In [1]:

```python
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
```

In [2]:

```python
def preprocess_data(file_path):
    # Load the data
    df = pd.read_csv(file_path)

    print(df.head())
    print(df.tail())
    print(df.info())
    print(df.describe())

    df['Date'] = pd.to_datetime(df['Date'], errors='coerce')
    df.set_index('Date', inplace=True)

    # Fill missing values with the mean
    df.fillna(df.mean(), inplace=True)

    # Detect and handle outliers using z-scores
    z_scores = (df - df.mean()) / df.std()

    # Use mask to replace values where condition is True
    df = df.mask(z_scores.abs() > 3)

    # Resample to daily intervals
    df = df.resample('D').agg({'Open': 'first', 'High': 'max', 'Low': 'min',
                               'Close': 'last', 'Adj Close': 'last', 'Volume': 'sum'})

    # Normalize/scaling using StandardScaler
    scaler = StandardScaler()
    numerical_cols = ['Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume']
    df[numerical_cols] = scaler.fit_transform(df[numerical_cols])

    # Feature engineering (moving average)
    df['MA20'] = df['Close'].rolling(window=20).mean()

    df.to_csv("preprocessed_btc_data.csv")
```

In [3]:

```python
preprocess_data("C:/Users/vijay/Downloads/IITR/BTC-USD.csv")
```

```
         Date          Open          High           Low         Close  \
0  2018-01-01  14112.200195  14112.200195  13154.700195  13657.200195   
1  2018-01-02  13625.000000  15444.599609  13163.599609  14982.099609   
2  2018-01-03  14978.200195  15572.799805  14844.500000  15201.000000   
3  2018-01-04  15270.700195  15739.700195  14522.200195  15599.200195   
4  2018-01-05  15477.200195  17705.199219  15202.799805  17429.500000   

      Adj Close       Volume  
0  13657.200195  10291200000  
1  14982.099609  16846600192  
2  15201.000000  16871900160  
3  15599.200195  21783199744  
4  17429.500000  23840899072  
            Date          Open          High           Low         Close  \
1487  2022-01-27  36841.878906  37148.324219  35629.281250  37138.234375   
1488  2022-01-28  37128.445313  37952.878906  36211.109375  37784.332031   
1489  2022-01-29  37780.714844  38576.261719  37406.472656  38138.179688   
1490  2022-01-30  38151.917969  38266.339844  37437.710938  37917.601563   
1491  2022-01-31  37920.281250  38647.261719  36733.574219  38483.125000   

         Adj Close       Volume  
1487  37138.234375  25041426629  
1488  37784.332031  22238830523  
1489  38138.179688  17194183075  
1490  37917.601563  14643548444  
1491  38483.125000  20734730465  
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1492 entries, 0 to 1491
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   Date       1492 non-null   object 
 1   Open       1492 non-null   float64
 2   High       1492 non-null   float64
 3   Low        1492 non-null   float64
 4   Close      1492 non-null   float64
 5   Adj Close  1492 non-null   float64
 6   Volume     1492 non-null   int64  
dtypes: float64(5), int64(1), object(1)
memory usage: 81.7+ KB
None
               Open          High           Low         Close     Adj Close  \
count   1492.000000   1492.000000   1492.000000   1492.000000   1492.000000   
mean   18834.412309  19328.141889  18277.369294  18847.704874  18847.704874   
std    17877.430021  18363.563384  17309.553030  17878.523772  17878.523772   
min     3236.274658   3275.377930   3191.303467   3236.761719   3236.761719   
25%     7184.975952   7320.137085   6939.701782   7186.172241   7186.172241   
50%     9543.832032   9717.452148   9358.137695   9544.869629   9544.869629   
75%    32745.621582  33915.827149  31244.488770  32792.104492  32792.104492   
max    67549.734375  68789.625000  66382.062500  67566.828125  67566.828125   

             Volume  
count  1.492000e+03  
mean   2.583255e+10  
std    2.103589e+10  
min    2.923670e+09  
25%    8.475918e+09  
50%    2.247845e+10  
75%    3.620490e+10  
max    3.509679e+11  
```

In [4]:

```
def mean_reversion_strategy(df, window=20, threshold=2):
    df['MA'] = df['Close'].rolling(window=window).mean()
    df['std'] = df['Close'].rolling(window=window).std()
    df['z_score'] = (df['Close'] - df['MA']) / df['std']

    signals = []

    for i in range(1, len(df)):
        if df['z_score'].iloc[i] > threshold:
            signals.append(1)  # Buy
        elif df['z_score'].iloc[i] < -threshold:
            signals.append(-1)  # Sell
        else:
            signals.append(0)  # Hold
    return signals
```

