 20:59:59.243            0.005

[24450 rows x 4 columns]
```



[7]:
```
df_pandas = trade_df.to_pandas()
# Convert index to datetime format
df_pandas['date'] = pd.to_datetime(df_pandas.date)
df_pandas.set_index('date', inplace=True)
df_pandas = df_pandas[['COND', 'EX', 'quoted_spread', 'effective_spread',␣
↪'realized_spread']]

average_spread = df_pandas.groupby('EX').agg({'quoted_spread': 'mean',␣
↪'effective_spread': 'mean', 'realized_spread': 'mean'})

print(average_spread)
```

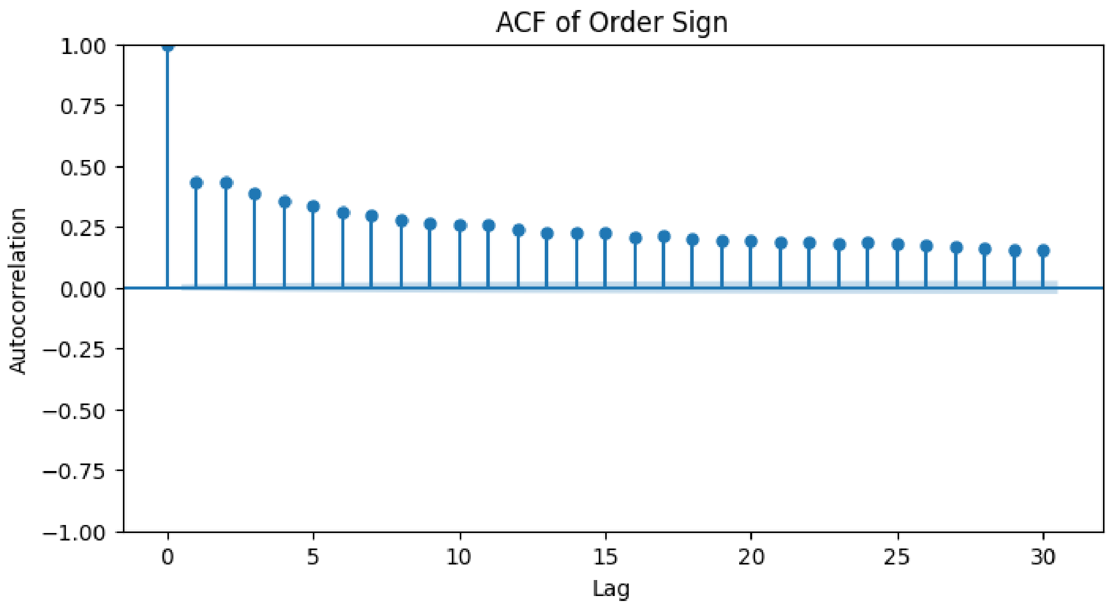
```
    quoted_spread  effective_spread  realized_spread
EX
A        0.000448          0.004352         0.031574
B        0.000405          0.003554         0.001162
J        0.000404          0.004569         0.001122
K        0.000460          0.002747         0.004042
M        0.000462          0.004259         0.008580
N        0.000418          0.002902         0.003492
P        0.000434          0.003052         0.000679
T        0.000442          0.002540         0.004959
V        0.000410          0.002898         0.001376
X        0.000420          0.003613        -0.006053
Y        0.000404          0.004392         0.002055
Z        0.000431          0.002054         0.002703
```

Order Sign Correlation As discussed in the lecture, order spliting is common for informed investors to minimize their price impact. How to empirically check this? One possibility is to examine auto-correlation of orders. With Lee and Ready algorithm, we have a sense how liquidity demanders (informed investors) trade. We start with some simple analysis to check how signed orders are correlated, and then check how to better fit the data to predict sign of next orders. Intuitively, the market makers have a good model to do so, they can 1) front-run investors to profit more, 2) adjust bid-ask prices and market depth to avoid being adversly selected by informed investors.

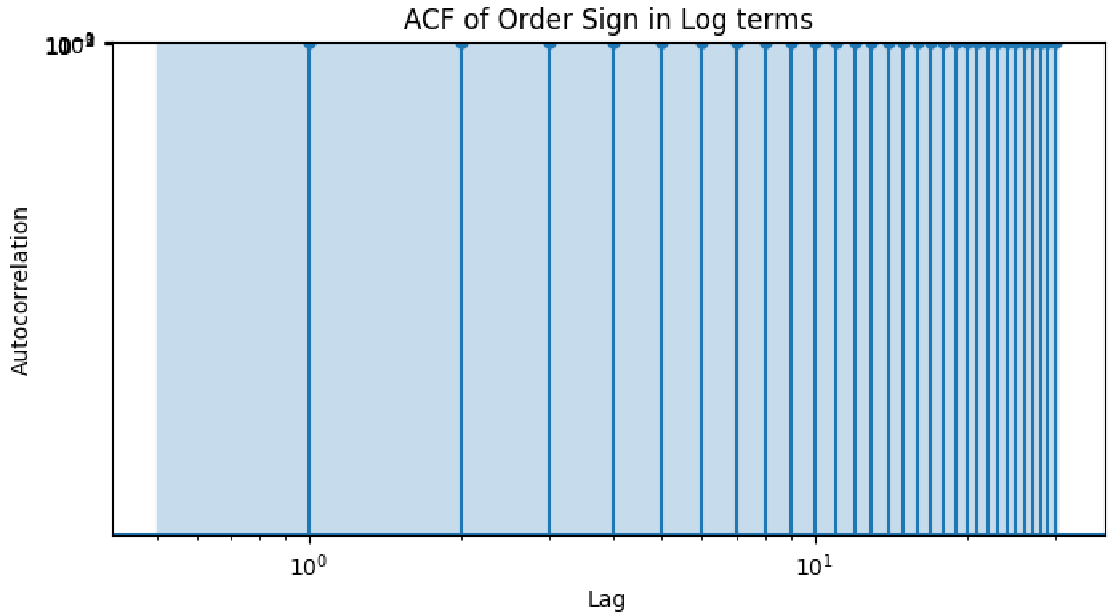
1. autocorrelation plot of order sign
2. re-produce the above figure in log term (both x-axis and y-axis are in log term)
3. fit regressions to check whehter past information can predict future order signs.

[8]:
```
from statsmodels.graphics.tsaplots import plot_acf

