Pairs Trading - Cointegration

1. Importing the S&P 500 Stock Data

a. Loading the Necessary Libraries

In [1]:
```python
import os
import itertools
import numpy as np
import pandas as pd
import statsmodels.api as sm
import matplotlib.pyplot as plt
from statsmodels.tsa.stattools import coint
from statsmodels.regression.linear_model import OLS
from scipy.cluster.hierarchy import linkage, dendrogram, fcluster, inconsistent
```

b. Changing Working Directory

In [2]:
```python
# Changing my working directory
os.chdir("C:/Users/jeswi/OneDrive/Desktop/Quantitative Research/Pairs trading - Cointegration")

# Printing my working directory
print("Current Working Directory:", os.getcwd())
```

```
Current Working Directory: C:\Users\jeswi\OneDrive\Desktop\Quantitative Research\Pairs trading - Cointegration
```

c. Loading the Data

In [3]:
```python
data = pd.read_csv("S&P_stock_data.csv")

# Previewing the data
print(data.head())
```

```
         Date         MMM        AOS         ABT        ABBV         ACN  \
0  19-01-2022  128.609131  74.333565  117.356979  118.657448  324.744202
1  20-01-2022  125.071053  73.081429  116.109398  116.744286  323.752289
2  21-01-2022  124.408150  71.782555  117.152153  115.822830  317.082398
3  24-01-2022  124.516235  73.614052  115.280769  115.980797  320.266144
4  25-01-2022  125.200806  70.455696  114.768692  116.796951  316.440033

         ADBE         AMD        AES        AFL  ...        WMB         WTW  \
0  516.580017  128.270004  19.911016  57.549511  ...  24.374533  215.411179
1  510.850006  121.889999  20.187204  56.760910  ...  24.156227  217.008545
2  499.910004  118.809998  19.807451  55.807259  ...  24.147833  214.536453
3  519.659973  116.529999  19.496744  56.375786  ...  23.736416  219.480560
4  502.720001  111.129997  19.134254  56.760910  ...  24.324156  215.496735

         WDAY       WYNN        XEL         XYL         YUM        ZBRA  \
0  249.460007  86.744919  60.221264  103.644325  115.870186  513.109985
1  251.770004  84.835892  60.274292  101.743545  115.051178  501.420013
2  245.690002  83.423576  60.309666  102.287987  115.628197  494.179993
3  247.630005  82.595673  59.611267  102.765572  117.582626  490.559998
4  236.110001  84.212524  60.009083   99.212387  113.589996  474.000000

          ZBH         ZTS
0  115.936348  193.803757
1  115.720016  195.137146
2  115.249779  193.562210
3  116.293739  193.311020
4  113.754364  188.653809

[5 rows x 498 columns]
```

2. Data Pre-Processing

a. Checking for missing values

In [4]:
```python
# Count of missing values in each column
null_counts = data.isnull().sum()

# Display columns with the number of missing values
null_counts = null_counts[null_counts > 0].sort_values(ascending=False)

print(null_counts)
```

```
SOLV    548
dtype: int64
```

The stock 'SOLV' has 548 missing data points. This stock has been dropped from the entire dataset as more than half its data points are missing.

In [5]:
```python
data = data.drop(columns=['SOLV'])
```

In [6]:
```python
print(data.dtypes)
```

```
Date     object
MMM     float64
AOS     float64
ABT     float64
ABBV    float64
         ...
XYL     float64
YUM     float64
ZBRA    float64
ZBH     float64
ZTS     float64
Length: 497, dtype: object
```

b. Formatting 'date' appropriately

In [7]:
```python
data['Date'] = pd.to_datetime(data['Date'], format='%d-%m-%Y', errors='coerce')
data.set_index('Date', inplace=True) #setting 'date' as an index

print(data.head())
```

```
                   MMM        AOS         ABT        ABBV         ACN  \
Date
2022-01-19  128.609131  74.333565  117.356979  118.657448  324.744202
2022-01-20  125.071053  73.081429  116.109398  116.744286  323.752289
2022-01-21  124.408150  71.782555  117.152153  115.822830  317.082398
2022-01-24  124.516235  73.614052  115.280769  115.980797  320.266144
2022-01-25  125.200806  70.455696  114.768692  116.796951  316.440033

                  ADBE         AMD        AES        AFL           A  ...  \
Date                                                                  ...
2022-01-19  516.580017  128.270004  19.911016  57.549511  136.696701  ...
2022-01-20  510.850006  121.889999  20.187204  56.760910  135.771973  ...
2022-01-21  499.910004  118.809998  19.807451  55.807259  133.854355  ...
2022-01-24  519.659973  116.529999  19.496744  56.375786  134.448120  ...
2022-01-25  502.720001  111.129997  19.134254  56.760910  130.992477  ...

                  WMB         WTW        WDAY       WYNN        XEL  \
Date
2022-01-19  24.374533  215.411179  249.460007  86.744919  60.221264
2022-01-20  24.156227  217.008545  251.770004  84.835892  60.274292
2022-01-21  24.147833  214.536453  245.690002  83.423576  60.309666
2022-01-24  23.736416  219.480560  247.630005  82.595673  59.611267
2022-01-25  24.324156  215.496735  236.110001  84.212524  60.009083

                   XYL         YUM        ZBRA         ZBH         ZTS
Date
2022-01-19  103.644325  115.870186  513.109985  115.936348  193.803757
2022-01-20  101.743545  115.051178  501.420013  115.720016  195.137146
2022-01-21  102.287987  115.628197  494.179993  115.249779  193.562210
2022-01-24  102.765572  117.582626  490.559998  116.293739  193.311020
2022-01-25   99.212387  113.589996  474.000000  113.754364  188.653809

[5 rows x 496 columns]
```

c. Converting raw prices to scale-free log-returns

In [8]:
```python
log_returns = np.log(data / data.shift(1)) * 100

# Drop the first row as it is null and rounding up to four decimal points
log_returns = log_returns.dropna().round(4)

# printing the dataset of log-returns
print(log_returns.head())
```

```
               MMM     AOS     ABT    ABBV     ACN    ADBE     AMD     AES  \
Date
2022-01-20 -2.7896 -1.6988 -1.0688 -1.6255 -0.3059 -1.1154 -5.1018  1.3776
2022-01-21 -0.5314 -1.7933  0.8941 -0.7924 -2.0817 -2.1648 -2.5593 -1.8991
2022-01-24  0.0868  2.5194 -1.6103  0.1363  0.9991  3.8747 -1.9377 -1.5811
2022-01-25  0.5483 -4.3852 -0.4452  0.7012 -1.2019 -3.3141 -4.7448 -1.8767
2022-01-26 -2.5945 -0.9060 -2.6302  0.8529 -1.5222 -0.3807 -0.3787  1.1213

               AFL       A  ...     WMB     WTW    WDAY    WYNN     XEL  \
Date                        ...
2022-01-20 -1.3798 -0.6788  ... -0.8997  0.7388  0.9217 -2.2253  0.0880
2022-01-21 -1.6944 -1.4225  ... -0.0348 -1.1457 -2.4445 -1.6788  0.0587
2022-01-24  1.0136  0.4426  ... -1.7184  2.2784  0.7865 -0.9974 -1.1648
2022-01-25  0.6808 -2.6039  ...  2.4460 -1.8318 -4.7638  1.9386  0.6651
2022-01-26  0.4192 -0.7908  ...  0.8250 -0.4865 -4.8020 -0.3824 -0.3837

               XYL     YUM    ZBRA     ZBH     ZTS
Date
2022-01-20 -1.8510 -0.7093 -2.3046 -0.1868  0.6857
2022-01-21  0.5337  0.5003 -1.4544 -0.4072 -0.8104
2022-01-24  0.4658  1.6761 -0.7352  0.9017 -0.1299
2022-01-25 -3.5188 -3.4546 -3.4340 -2.2078 -2.4387
2022-01-26 -1.2887 -1.5773  0.3580 -0.3810 -2.8099

[5 rows x 496 columns]
```