In [5]:

```
df = pd.read_csv("preprocessed_btc_data.csv")
df['Date'] = pd.to_datetime(df['Date'])
```

In [6]:

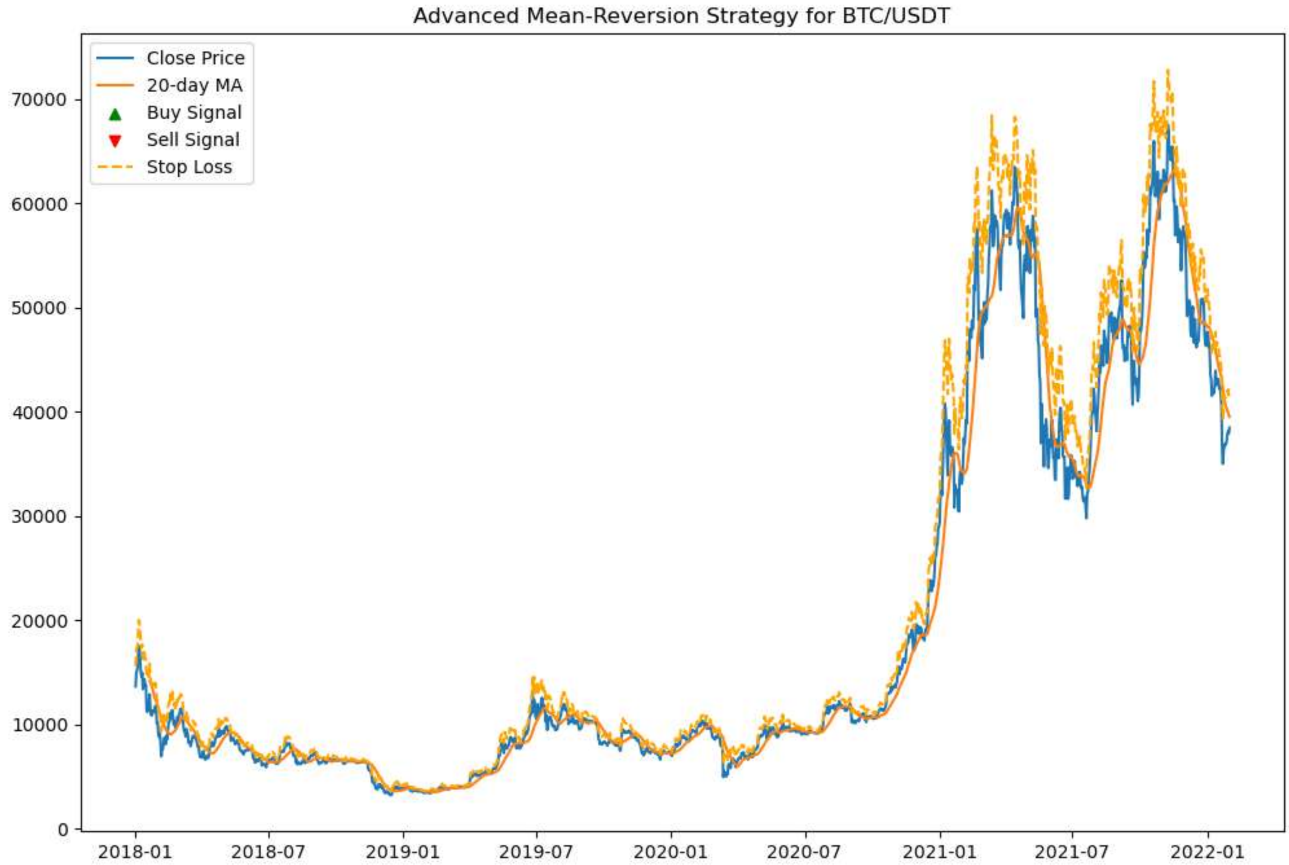
```
df.set_index('Date', inplace=True)
signals = mean_reversion_strategy(df)
```

