

In [198...]

```
import ccxt
import pandas as pd
import numpy as np
import talib as ta
import matplotlib.pyplot as plt
import dash
import plotly.express as px
import plotly.graph_objects as go
import urllib
import socket
import os
from datetime import datetime, timedelta
from sqlalchemy import create_engine
from dash import dcc, html
from flask import Flask
```

In [199...]

```
# Initialize Kraken exchange via ccxt
kraken = ccxt.kraken()
```

In [200...]

```
# Download historical data from Kraken
def get_crypto_data(symbol, timeframe='1m'):
    # Calculate the 'since' timestamp to fetch data from 7 days ago
    seven_days_ago = datetime.now() - timedelta(days=7)
    since = int(seven_days_ago.timestamp() * 1000) # Convert to milliseconds

    ohlcv = kraken.fetch_ohlcv(symbol, timeframe=timeframe, since=since)

    # Ensure the columns match the typical OHLCV format returned by Kraken
    df = pd.DataFrame(ohlcv, columns=['timestamp', 'Open', 'High', 'Low', 'Close'])

    # Convert the timestamp from milliseconds to datetime and set as index
    df['timestamp'] = pd.to_datetime(df['timestamp'], unit='ms')
    df.set_index('timestamp', inplace=True)

    return df
```

In [201...]

```
# Create Calculations of Volatility
def calculate_volatility(df, window):
    df['returns'] = df['Close'].pct_change()
    df['volatility'] = df['returns'].rolling(window=window).std() * np.sqrt(window)
    return df
```

In [202...]

```
# Find the support and Resistance High and Low Factors
def find_support_resistance(df):
    df['support'] = df['Low'].rolling(window=60).min()
    df['resistance'] = df['High'].rolling(window=60).max()
    return df
```

```
In [203...]: # Calculate the moving averages
def calculate_moving_averages(df, short_window=14, long_window=50):
    df['SMA_14'] = ta.SMA(df['Close'], timeperiod=short_window)
    df['EMA_50'] = ta.EMA(df['Close'], timeperiod=long_window)
    return df
```

```
In [204...]: # Calcualte the bollinger bands
def calculate_bollinger_bands(df, window=20, num_std=2):
    df['BB_upper'], df['BB_middle'], df['BB_lower'] = ta.BBANDS(df['Close'], timepe
    return df
```

```
In [205...]: # Calculate the RSI
def calculate_rsi(df, period=14):
    df['RSI'] = ta.RSI(df['Close'], timeperiod=period)
    return df
```

```
In [206...]: # Calculate the VWAP
def calculate_vwap(df):
    df['vwap'] = (df['Volume'] * (df['High'] + df['Low'] + df['Close']) / 3).cumsum
    return df
```

```
In [207...]: # Calcualte the Fibonacci Levels
def calculate_fibonacci_levels(df):
    max_price = df['Close'].max()
    min_price = df['Close'].min()
    diff = max_price - min_price
    df['fib_0.236'] = max_price - 0.236 * diff
    df['fib_0.382'] = max_price - 0.382 * diff
    df['fib_0.5'] = max_price - 0.5 * diff
    df['fib_0.618'] = max_price - 0.618 * diff
    df['fib_1'] = min_price
    return df
```

```
In [208...]: # Calculate the MACD
def calculate_macd(df):
    df['macd'], df['macdsignal'], df['macdhist'] = ta.MACD(df['Close'], fastperiod=
    return df
```

```
In [209...]: # Calcualte the ATR
def calculate_atr(df, window=14):
    df['ATR'] = ta.ATR(df['High'], df['Low'], df['Close'], timeperiod=window)
    return df
```

```
In [210...]: # Stochastic Oscillator
def calculate_stochastic(df, k_window=14, d_window=3):
    df['slowk'], df['slowd'] = ta.STOCH(df['High'], df['Low'], df['Close'], fastk_p
    return df
```

```
In [211...]: # Ichimoku Cloud
def calculate_ichimoku(df):
    df['ichimoku_a'], df['ichimoku_b'], df['ichimoku_c'], df['ichimoku_d'], df['ich
    return df
```

```
In [212... # Parabolic SAR (Stop and Reverse)
def calculate_parabolic_sar(df):
    df['SAR'] = ta.SAR(df['High'], df['Low'], acceleration=0.02, maximum=0.2)
    return df
```

```
In [213... # ADX (Average Directional Index)
def calculate_adx(df, period=14):
    df['ADX'] = ta.ADX(df['High'], df['Low'], df['Close'], timeperiod=period)
    return df
```

```
In [214... # Chaikin Money Flow (CMF)
def calculate_cmf(df, window=20):
    df['CMF'] = ta.ADOOSC(df['High'], df['Low'], df['Close'], df['Volume'], df['4H'])
    return df
```

```
In [215... # On-Balance Volume (OBV)
def calculate_obv(df):
    df['OBV'] = ta.OBV(df['Close'], df['Volume'])
    return df
```

```
In [216... # Sweep and Clean the data
def clean_data(df):
    df.dropna(how='all', inplace=True)
    df.fillna(method='ffill') # Forward fill missing data
    df.fillna(method='bfill') # Backward fill missing data
    df.replace([np.inf, -np.inf], np.nan, inplace=True)
    df.dropna(inplace=True)
    return df
```

```
In [217... # Calculate buy/sell signal based on percentage change
def calculate_buy_sell_signal(df, threshold=0.15):
    # Calculate the percentage change from the previous close
    df['percent_change'] = df['Close'].pct_change() * 100

    # Generate "BUY" or "SELL" based on the threshold
    df['Signal'] = df['percent_change'].apply(lambda x: "SELL" if abs(x) >= threshold else "BUY")
    return df
```

```
In [218... def plot_data(df, symbol):
    """Plot the closing price and indicators for a given symbol."""
    plt.figure(figsize=(14, 10))
    plt.subplot(2, 1, 1)
    plt.plot(df['Close'], label='Close Price')

    # Plot SMA_14 if available
    if 'SMA_14' in df.columns:
        plt.plot(df['SMA_14'], label='SMA 14', linestyle='--')
    else:
        print(f"'SMA_14' not found for {symbol}. Skipping SMA plot.")

    # Plot EMA_50 if available
    if 'EMA_50' in df.columns:
        plt.plot(df['EMA_50'], label='EMA 50', linestyle='--')
    else:
```

```

    print(f"'EMA_50' not found for {symbol}. Skipping EMA plot.")

    # Plot Bollinger Bands if available
    if 'BB_upper' in df.columns and 'BB_lower' in df.columns:
        plt.plot(df['BB_upper'], label='Upper BB', linestyle='--')
        plt.plot(df['BB_lower'], label='Lower BB', linestyle='--')
    else:
        print(f"Bollinger Bands not found for {symbol}. Skipping BB plot.")

    plt.title(f"{symbol} Price with Indicators")
    plt.legend()
    plt.show()