3. Cluster Analysis

a. Computing correlation based distance matrix

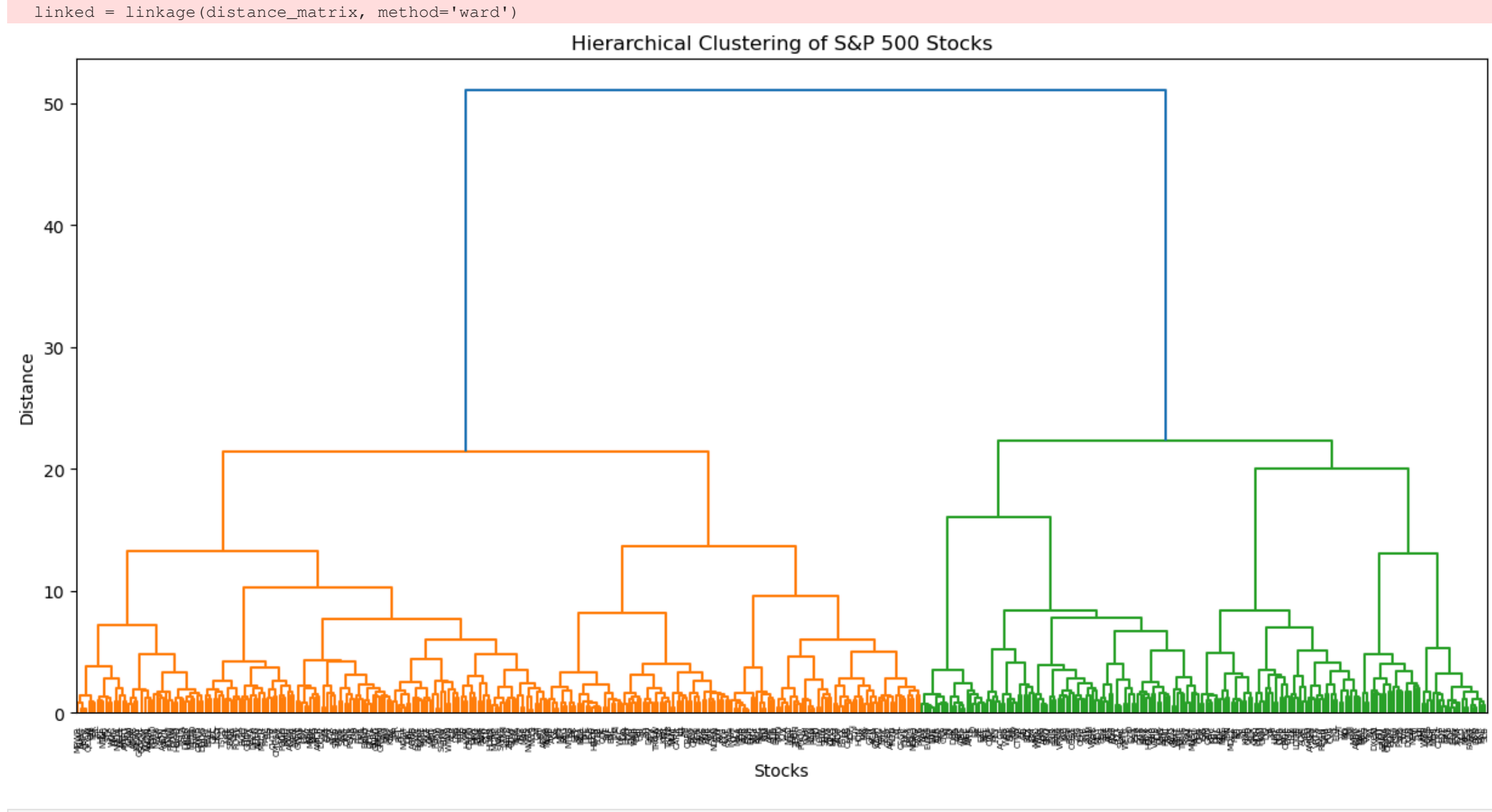
In [9]:
```python
# Correlation matrix of stocks
corr_matrix = log_returns.corr()

# Convert correlation to distance for clustering
distance_matrix = 1 - corr_matrix

print(distance_matrix)
```

```
           MMM       AOS       ABT      ABBV       ACN      ADBE       AMD  \
MMM   0.000000  0.534278  0.712933  0.751459  0.618890  0.706137  0.693560
AOS   0.534278  0.000000  0.703753  0.782629  0.570781  0.655509  0.653458
ABT   0.712933  0.703753  0.000000  0.662106  0.613704  0.690275  0.799555
ABBV  0.751459  0.782629  0.662106  0.000000  0.814740  0.893400  0.964565
ACN   0.618890  0.570781  0.613704  0.814740  0.000000  0.467642  0.561409
...        ...       ...       ...       ...       ...       ...       ...
XYL   0.529711  0.435295  0.604930  0.778335  0.517357  0.607826  0.585906
YUM   0.709044  0.603700  0.605195  0.736797  0.622489  0.687291  0.692474
ZBRA  0.558179  0.534350  0.706231  0.878658  0.518975  0.600306  0.512939
ZBH   0.694936  0.640814  0.513738  0.705764  0.652337  0.727047  0.768720
ZTS   0.667458  0.628006  0.525717  0.705678  0.539466  0.629863  0.733607

           AES       AFL         A  ...       WMB       WTW      WDAY  \
MMM   0.684230  0.622526  0.601839  ...  0.684574  0.647170  0.768600
AOS   0.653694  0.615478  0.530277  ...  0.771594  0.579120  0.683700
ABT   0.742096  0.637214  0.574443  ...  0.780693  0.589838  0.719510
ABBV  0.810710  0.746838  0.702579  ...  0.798528  0.724190  0.890777
ACN   0.707094  0.630794  0.508951  ...  0.764338  0.605173  0.465669
...        ...       ...       ...  ...       ...       ...       ...
XYL   0.592414  0.554085  0.493720  ...  0.693530  0.557928  0.610643
YUM   0.655565  0.621194  0.602011  ...  0.783192  0.603727  0.699335
ZBRA  0.620898  0.737662  0.497291  ...  0.690363  0.687493  0.586618
ZBH   0.658536  0.668946  0.596546  ...  0.796473  0.657734  0.706693
ZTS   0.660590  0.684576  0.474935  ...  0.779625  0.651171  0.619209

          WYNN       XEL       XYL       YUM      ZBRA       ZBH       ZTS
MMM   0.681606  0.725198  0.529711  0.709044  0.558179  0.694936  0.667458
AOS   0.616158  0.727703  0.435295  0.603700  0.534350  0.640814  0.628006
ABT   0.777940  0.623706  0.604930  0.605195  0.706231  0.513738  0.525717
ABBV  0.895428  0.730812  0.778335  0.736797  0.878658  0.705764  0.705678
ACN   0.633584  0.731962  0.517357  0.622489  0.518975  0.652337  0.539466
...        ...       ...       ...       ...       ...       ...       ...
XYL   0.611834  0.719511  0.000000  0.585345  0.532710  0.592163  0.581131
YUM   0.655817  0.677293  0.585345  0.000000  0.661804  0.633515  0.596696
ZBRA  0.602967  0.876418  0.532710  0.661804  0.000000  0.655366  0.551425
ZBH   0.715477  0.716622  0.592163  0.633515  0.655366  0.000000  0.607269
ZTS   0.678162  0.709686  0.581131  0.596696  0.551425  0.607269  0.000000

[496 rows x 496 columns]
```