# Plot ACF
plt.figure(figsize=(8, 4))
plot_acf(df_pandas['COND'], lags=30, ax=plt.gca())
plt.xlabel('Lag')
plt.ylabel('Autocorrelation')
plt.title('ACF of Order Sign')
```

```
plt.show()
```



[9]:
```
# Plot ACF
plt.figure(figsize=(8, 4))
plot_acf(df_pandas['COND'], lags=30, ax=plt.gca())
plt.xscale('log')
plt.yscale('log')
plt.xlabel('Lag')
plt.ylabel('Autocorrelation')
plt.title('ACF of Order Sign in Log terms')
plt.show()
```




```
[10]: import statsmodels.api as sm
df = df_pandas[['COND']]
for i in range(1, 16):    # based on the ACF plot we took 15 lags
    df[f'lag_{i}'] = df['COND'].shift(i)

# Drop rows with missing values introduced by shifting
df.dropna(inplace=True)

# Split data into features (X) and target (y)
X = df.iloc[:, -15:]  # Use lagged features
y = df['COND']

# Add constant to the features
X = sm.add_constant(X)

# Fit OLS (Ordinary Least Squares) regression model
model = sm.OLS(y, X).fit()

# Print summary statistics
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                   COND   R-squared:                       0.304
Model:                            OLS   Adj. R-squared:                  0.303
Method:                 Least Squares   F-statistic:                     710.2
Date:                Wed, 21 Feb 2024   Prob (F-statistic):               0.00
```