```

In [219...]

```

def process_symbol_data(symbol, timeframe, since):
    """Fetch and process data for a specific symbol."""
    print(f"\nFetching data for {symbol}...")
    df = get_crypto_data(symbol, timeframe, since)
    if df is None or df.empty:
        print(f"No data returned for {symbol}. Skipping...")
        return None # Skip processing if no data is available

    # Sequential data processing with error handling
    try:
        df = clean_data(df)
        df = calculate_moving_averages(df)
        df = calculate_bollinger_bands(df)
        df = calculate_rsi(df)
        df = calculate_volatility(df, window=14)
        df = find_support_resistance(df)
        df = calculate_vwap(df)
        df = calculate_fibonacci_levels(df)
        df = calculate_macd(df)
        df = calculate_atr(df, window=14)
        df = calculate_buy_sell_signal(df)
    except Exception as e:
        print(f"Error processing data for {symbol}: {e}")
        return None

    return df

```

In [220...]

```

def process_symbol_data(symbol, timeframe, since):
    try:
        df = get_crypto_data(symbol, timeframe, since)
        print(f"Data for {symbol}:\n{df.head()}" ) # Print the DataFrame for inspection
        # Ensure 'Close' column exists
        if 'close' not in df.columns:
            print(f"Warning: 'close' column missing for {symbol}")
            return None # Return None if the 'close' column is missing
        df.rename(columns={'close': 'Close'}, inplace=True) # Rename to 'Close' if needed
    except Exception as e:
        print(f"Error processing data for {symbol}: {e}")
        return None