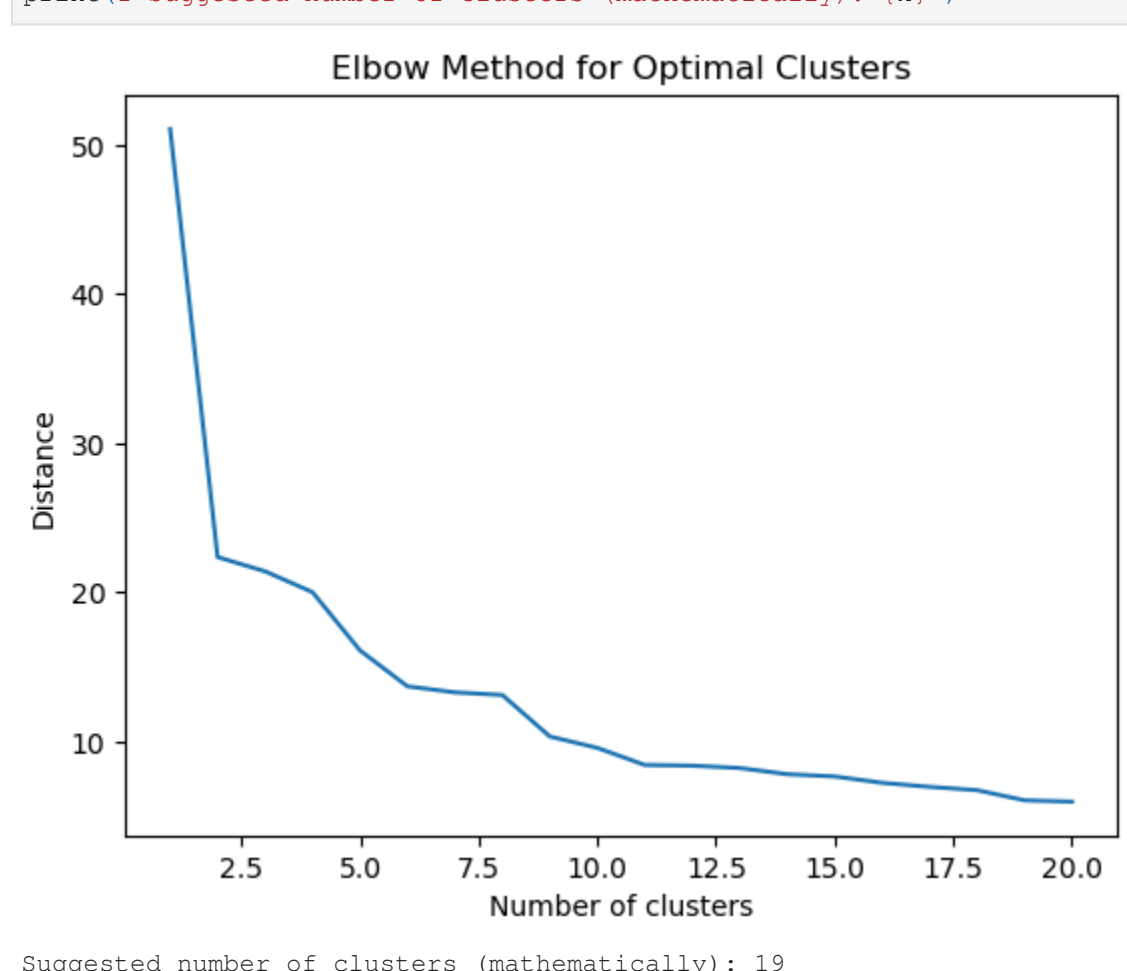
b. Hierarchical cluster analysis

In [10]:
```python
# Perform hierarchical clustering using Ward's method
linked = linkage(distance_matrix, method='ward')

# Plot dendrogram
plt.figure(figsize=(15, 7))
dendrogram(linked, labels=log_returns.columns, leaf_rotation=90)
plt.title('Hierarchical Clustering of S&P 500 Stocks')
plt.xlabel('Stocks')
plt.ylabel('Distance')
plt.show()
```

```
C:\Users\jeswi\AppData\Local\Temp\ipykernel_20648\2126663342.py:2: ClusterWarning: scipy.cluster: The symmetric non-negative hollow observation ma
trix looks suspiciously like an uncondensed distance matrix
  linked = linkage(distance_matrix, method='ward')
```



In [11]:
```python
# Extract distances where clusters merge
last = linked[-20:, 2]  # last 20 merge distances
num_clusters = range(1, 21)

# Compute difference between successive merges
acceleration = np.diff(last, 2)  # 2nd derivative
k = acceleration.argmax() + 2

plt.plot(num_clusters, last[::-1])
plt.title('Elbow Method for Optimal Clusters')
plt.xlabel('Number of clusters')
plt.ylabel('Distance')
plt.show()

print(f"Suggested number of clusters (mathematically): {k}")
```




```
Suggested number of clusters (mathematically): 19
```

In [12]:
```python
num_clusters = 6
cluster_labels = fcluster(linked, num_clusters, criterion='maxclust')

# Map stocks to clusters
clusters = pd.DataFrame({'Stock': log_returns.columns, 'Cluster': cluster_labels})
print(clusters.sort_values('Cluster'))
```

```
    Stock  Cluster
0     MMM        1
412   SNA        1
182  EXPE        1
183  EXPD        1
409  SWKS        1
..    ...      ...
440   TKO        6
116   COP        6
97    CVX        6
77   CHRW        6
163    EA        6

[496 rows x 2 columns]
```