```
Time:                        00:25:13   Log-Likelihood:                -28603.
No. Observations:               24445   AIC:                         5.724e+04
Df Residuals:                   24429   BIC:                         5.737e+04
Df Model:                          15
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         -0.0036      0.005     -0.724      0.469      -0.013       0.006
lag_1          0.2119      0.006     33.127      0.000       0.199       0.224
lag_2          0.1945      0.007     29.759      0.000       0.182       0.207
lag_3          0.1075      0.007     16.151      0.000       0.094       0.121
lag_4          0.0576      0.007      8.604      0.000       0.044       0.071
lag_5          0.0548      0.007      8.175      0.000       0.042       0.068
lag_6          0.0294      0.007      4.387      0.000       0.016       0.043
lag_7          0.0219      0.007      3.266      0.001       0.009       0.035
lag_8          0.0175      0.007      2.608      0.009       0.004       0.031
lag_9          0.0108      0.007      1.604      0.109      -0.002       0.024
lag_10         0.0141      0.007      2.103      0.035       0.001       0.027
lag_11         0.0293      0.007      4.372      0.000       0.016       0.042
lag_12         0.0116      0.007      1.737      0.082      -0.001       0.025
lag_13         0.0021      0.007      0.313      0.754      -0.011       0.015
lag_14         0.0157      0.007      2.406      0.016       0.003       0.029
lag_15         0.0274      0.006      4.288      0.000       0.015       0.040
==============================================================================
Omnibus:                      173.081   Durbin-Watson:                   2.000
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              118.214
Skew:                          -0.012   Prob(JB):                     2.14e-26
Kurtosis:                       2.660   Cond. No.                         3.35
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
```

```
/var/folders/44/r2pt84y14r968g9_vxmvrlxh0000gn/T/ipykernel_69087/3778300692.py:4
: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df[f'lag_{i}'] = df['COND'].shift(i)
/var/folders/44/r2pt84y14r968g9_vxmvrlxh0000gn/T/ipykernel_69087/3778300692.py:4
: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df[f'lag_{i}'] = df['COND'].shift(i)
/var/folders/44/r2pt84y14r968g9_vxmvrlxh0000gn/T/ipykernel_69087/3778300692.py:4
: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df[f'lag_{i}'] = df['COND'].shift(i)
```

Question 3 Quote data As we discussed in the class, not only transacted orders are informative, orders sitting on the books could potential provide some valuation information. In this exercise, we try to test this idea to see whether inbalance order book can help form some trading signals.

Data BAC_nbbo.csv is order bood data (only the best quotes), with each row one of the price or size at the best bid or ask changes which corresponds to change in the supply or demand. * Calculate order imbalance OFI (keep only Nasdaq exchanges) * Aggregate OFI to second level (take summation)

Order Imbalance Order flow imbalance represents the changes in supply and demand. * Best bid or size at the best bid increase -> increase in demand. * Best bid or size at the best bid decreases -> decrease in demand. * Best ask decreases or size at the best ask increases -> increase in supply. * Best ask increases or size at the best ask decreases -> decrease in supply.

Mathematically we summarise these four effects at from time $n-1$ to n as:

$$e_n = I_{B_n \geq B_{n-1}} q_n - I_{B_n \leq B_{n-1}} q_{n-1} - I_{A_n \leq A_{n-1}} q_n + I_{A_n \geq A_{n-1}} q_{n-1}$$

where B_n is the beset Bid price at time n and q_n is the size at those prices, and I is an indicator function. For exampel, $I_{B_n \geq B_{n-1}} = 1$ if $B_n \geq B_{n-1}$ and 0, otherwise.

[17]:
```
# Load the CSV file into a DataFrame
df = pd.read_csv("./BAC_nbbo.csv")
```

```
/var/folders/44/r2pt84y14r968g9_vxmvrlxh0000gn/T/ipykernel_69087/1948102664.py:2
: DtypeWarning: Columns (13) have mixed types. Specify dtype option on import or
set low_memory=False.
  df = pd.read_csv("./BAC_nbbo.csv")
```

[18]:
```
# Creat second stamp
df['TIME_M'] = pd.to_datetime(df['TIME_M'])
```