```

In [221...]

```
# Database connection configuration
DATABASE_TYPE = 'mssql'
DBAPI = 'pyodbc'
SERVER = 'MARTIN'
DATABASE = 'crypto_data'
DRIVER = 'ODBC Driver 17 for SQL Server'

# SQLAlchemy engine for SQL Server connection
engine = create_engine(f'{DATABASE_TYPE}+{DBAPI}://@{SERVER}/{DATABASE}?driver={DRIVER}')

# Define function to save data to SQL
def save_to_sql(df, table_name):
    # try:
    #     if df.empty:
    #         print("Data is empty after cleaning. Nothing to save.")
    #         return
    #     df.to_sql(table_name, con=engine, if_exists='replace', index_label='timestamp')
    #     print(f"Data successfully saved to SQL table: {table_name}")
    # except Exception as e:
    #     print(f"Error saving to SQL Server: {e}")
    # finally:
    #     engine.dispose()
    #     print("SQL connection closed.")

# Define function to save data to CSV
def save_to_csv(df, file_name):
    file_path = os.path.join(os.getcwd(), file_name)
    df.to_csv(file_path, index=True)
    print(f"Data saved to CSV file: {file_name}")

# Define function to fetch and process data for a specific symbol
def get_crypto_data(symbol, timeframe='1m', since=None):
    # Ensure kraken is defined and connected
    ohlcv = kraken.fetch_ohlcv(symbol, timeframe=timeframe, since=since)
    df = pd.DataFrame(ohlcv, columns=['timestamp', 'open', 'high', 'low', 'close',
                                      'timestamp'] = pd.to_datetime(df['timestamp'], unit='ms')
    df.set_index('timestamp', inplace=True)
    return df

# Define function to process symbol data
def process_symbol_data(symbol, timeframe, since):
    try:
        df = get_crypto_data(symbol, timeframe, since)

        if 'close' not in df.columns:
            print(f"Warning: 'close' column missing for {symbol}")
            return None
        df.rename(columns={'close': 'Close'}, inplace=True)

        df['Signal'] = None
        df.loc[df.index[1:], 'Signal'] = [
            'BUY' if df['Close'].iloc[i] > df['Close'].iloc[i-1] else 'SELL'
            for i in range(1, len(df))
        ]
    
```

```

df['Volatility'] = df['high'] - df['low']

# Calculate indicators
df['SMA_14'] = df['Close'].rolling(window=14).mean()
df['EMA_50'] = df['Close'].ewm(span=50, adjust=False).mean()
df['BB_upper'] = df['SMA_14'] + 2 * df['Close'].rolling(window=14).std()
df['BB_lower'] = df['SMA_14'] - 2 * df['Close'].rolling(window=14).std()
df['RSI'] = compute_rsi(df['Close'], 14)

return df
except Exception as e:
    print(f"Error processing data for {symbol}: {e}")
return None

def compute_rsi(series, window):
    delta = series.diff(1)
    gain = (delta.where(delta > 0, 0)).rolling(window=window).mean()
    loss = (-delta.where(delta < 0, 0)).rolling(window=window).mean()
    rs = gain / loss
    return 100 - (100 / (1 + rs))

# Plot data points
def plot_data(df, symbol):
    plt.figure(figsize=(14, 8))

    plt.subplot(2, 1, 1)
    plt.plot(df['Close'], label='Close Price')
    plt.plot(df['SMA_14'], label='SMA 14', linestyle='--')
    plt.plot(df['EMA_50'], label='EMA 50', linestyle='--')
    plt.plot(df['BB_upper'], label='Upper BB', linestyle='--')
    plt.plot(df['BB_lower'], label='Lower BB', linestyle='--')
    plt.title(f'{symbol} Close Price with Moving Averages and Bollinger Bands')
    plt.legend()

    plt.subplot(2, 1, 2)
    plt.plot(df['RSI'], label='RSI', color='green')
    plt.axhline(70, color='red', linestyle='--', label='Overbought (70)')
    plt.axhline(30, color='blue', linestyle='--', label='Oversold (30)')
    plt.title(f'{symbol} RSI')
    plt.legend()

    plt.tight_layout()
    plt.show()

# Plot signal and volatility
plt.figure(figsize=(14, 8))
plt.subplot(2, 1, 1)
sell_signals = df['Signal'].value_counts().get('SELL', 0)
plt.plot(df.index, [sell_signals] * len(df), label='Sell Signals')
plt.title(f'{symbol} Sell Signal Counts')
plt.legend()

plt.subplot(2, 1, 2)
plt.plot(df.index, df['Volatility'], label='Volatility', color='orange')
plt.title(f'{symbol} Volatility')