3. Cointegration Test

a. Cluster 1

In [13]:
```python
# Step 1: Select stocks in Cluster 1
cluster1_stocks = clusters[clusters['Cluster'] == 1]['Stock'].tolist()
cluster1_data = log_returns[cluster1_stocks]

# Step 2: Generate all possible pairs
pairs = list(itertools.combinations(cluster1_stocks, 2))

# Step 3: Test cointegration and store results
coint_results = []

for stock1, stock2 in pairs:
    score, pvalue, _ = coint(cluster1_data[stock1], cluster1_data[stock2])
    if pvalue < 0.01:  # filter for significant pairs
        coint_results.append({'Stock1': stock1, 'Stock2': stock2, 'p_value': pvalue})

# Step 4: Convert to DataFrame
coint_df_1 = pd.DataFrame(coint_results)
coint_df_1 = coint_df_1.sort_values('p_value')

print(coint_df_1)
```

```
     Stock1 Stock2   p_value
0       MMM   ADBE  0.000000
8736    FCX    UPS  0.000000
8737    FCX    GWW  0.000000
8738    FCX    DIS  0.000000
8739    FCX    WBD  0.000000
...     ...    ...       ...
5902    CSX   PAYC  0.000023
5919    CSX   SBUX  0.000025
5852    CSX    HAS  0.000025
5872    CSX    LKQ  0.000031
5848    CSX   GNRC  0.000040

[13695 rows x 3 columns]
```

In [14]:
```python
top_50_pairs_1 = coint_df_1.sort_values('p_value').head(50)

print(top_50_pairs_1)
```

```
     Stock1 Stock2  p_value
0       MMM   ADBE      0.0
3594     BA    HST      0.0
3584     BA   GRMN      0.0
3595     BA    HWM      0.0
3597     BA   HUBB      0.0
3598     BA   PODD      0.0
3599     BA   INTC      0.0
3600     BA   IBKR      0.0
3601     BA    IFF      0.0
3602     BA     IP      0.0
3603     BA    IPG      0.0
3604     BA   JBHT      0.0
3605     BA    JBL      0.0
3596     BA    HPQ      0.0
3077   AXON    CRM      0.0
3076   AXON    RCL      0.0
6088   DDOG   FTNT      0.0
2750   APTV    MGM      0.0
3593     BA    HLT      0.0
2751   APTV   MCHP      0.0
3592     BA    HPE      0.0
3590     BA   GDDY      0.0
3620     BA   MSFT      0.0
3621     BA    MHK      0.0
3622     BA   MPWR      0.0
3623     BA   NTAP      0.0
3624     BA   NFLX      0.0
3625     BA   NWSA      0.0
3626     BA    NWS      0.0
3627     BA    NKE      0.0
3628     BA    NSC      0.0
3609     BA    LVS      0.0
3608     BA   LRCX      0.0
3607     BA   KLAC      0.0
3606     BA   KEYS      0.0
3586     BA    GEN      0.0
3587     BA   GNRC      0.0
3588     BA     GM      0.0
3589     BA    GPC      0.0
3591     BA    HAS      0.0
3619     BA     MU      0.0
2752   APTV     MU      0.0
2758   APTV   NWSA      0.0
2724   APTV    HAS      0.0
2725   APTV    HPE      0.0
2726   APTV    HLT      0.0
2727   APTV    HST      0.0
2728   APTV    HWM      0.0
2729   APTV    HPQ      0.0
2780   APTV   HOOD      0.0
```

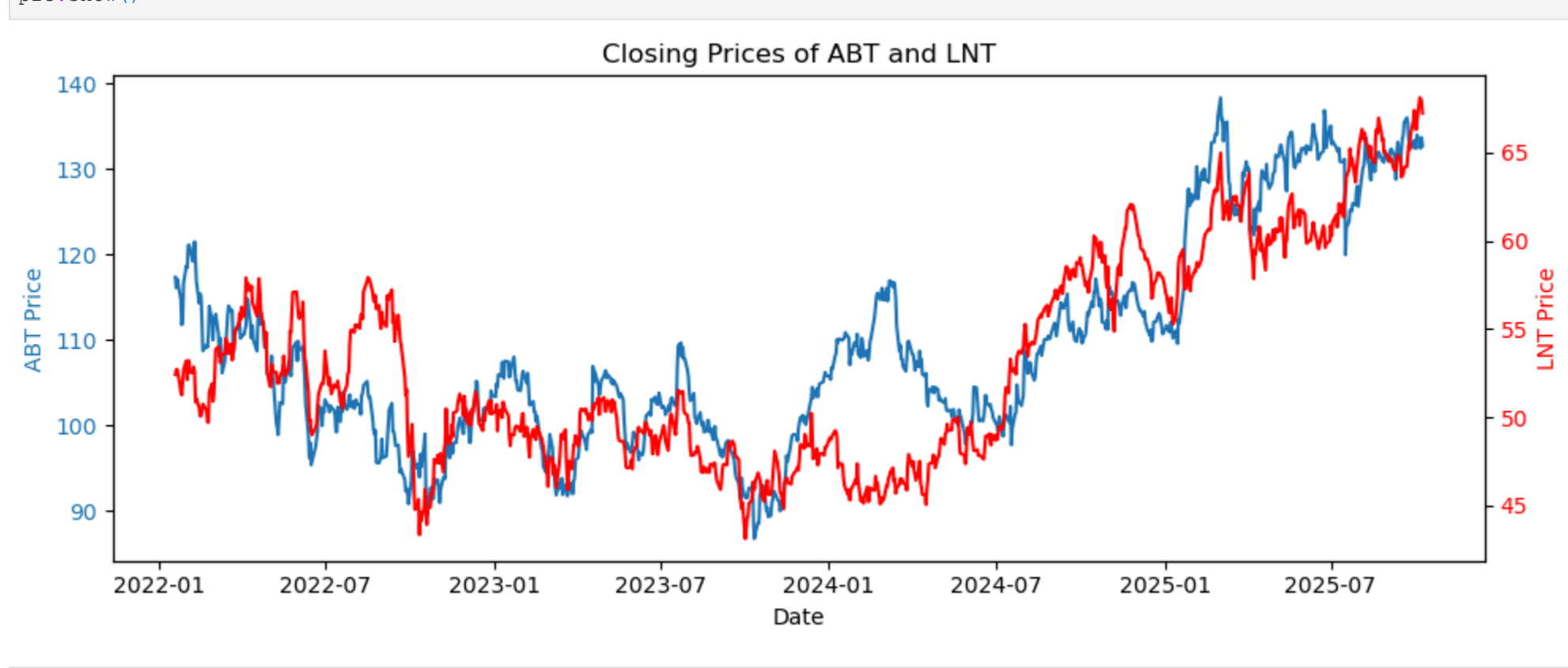
4. Building a Trading Strategy

Basic framework of pairs trading:

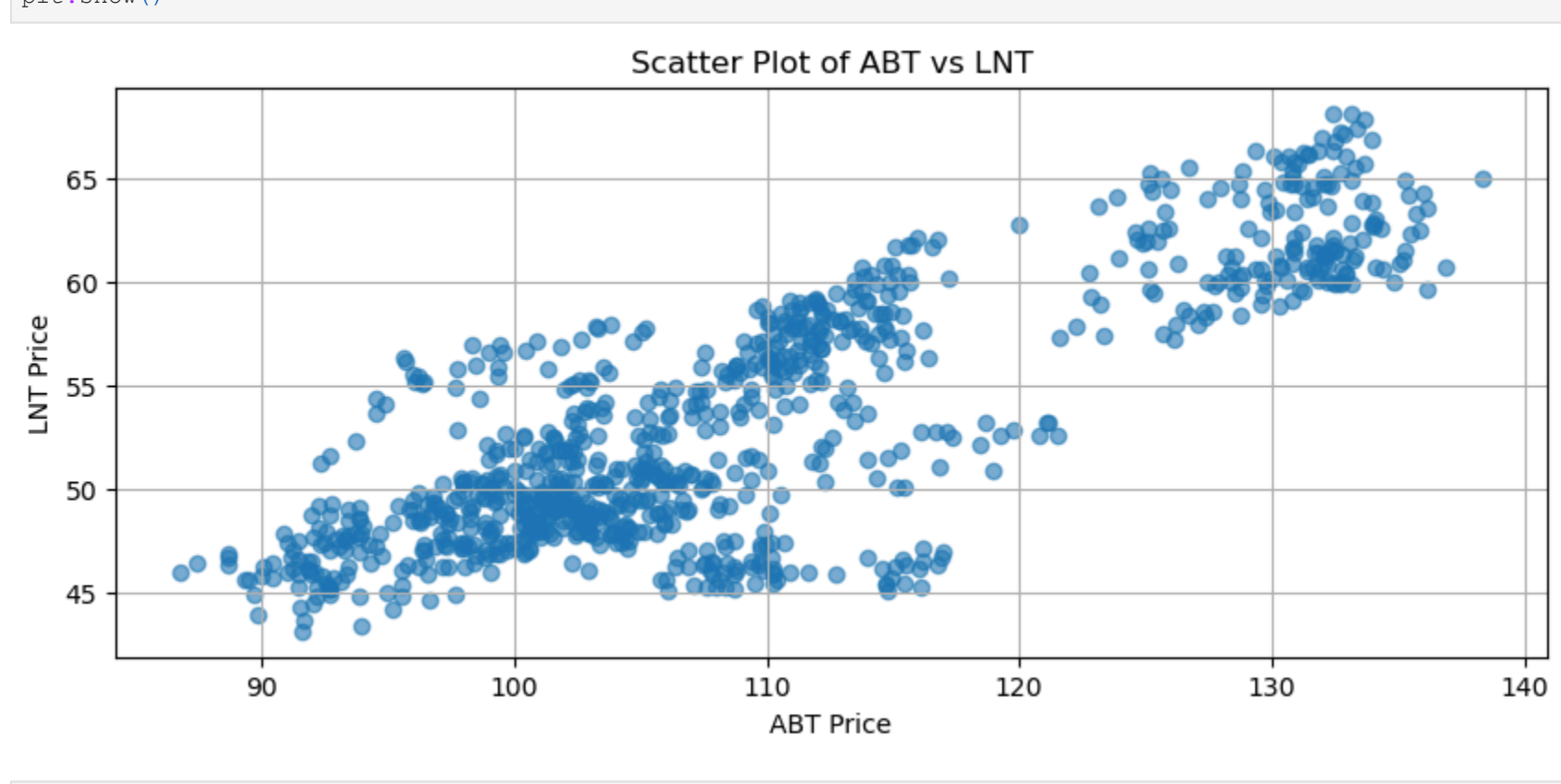
1. Calculate the spread between the cointegrated stocks.
2. Look at the z-score of the spread.
3. Trade when the z-score is above or below a certain threshold.

a. Plotting our cointegrated pair

In [15]:
```python
pair = ['ABT', 'LNT']
pair_data = data[pair]

fig, ax1 = plt.subplots(figsize=(10,4))

# Left y-axis for ABT
color = 'tab:blue'
ax1.set_xlabel('Date')
ax1.set_ylabel('ABT Price', color=color)
ax1.plot(pair_data['ABT'], color=color, label='ABT')
ax1.tick_params(axis='y', labelcolor=color)

# Right y-axis for LNT
ax2 = ax1.twinx()
color = 'red'
ax2.set_ylabel('LNT Price', color=color)
ax2.plot(pair_data['LNT'], color=color, label='LNT')
ax2.tick_params(axis='y', labelcolor=color)

plt.title('Closing Prices of ABT and LNT')
fig.tight_layout()
plt.show()
```



In [16]:
```python
plt.figure(figsize=(10,4))
plt.scatter(pair_data['ABT'], pair_data['LNT'], alpha=0.6)
plt.title('Scatter Plot of ABT vs LNT')
plt.xlabel('ABT Price')
plt.ylabel('LNT Price')
plt.grid(True)
plt.show()
```



In [17]:
```python
correlation = pair_data['ABT'].corr(pair_data['LNT'])
print("Correlation between ABT and LNT:", correlation.round(2))
```