```
/var/folders/44/r2pt84y14r968g9_vxmvrlxh0000gn/T/ipykernel_69087/3692599462.py:2
: UserWarning: Could not infer format, so each element will be parsed
individually, falling back to `dateutil`. To ensure parsing is consistent and
as-expected, please specify a format.
  df['TIME_M'] = pd.to_datetime(df['TIME_M'])
```

[25]:
```python
## only keep trading hours
df.set_index('TIME_M', inplace=True)
```

[27]:
```python
# notice the extreme values in BID and ASK!!
# need to clean data
# first, remove negative spreads
# then outlier quotes
df['Spread'] = df['ASK'] - df['BID']
df = df[df['Spread'] >= 0]

Q1 = df[['BID', 'ASK']].quantile(0.25)
Q3 = df[['BID', 'ASK']].quantile(0.75)
IQR = Q3 - Q1

outliers_column1 = ((df['BID'] < (Q1['BID'] - 1.5 * IQR['BID'])) |
                     (df['BID'] > (Q3['BID'] + 1.5 * IQR['BID'])))
outliers_column2 = ((df['ASK'] < (Q1['ASK'] - 1.5 * IQR['ASK'])) |
                     (df['ASK'] > (Q3['ASK'] + 1.5 * IQR['ASK'])))

df.loc[outliers_column1, 'BID'] = np.nan
df.loc[outliers_column2, 'ASK'] = np.nan

df = df.dropna(subset=['BID', 'ASK'])
df = df[df['EX'] == 'N']
```

[28]:
```python
df['increase demand indicator'] = 0  # Initialize the column with zeros
df['decrease demand indicator'] = 0  # Initialize the column with zeros
df['increase supply indicator'] = 0  # Initialize the column with zeros
df['decrease supply indicator'] = 0  # Initialize the column with zeros

#previous value columns
df['bid_shifted'] = df['BID'].shift(1)
df['ask_shifted'] = df['ASK'].shift(1)
df['bidsize_shifted'] = df['BIDSIZ'].shift(1)
df['asksize_shifted'] = df['ASKSIZ'].shift(1)




# Apply conditions
df.loc[(df['BID'] > df['bid_shifted']) | (df['BIDSIZ'] >␣
 ↪df['bidsize_shifted']), 'increase demand indicator'] = 1
df.loc[(df['BID'] < df['bid_shifted']) | (df['BIDSIZ'] <␣
 ↪df['bidsize_shifted']), 'decrease demand indicator'] = 1
df.loc[(df['ASK'] < df['ask_shifted']) | (df['ASKSIZ'] >␣
 ↪df['asksize_shifted']), 'increase supply indicator'] = 1
df.loc[(df['ASK'] > df['ask_shifted']) | (df['ASKSIZ'] <␣
 ↪df['asksize_shifted']), 'decrease supply indicator'] = 1
```

```
# OFI column
df['OFI'] = (df['increase demand indicator'] * df['BIDSIZ']) - (df['decrease␣
↪demand indicator'] * df['bidsize_shifted'])- (df['increase supply␣
↪indicator'] * df['ASKSIZ'])+ (df['decrease supply indicator'] *␣
↪df['asksize_shifted'])
df['Mid price'] = 0.5 * (df['BID'] + df['ASK'])
```

[37]:
```
# Aggregate by second
# Construct return as log difference of last mid price and first mid price of␣
↪each second
# Resample to get the first and last price of every second
# print(df.head())
# sub_df = df[['Mid price', 'OFI']]
first_prices = df['Mid price'].resample('1S').first()
last_prices = df['Mid price'].resample('1S').last()
# Calculate log returns
log_returns = np.log(last_prices / first_prices)

total_OFI_by_sec = df['OFI'].resample('1S').sum()

merged_df = pd.DataFrame({
    'log_returns': log_returns,
    'total_OFI': total_OFI_by_sec
})

# Drop NaN values that may result from resampling
merged_df.dropna(inplace=True)
```

Using OFI to generate trading signal: first do train/test split by selecting the first 70% of the data

[38]:
```
# Test whether OFI can explain return variations in train data
split_point = int(len(merged_df) * 0.7)
train_df = merged_df.iloc[:split_point]
test_df = merged_df.iloc[split_point:]