```

```

plt.legend()
plt.tight_layout()
plt.show()

# Main function
def main():
    symbols = [
        'ADA/USD', 'APE/USD', 'AUCTION/USD', 'BODEN/USD', 'BTC/USD', 'CPOOL/USD', 'EUL/USD', 'GMT/USD', 'LINK/USD', 'USDT/USD', 'MEME/USD', 'MNT/USD', 'MOG/UNTRN/USD', 'PYTH/USD', 'RENDER/USD', 'SAFE/USD', 'SUPER/USD', 'TNSR/USD', 'XMR/USD', 'ZRX/USD', 'LTC/USD', 'DOGE/USD'
    ]
    timeframe = '1m'
    since = kraken.parse8601('2024-01-01T00:00:00Z')
    risk_threshold = 0.05
    all_crypto_data = {}
    sell_counts = {}
    buy_counts = {}
    risk_labels = {}

    for symbol in symbols:
        df = process_symbol_data(symbol, timeframe, since)
        if df is None or 'Close' not in df.columns or 'Signal' not in df.columns:
            print(f"Skipping {symbol}: Data not available or required columns missing")
            continue

        sell_counts[symbol] = df['Signal'].value_counts().get('SELL', 0)
        buy_counts[symbol] = df['Signal'].value_counts().get('BUY', 0)

        avg_volatility = df['Volatility'].mean() if 'Volatility' in df.columns else None
        risk_label = "High Risk" if avg_volatility and avg_volatility > risk_threshold else "Low Risk"
        risk_labels[symbol] = risk_label
        df['Risk'] = risk_label

        all_crypto_data[symbol] = df[['Close', 'Signal', 'Risk']]
        table_name = f"{symbol.replace('/', '_")}_data"
        save_to_sql(df, table_name)
        save_to_csv(df, f"{table_name}.csv")
        plot_data(df, symbol)

    top_sell_symbols = sorted(sell_counts.items(), key=lambda x: x[1], reverse=True)
    print("\nTop 10 Symbols with Most 'SELL' Signals:")
    for symbol, count in top_sell_symbols:
        print(f"{symbol}: {count} 'SELL' signals")

    print("\nRisk Classification:")
    for symbol, risk in risk_labels.items():
        print(f"{symbol}: {risk}")

    return all_crypto_data

if __name__ == "__main__":
    all_crypto_data = main()

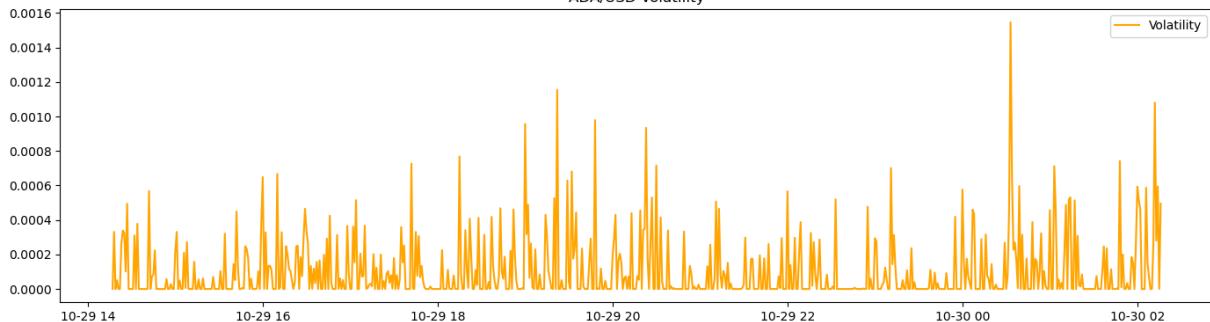
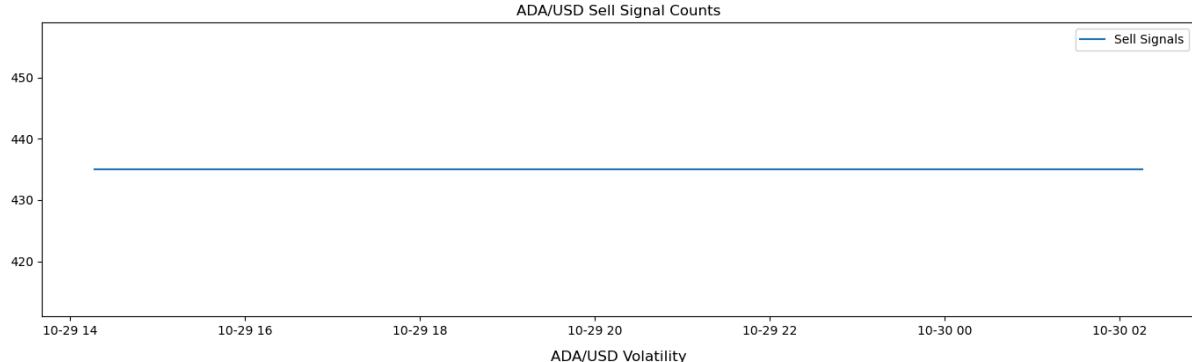
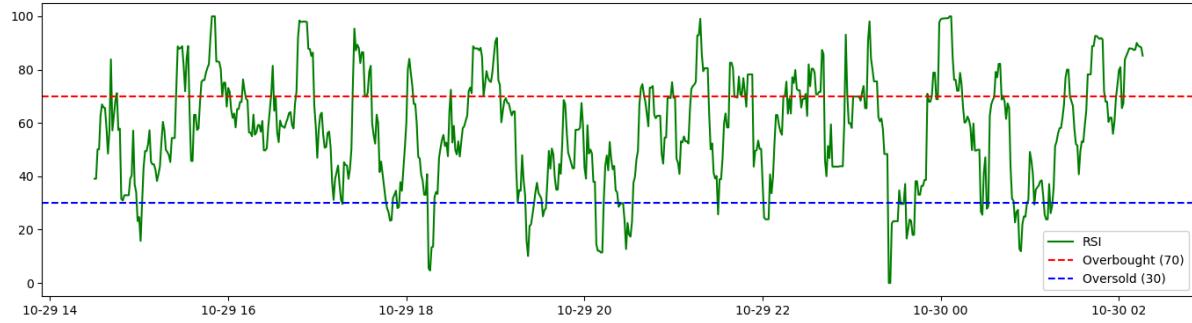
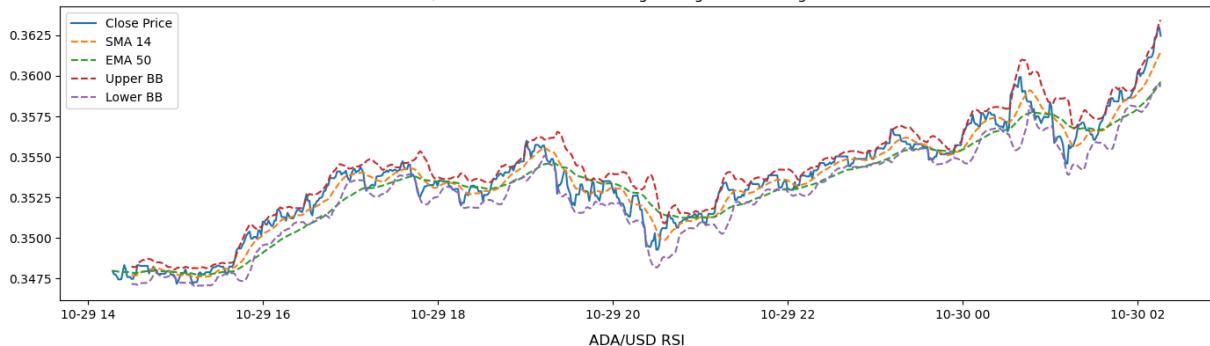
```