In [7]:

```
def advanced_mean_reversion_strategy(data, atr_window=14, risk_factor=2):
    data['20_MA'] = data['Close'].rolling(window=20).mean()

    # Calculate ATR for volatility
    data['ATR'] = data['High'] - data['Low']
    data['ATR'] = data['ATR'].ewm(span=atr_window, adjust=False).mean()

    # Set dynamic oversold and overbought thresholds based on ATR
    data['Oversold_Threshold'] = -risk_factor * data['ATR']
    data['Overbought_Threshold'] = risk_factor * data['ATR']

    # Generate signals
    data['Signal'] = 0
    data.loc[data['Close'] < (1 + data['Oversold_Threshold'] / 100) *
    data['20_MA'], 'Signal'] = 1

    data.loc[data['Close'] > (1 + data['Overbought_Threshold'] / 100) *
    data['20_MA'], 'Signal'] = -1

    # Execute trades
    data['Position'] = data['Signal'].diff()
    data['Position'].fillna(0, inplace=True)

    # Calculate daily returns
    data['Daily_Returns'] = data['Close'].pct_change()

    # Calculate trade returns
    data['Trade_Returns'] = data['Position'].shift(1) * data['Daily_Returns']

    # Calculate stop-loss levels
    data['Stop_Loss'] = data['Close'] - data['Oversold_Threshold']
    data['Stop_Loss'].fillna(0, inplace=True)

    # Plot results
    plt.figure(figsize=(12, 8))
    plt.plot(data['Close'], label='Close Price')
    plt.plot(data['20_MA'], label='20-day MA')
    plt.scatter(data[data['Position'] == 1].index, data[data['Position'] == 1]
                ['Close'], marker='^', color='g', label='Buy Signal')

    plt.scatter(data[data['Position'] == -1].index, data[data['Position'] == -1]
                ['Close'], marker='v', color='r', label='Sell Signal')

    plt.plot(data['Stop_Loss'], label='Stop Loss', linestyle='--', color='orange')

    plt.title('Advanced Mean-Reversion Strategy for BTC/USDT')
    plt.legend()
    plt.show()
```

In [8]:

```
data = pd.read_csv('C:/Users/vijay/Downloads/IITR/BTC-USD.csv')
data['Date'] = pd.to_datetime(data['Date'])
data.set_index('Date', inplace=True)
advanced_mean_reversion_strategy(data)
```



In [9]:

```
def backtest_strategy(strategy_function, data, transaction_cost=0.0015):
    # Implement backtesting with transaction costs and slippage
    signals = strategy_function(data.copy())
    signals = pd.Series(signals)  # Convert list to Series if necessary
    positions = signals.diff()
    positions.fillna(0, inplace=True)

    # Calculate actual returns, incorporating slippage and transaction costs
    actual_returns = positions.shift(1) * data['Close'].pct_change() *
    (1 - transaction_cost)

    actual_returns.iloc[0] = 0  # Correct the first return

    # Calculate cumulative returns
    cumulative_returns = (1 + actual_returns).cumprod()
    return cumulative_returns.iloc[-1], actual_returns.std()
```