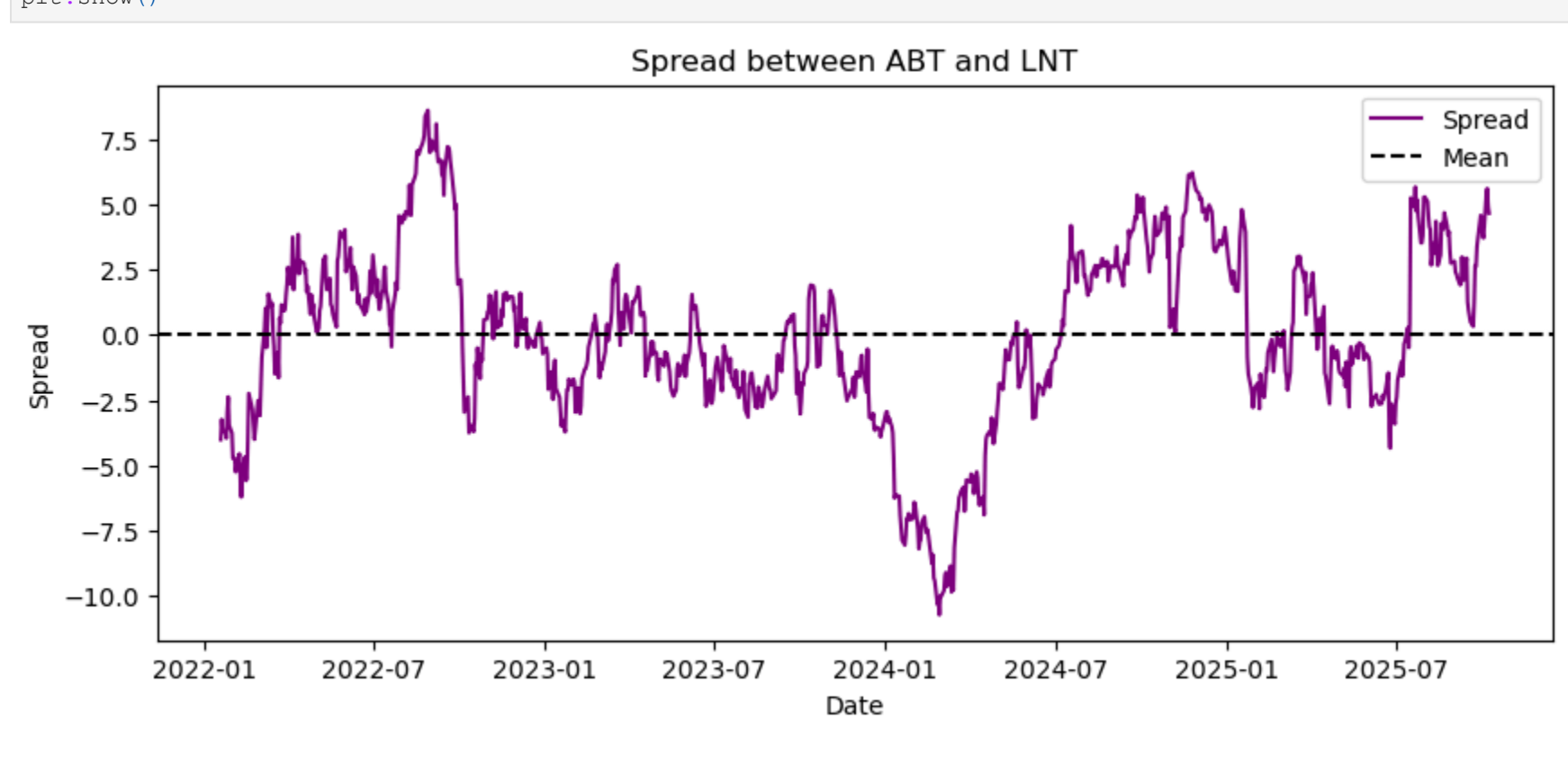
```
Correlation between ABT and LNT: 0.82
```

b. Computing the spread: OLS Regression

In [18]:
```python
pair_data = data[pair].copy()

X = pair_data['ABT']
y = pair_data['LNT']

# Add constant for intercept
X_const = sm.add_constant(X)
model = sm.OLS(y, X_const).fit()
beta = model.params['ABT']   # slope
alpha = model.params['const']   # intercept

# Compute spread
pair_data['Spread'] = y - (alpha + beta * X)
print(pair_data)

print(pair_data.describe())
```

```
                   ABT        LNT    Spread
Date
2022-01-19  117.356979  52.419163 -4.014524
2022-01-20  116.109398  52.692635 -3.242071
2022-01-21  117.152153  52.719101 -3.632664
2022-01-24  115.280769  51.828102 -3.775187
2022-01-25  114.768692  51.439930 -3.958549
...                ...        ...       ...
2025-10-03  133.994156  66.800003  3.712136
2025-10-06  133.147934  68.120003  5.370590
2025-10-07  132.431122  68.070000  5.607282
2025-10-08  133.675583  67.870003  4.909552
2025-10-09  132.719818  67.239998  4.661814

[935 rows x 3 columns]
              ABT         LNT        Spread
count  935.000000  935.000000  9.350000e+02
mean   109.146785   53.149950 -1.957602e-14
std     12.226493    5.951737  3.392668e+00
min     86.781013   43.104118 -1.072926e+01
25%    100.468239   48.268116 -1.985197e+00
50%    106.156654   51.315544 -1.285737e-01
75%    114.728615   57.888855  2.344256e+00
max    138.334625   68.120003  8.605297e+00
```

In [19]:
```python
plt.figure(figsize=(10,4))
plt.plot(pair_data['Spread'], label='Spread', color='purple')
plt.axhline(0, color='black', linestyle='--', label='Mean')
plt.title('Spread between ABT and LNT')
plt.xlabel('Date')
plt.ylabel('Spread')
plt.legend()
plt.show()
```