Data successfully saved to SQL table: ADA_USD_data

SQL connection closed.

Data saved to CSV file: ADA_USD_data.csv

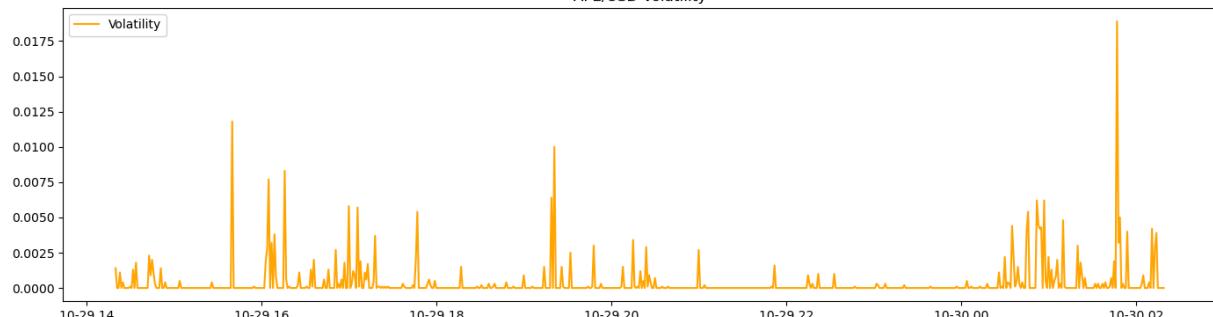
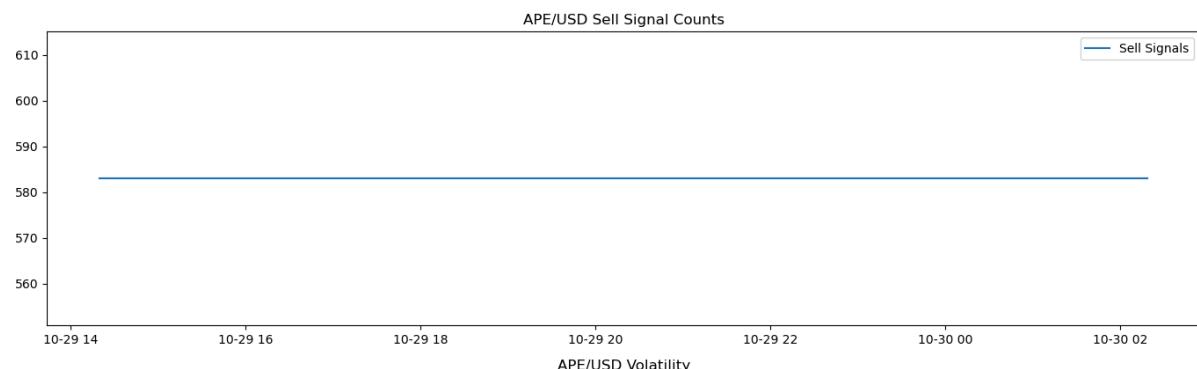
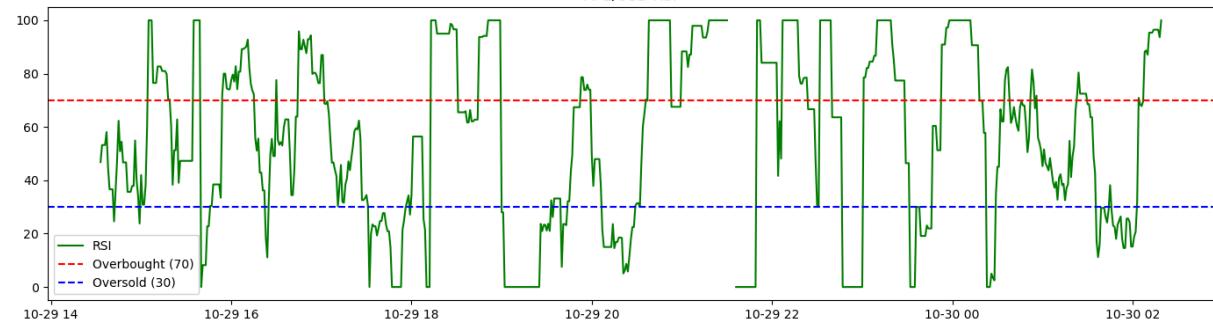
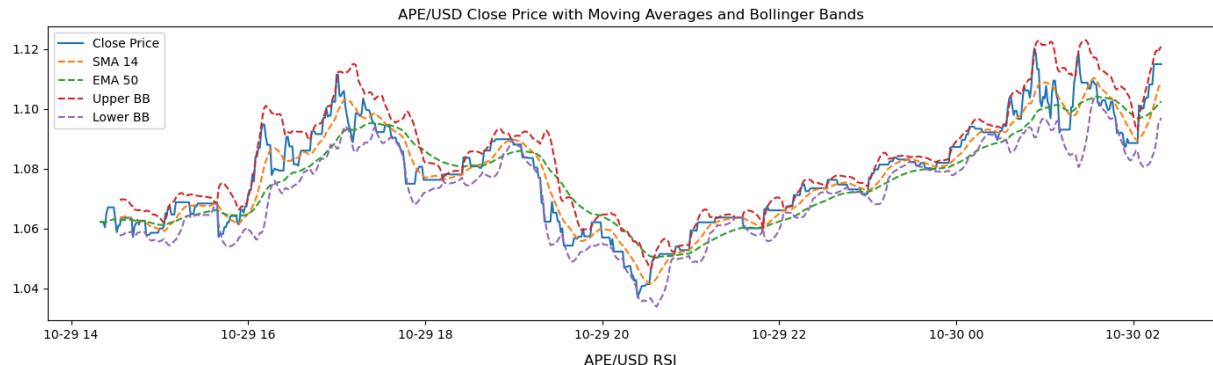
ADA/USD Close Price with Moving Averages and Bollinger Bands