# Adding a constant for the intercept
X_train = sm.add_constant(train_df['total_OFI'])
y_train = train_df['log_returns']

# Fit the model
model = sm.OLS(y_train, X_train).fit()

# Check the summary for R-squared value
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
```

```
Dep. Variable:            log_returns   R-squared:                       0.182
Model:                            OLS   Adj. R-squared:                  0.182
Method:                 Least Squares   F-statistic:                     3136.
Date:                Wed, 21 Feb 2024   Prob (F-statistic):               0.00
Time:                        00:39:37   Log-Likelihood:             1.1449e+05
No. Observations:               14080   AIC:                        -2.290e+05
Df Residuals:                   14078   BIC:                        -2.290e+05
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const       5.578e-07      6e-07      0.930      0.353   -6.18e-07    1.73e-06
total_OFI   1.555e-07   2.78e-09     55.997      0.000     1.5e-07    1.61e-07
==============================================================================
Omnibus:                     4891.075   Durbin-Watson:                   2.185
Prob(Omnibus):                  0.000   Jarque-Bera (JB):          1496400.070
Skew:                           0.300   Prob(JB):                         0.00
Kurtosis:                      53.501   Cond. No.                         216.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
```

[ ]:
```python
# explainatry power for test sample
```

Construct a Predictive Trading Signal BUT! The above analysis is in-sample. We want to see out-sample results.

[39]:
```python
# calculate one-second ahead return
merged_df['lagged_OFI'] = merged_df['total_OFI'].shift(1)

# Drop NaN values that might have been introduced by shifting
merged_df.dropna(inplace=True)
```

[40]:
```python
# Test whether OFI can explain FUTURE return variations
# Split sample to test and train samples again
# Test whether lagged OFI can predict FUTURE return
split_point = int(len(merged_df) * 0.7)
train_df = merged_df.iloc[:split_point]
test_df = merged_df.iloc[split_point:]

# Fit the model on the training set
X_train = sm.add_constant(train_df['lagged_OFI'])
y_train = train_df['log_returns']
```

```python
model = sm.OLS(y_train, X_train).fit()
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:            log_returns   R-squared:                       0.003
Model:                            OLS   Adj. R-squared:                  0.003
Method:                 Least Squares   F-statistic:                     39.14
Date:                Wed, 21 Feb 2024   Prob (F-statistic):           4.06e-10
Time:                        00:40:45   Log-Likelihood:             1.1309e+05
No. Observations:               14079   AIC:                        -2.262e+05
Df Residuals:                   14077   BIC:                        -2.262e+05
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const       2.553e-07   6.62e-07      0.385      0.700   -1.04e-06    1.55e-06
lagged_OFI  1.918e-08   3.07e-09      6.256      0.000    1.32e-08    2.52e-08
==============================================================================
Omnibus:                     4480.465   Durbin-Watson:                   2.191
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           920820.609
Skew:                           0.217   Prob(JB):                         0.00
Kurtosis:                      42.617   Cond. No.                         216.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
```

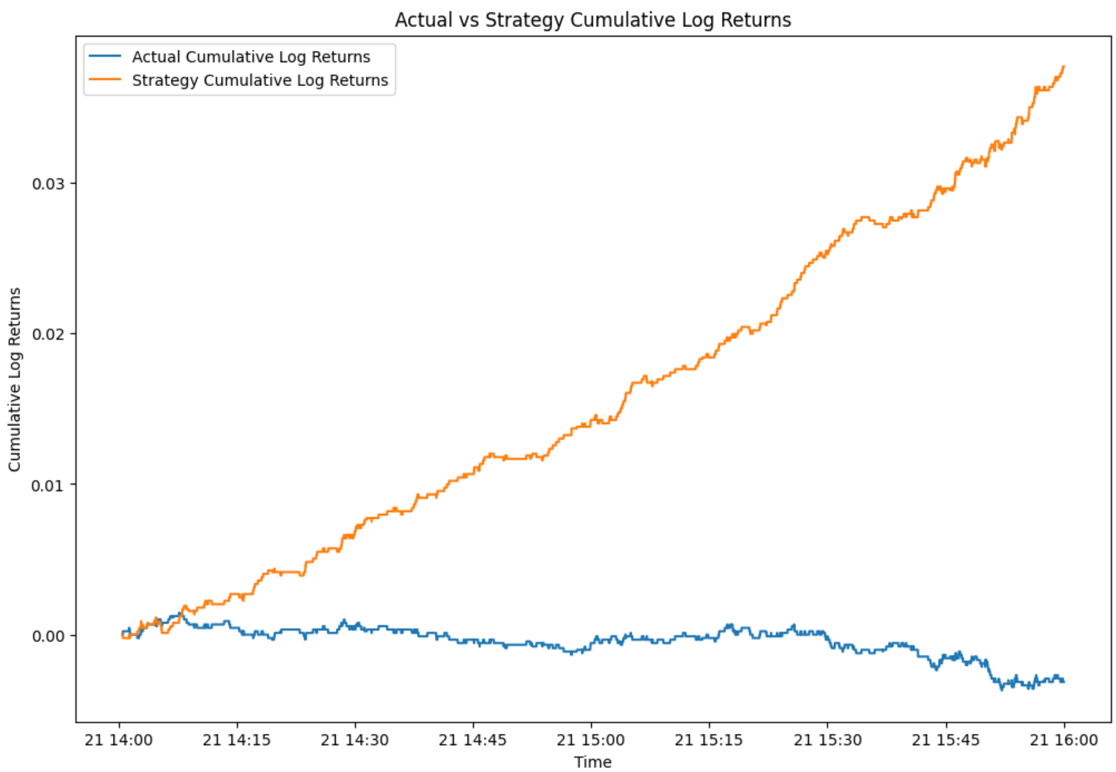
[41]:
```python
# explainatry power for test sample
X_test = sm.add_constant(test_df['lagged_OFI'])
y_test = test_df['log_returns']