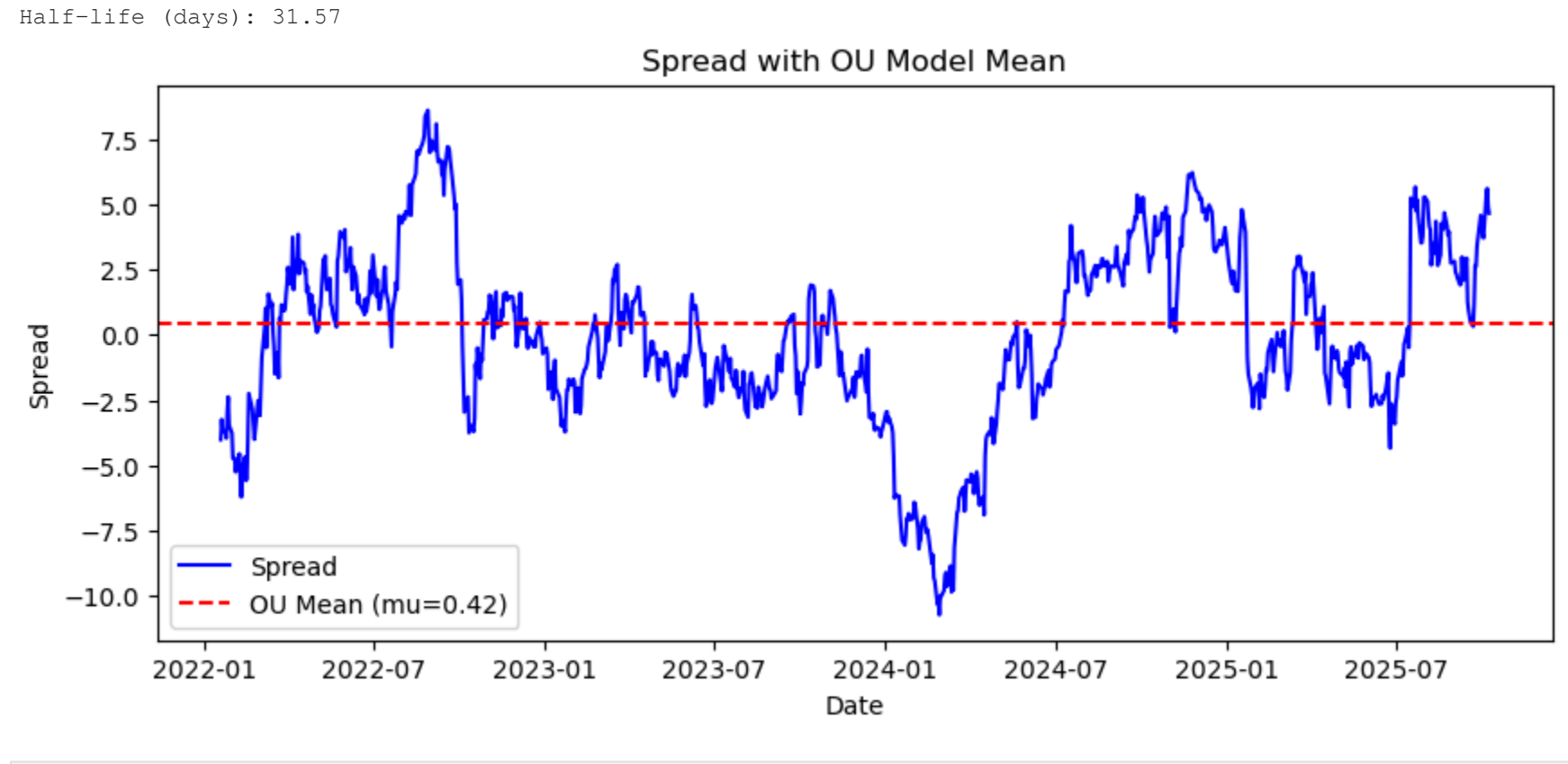
c. Fitting an Ornstein-Uhlenbeck Model

In [20]:
```python
spread = pair_data['Spread'].values

# Step 1: Lag the spread
spread_lag = spread[:-1]            # X_t
spread_next = spread[1:]            # X_{t+1}
delta_spread = spread_next - spread_lag   # ΔX_t

# Step 2: Fit AR(1) model: ΔX_t = a + b*X_t + ε_t
X = sm.add_constant(spread_lag)
model = sm.OLS(delta_spread, X).fit()

# Step 3: Extract parameters
a = model.params[0]     # intercept
b = model.params[1]     # slope

# Step 4: Compute OU parameters
theta = -np.log(1 + b)         # speed of mean reversion
mu = -a / b                     # long-term mean
sigma = np.std(model.resid)     # volatility

# Step 5: Compute half-life
half_life = np.log(2) / theta

# Print the OU model equation
print(f"OU Model: dX_t = {theta:.4f} * ({mu:.4f} - X_t) dt + {sigma:.4f} * dW_t")
print(f"Half-life (days): {half_life:.2f}")
```

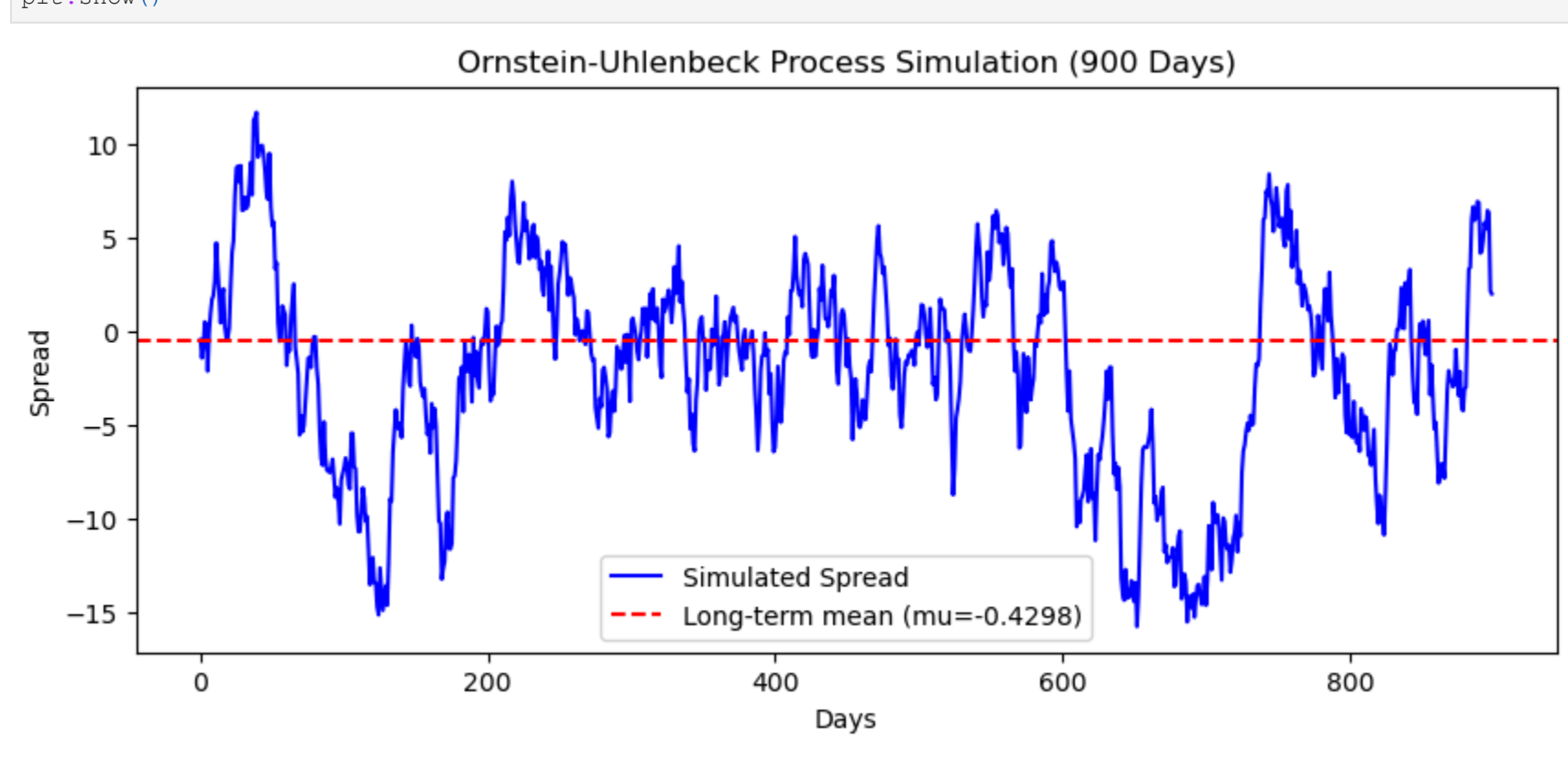
```
OU Model: dX_t = 0.0220 * (0.4228 - X_t) dt + 0.7068 * dW_t
Half-life (days): 31.57
```

In [21]:
```python
# Step 6: Plot the spread and OU mean
plt.figure(figsize=(10,4))
plt.plot(pair_data.index, spread, label='Spread', color='blue')
plt.axhline(mu, color='red', linestyle='--', label=f'OU Mean (mu={mu:.2f})')
plt.title('Spread with OU Model Mean')
plt.xlabel('Date')
plt.ylabel('Spread')
plt.legend()

# Display results
print(f"Mean reversion speed (theta): {theta:.4f}")
print(f"Long-term mean (mu): {mu:.4f}")
print(f"Volatility (sigma): {sigma:.4f}")
print(f"Half-life (days): {half_life:.2f}")
```

```
Mean reversion speed (theta): 0.0220
Long-term mean (mu): 0.4228
Volatility (sigma): 0.7068
Half-life (days): 31.57
```