Data successfully saved to SQL table: APE_USD_data

SQL connection closed.

Data saved to CSV file: APE_USD_data.csv



Data successfully saved to SQL table: AUCTION_USD_data

SQL connection closed.

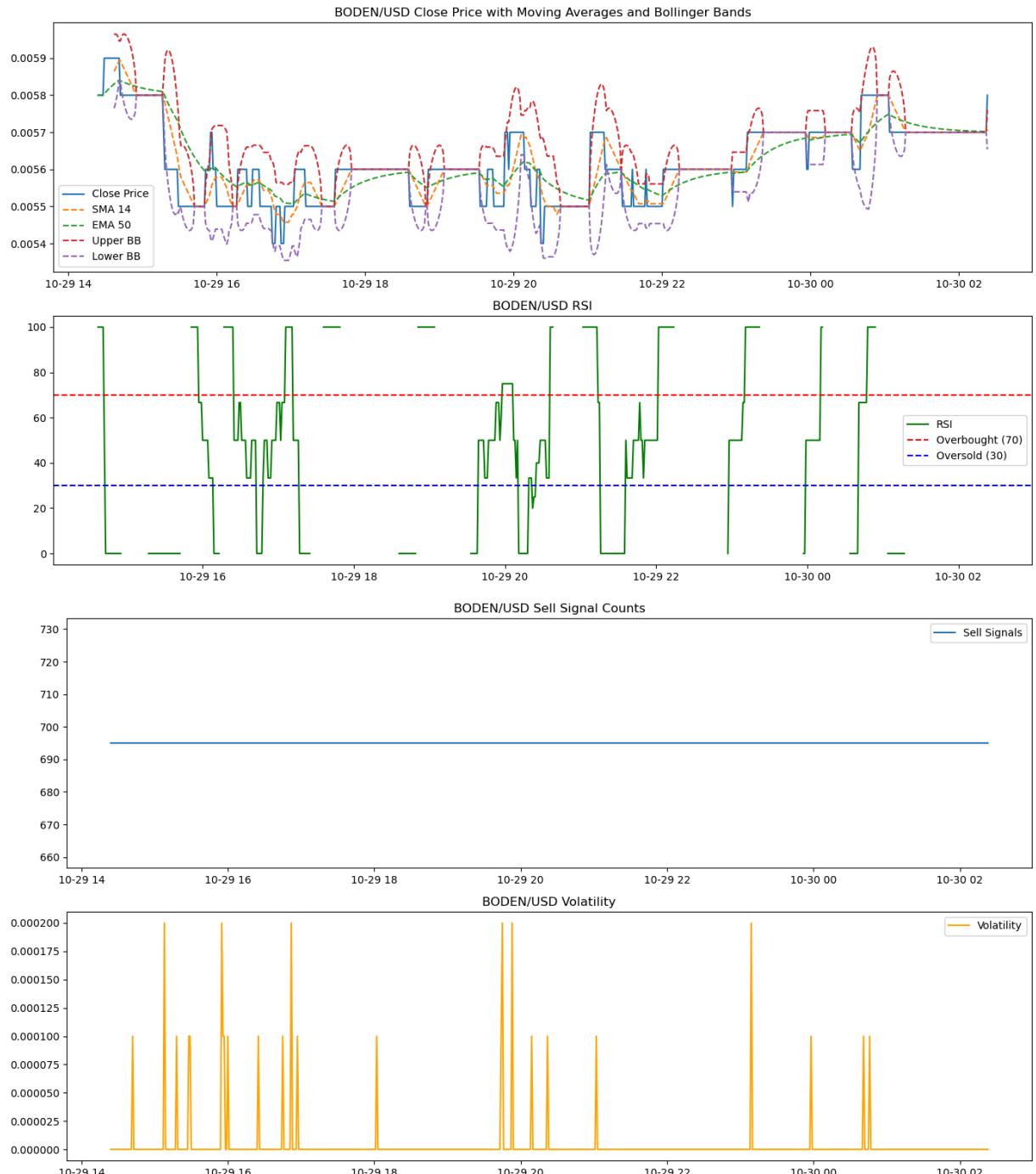
Data saved to CSV file: AUCTION_USD_data.csv