# Predict
predictions = model.predict(X_test)

# Calculate and plot cumulative returns from predictions for the test set
test_df['predicted_signal'] = np.where(predictions > 0, 1, -1)
```

```
/var/folders/44/r2pt84y14r968g9_vxmvrlxh0000gn/T/ipykernel_69087/3620246845.py:9
: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  test_df['predicted_signal'] = np.where(predictions > 0, 1, -1)
```

[44]:

```python
# plots cummulative return of the strategy using signal from past OFI

test_df['strategy_returns'] = test_df['predicted_signal'] *␣
↪test_df['log_returns']
test_df['cumulative_strategy_returns'] = test_df['strategy_returns'].cumsum()
test_df['cumulative_actual_returns'] = test_df['log_returns'].cumsum()
# Plot
plt.figure(figsize=(12, 8))
plt.plot(test_df.index, test_df['cumulative_actual_returns'], label='Actual␣
↪Cumulative Log Returns')
plt.plot(test_df.index, test_df['cumulative_strategy_returns'], label='Strategy␣
↪Cumulative Log Returns')
plt.title('Actual vs Strategy Cumulative Log Returns')
plt.xlabel('Time')
plt.ylabel('Cumulative Log Returns')
plt.legend()
plt.show()
```

```
/var/folders/44/r2pt84y14r968g9_vxmvrlxh0000gn/T/ipykernel_69087/1157031162.py:3
: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  test_df['strategy_returns'] = test_df['predicted_signal'] *
test_df['log_returns']
/var/folders/44/r2pt84y14r968g9_vxmvrlxh0000gn/T/ipykernel_69087/1157031162.py:4
: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  test_df['cumulative_strategy_returns'] = test_df['strategy_returns'].cumsum()
/var/folders/44/r2pt84y14r968g9_vxmvrlxh0000gn/T/ipykernel_69087/1157031162.py:5
: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  test_df['cumulative_actual_returns'] = test_df['log_returns'].cumsum()
```


Actual vs Strategy Cumulative Log Returns
Actual Cumulative Log Returns
Strategy Cumulative Log Returns
Cumulative Log Returns
0.03
0.02
0.01
0.00
21 14:00
21 14:15
21 14:30
21 14:45
21 15:00
21 15:15
21 15:30
21 15:45
21 16:00
Time

```
[ ]:
```