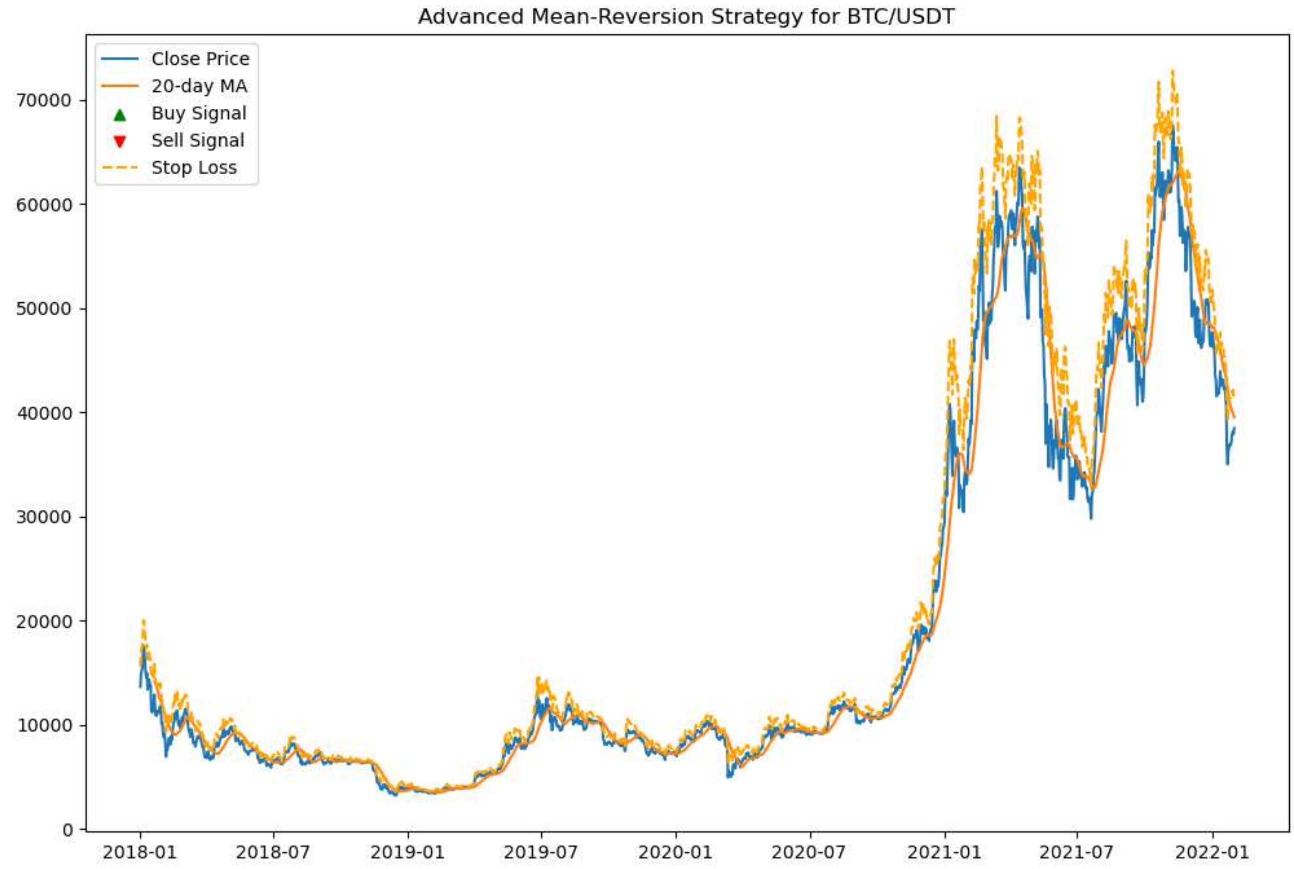
In [10]:

```
data = pd.read_csv('C:/Users/vijay/Downloads/IITR/BTC-USD.csv')

data['Date'] = pd.to_datetime(data['Date'])
data.set_index('Date', inplace=True)

preprocess_data("C:/Users/vijay/Downloads/IITR/BTC-USD.csv")
```

```
         Date          Open          High           Low         Close  \
0  2018-01-01  14112.200195  14112.200195  13154.700195  13657.200195
1  2018-01-02  13625.000000  15444.599609  13163.599609  14982.099609
2  2018-01-03  14978.200195  15572.799805  14844.500000  15201.000000
3  2018-01-04  15270.700195  15739.700195  14522.200195  15599.200195
4  2018-01-05  15477.200195  17705.199219  15202.799805  17429.500000

      Adj Close       Volume
0  13657.200195  10291200000
1  14982.099609  16846600192
2  15201.000000  16871900160
3  15599.200195  21783199744
4  17429.500000  23840899072
            Date          Open          High           Low         Close  \
1487  2022-01-27  36841.878906  37148.324219  35629.281250  37138.234375
1488  2022-01-28  37128.445313  37952.878906  36211.109375  37784.332031
1489  2022-01-29  37780.714844  38576.261719  37406.472656  38138.179688
1490  2022-01-30  38151.917969  38266.339844  37437.710938  37917.601563
1491  2022-01-31  37920.281250  38647.261719  36733.574219  38483.125000

         Adj Close       Volume
1487  37138.234375  25041426629
1488  37784.332031  22238830523
1489  38138.179688  17194183075
1490  37917.601563  14643548444
1491  38483.125000  20734730465
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1492 entries, 0 to 1491
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   Date       1492 non-null   object 
 1   Open       1492 non-null   float64
 2   High       1492 non-null   float64
 3   Low        1492 non-null   float64
 4   Close      1492 non-null   float64
 5   Adj Close  1492 non-null   float64
 6   Volume     1492 non-null   int64  
dtypes: float64(5), int64(1), object(1)
memory usage: 81.7+ KB
None
               Open          High           Low         Close     Adj Close  \
count   1492.000000   1492.000000   1492.000000   1492.000000   1492.000000
mean   18834.412309  19328.141889  18277.369294  18847.704874  18847.704874
std    17877.430021  18363.563384  17309.553030  17878.523772  17878.523772
min     3236.274658   3275.377930   3191.303467   3236.761719   3236.761719
25%     7184.975952   7320.137085   6939.701782   7186.172241   7186.172241
50%     9543.832032   9717.452148   9358.137695   9544.869629   9544.869629
75%    32745.621582  33915.827149  31244.488770  32792.104492  32792.104492
max    67549.734375  68789.625000  66382.062500  67566.828125  67566.828125

             Volume
count  1.492000e+03
mean   2.583255e+10
std    2.103589e+10
min    2.923670e+09
25%    8.475918e+09
50%    2.247845e+10
75%    3.620490e+10
max    3.509679e+11
```

In [11]:

```
# Backtest the strategies
mean_reversion_results = backtest_strategy(mean_reversion_strategy, data.copy())

advanced_mean_reversion_results =
backtest_strategy(advanced_mean_reversion_strategy, data.copy())
```


Advanced Mean-Reversion Strategy for BTC/USDT
Close Price
20-day MA
▲ Buy Signal
▼ Sell Signal
Stop Loss