Data successfully saved to SQL table: BODEN_USD_data

SQL connection closed.

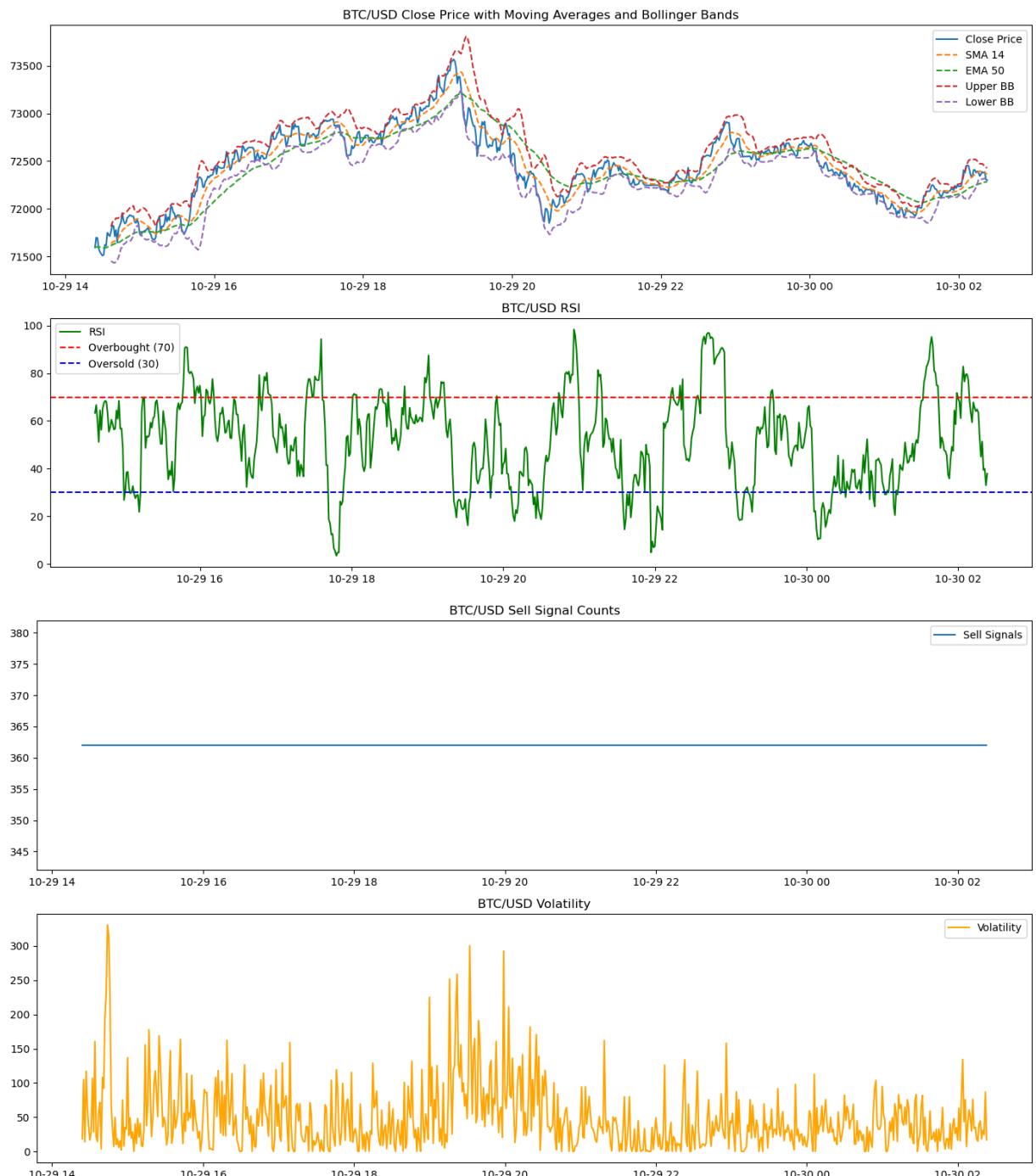
Data saved to CSV file: BODEN_USD_data.csv