In [22]:
```python
theta = 0.0243    # speed of mean reversion
mu = -0.4298       # long-term mean
sigma = 1.4869    # volatility
days = 900     # simulation length

# Step 1: Initialize the process
dt = 1  # daily steps
X = np.zeros(days)
X[0] = mu  # start at mean

# Step 2: Simulate OU process
for t in range(1, days):
    X[t] = X[t-1] + theta*(mu - X[t-1])*dt + sigma*np.sqrt(dt)*np.random.normal()

# Step 3: Plot the simulated OU process
plt.figure(figsize=(10,4))
plt.plot(X, label='Simulated Spread', color='blue')
plt.axhline(mu, color='red', linestyle='--', label=f'Long-term mean (mu={mu})')
plt.title('Ornstein-Uhlenbeck Process Simulation (900 Days)')
plt.xlabel('Days')
plt.ylabel('Spread')
plt.legend()
plt.show()
```

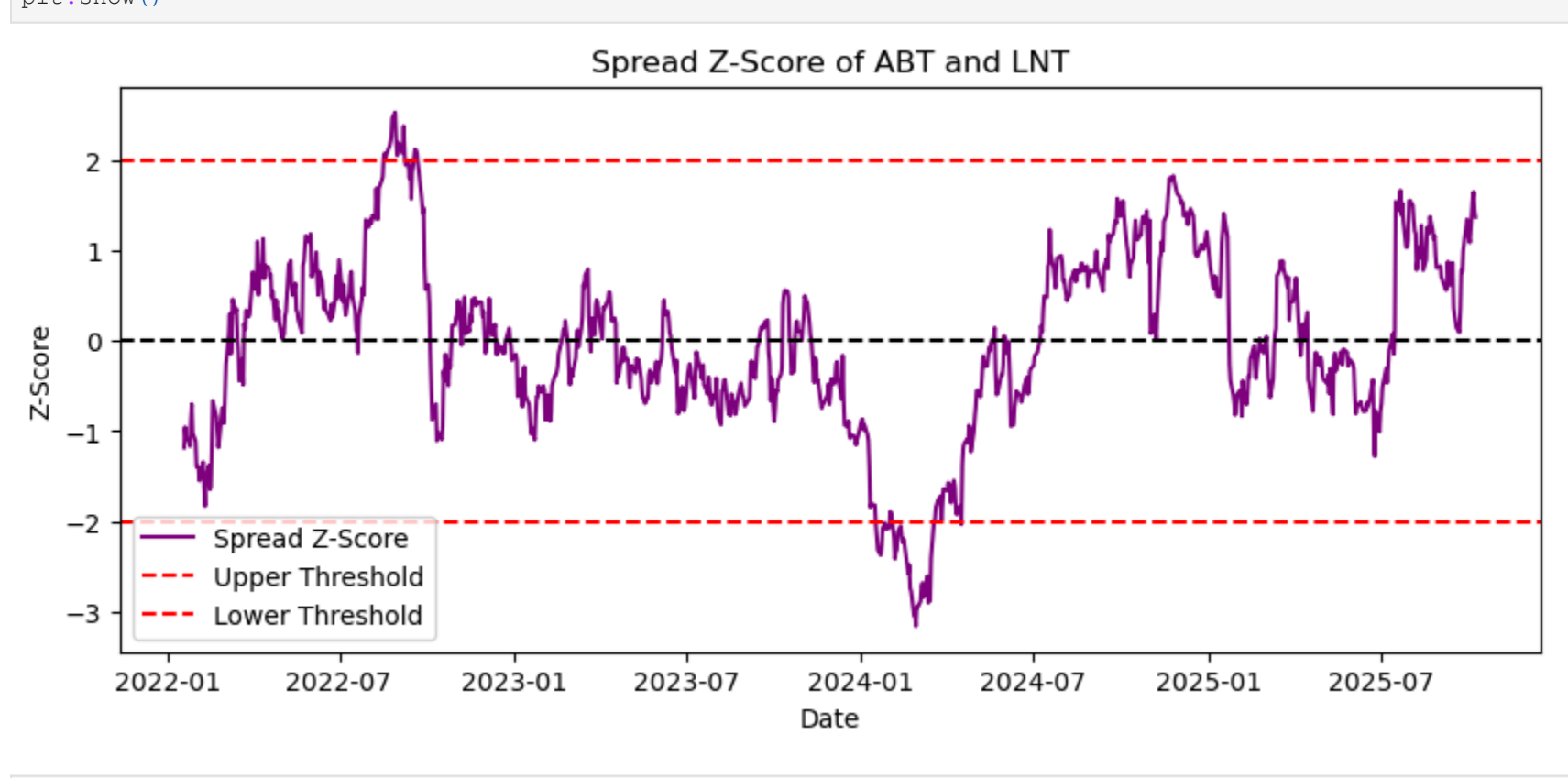


d. Computing the z-score of the spreads

In [23]:
```python
pair_data['Z_Score'] = (pair_data['Spread'] - pair_data['Spread'].mean()) / pair_data['Spread'].std()
print(pair_data)
```

```
                   ABT        LNT    Spread   Z_Score
Date
2022-01-19  117.356979  52.419163 -4.014524 -1.183294
2022-01-20  116.109398  52.692635 -3.242071 -0.955611
2022-01-21  117.152153  52.719101 -3.632664 -1.070739
2022-01-24  115.280769  51.828102 -3.775187 -1.112749
2022-01-25  114.768692  51.439930 -3.958549 -1.166795
...                ...        ...       ...       ...
2025-10-03  133.994156  66.800003  3.712136  1.094164
2025-10-06  133.147934  68.120003  5.370590  1.582999
2025-10-07  132.431122  68.070000  5.607282  1.652764
2025-10-08  133.675583  67.870003  4.909552  1.447106
2025-10-09  132.719818  67.239998  4.661814  1.374085

[935 rows x 4 columns]
```

In [ ]:
```python
plt.figure(figsize=(10,4))
plt.plot(pair_data['Z_Score'], label='Spread Z-Score', color='purple')
plt.axhline(0, color='black', linestyle='--')
plt.axhline(2, color='red', linestyle='--', label='Upper Threshold')
plt.axhline(-2, color='red', linestyle='--', label='Lower Threshold')
plt.title('Spread Z-Score of ABT and LNT')
plt.xlabel('Date')
plt.ylabel('Z-Score')
plt.legend()
plt.show()
```



In [ ]:
```python
# Define file path and name
file_path = 'pair_data.csv'

# Save the dataframe to CSV
pair_data.to_csv(file_path, index=False)

print(f"pair_data has been saved as {file_path}")
```