Data successfully saved to SQL table: BTC_USD_data

SQL connection closed.

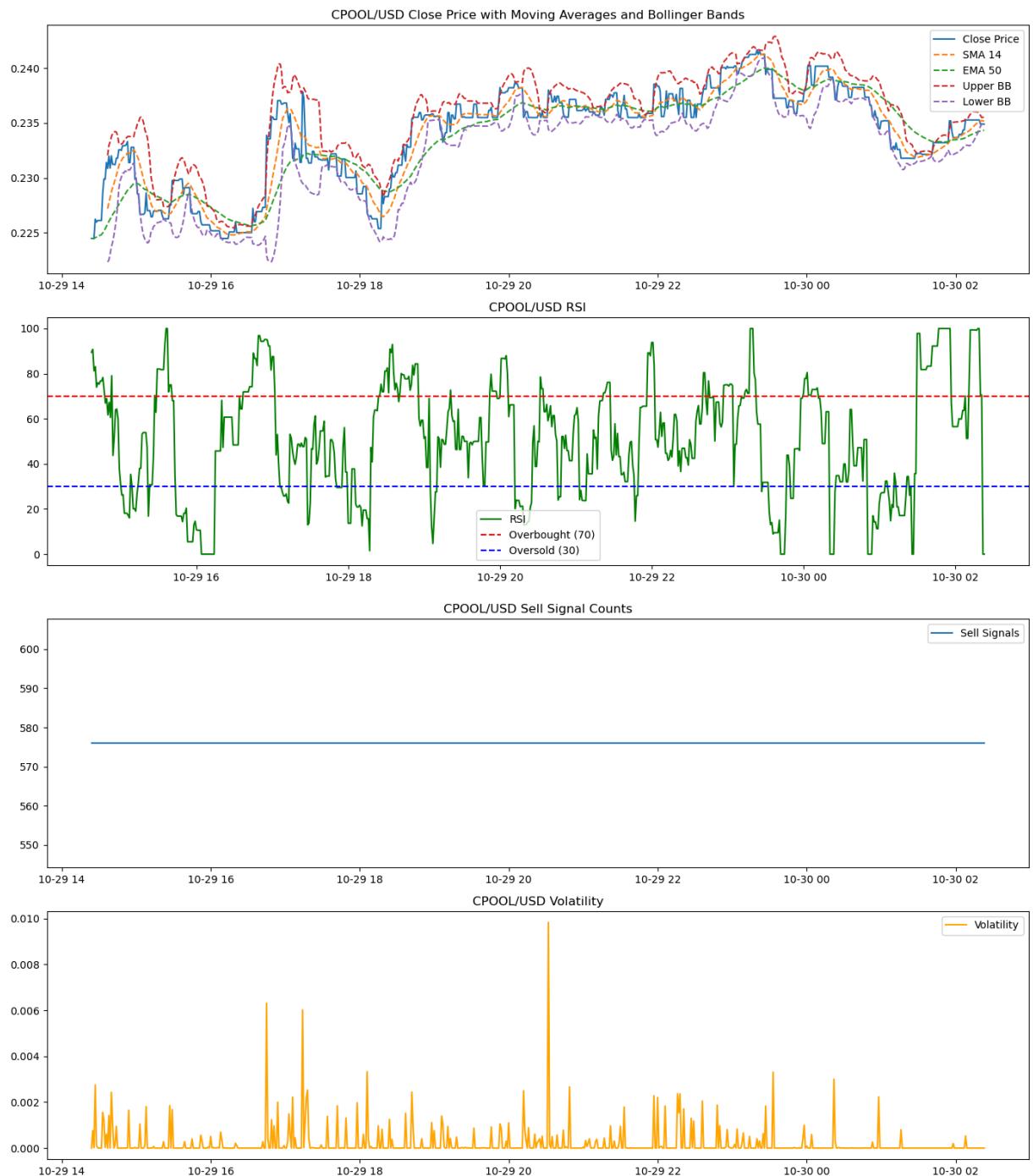
Data saved to CSV file: BTC_USD_data.csv



Data successfully saved to SQL table: CPOOL_USD_data

SQL connection closed.

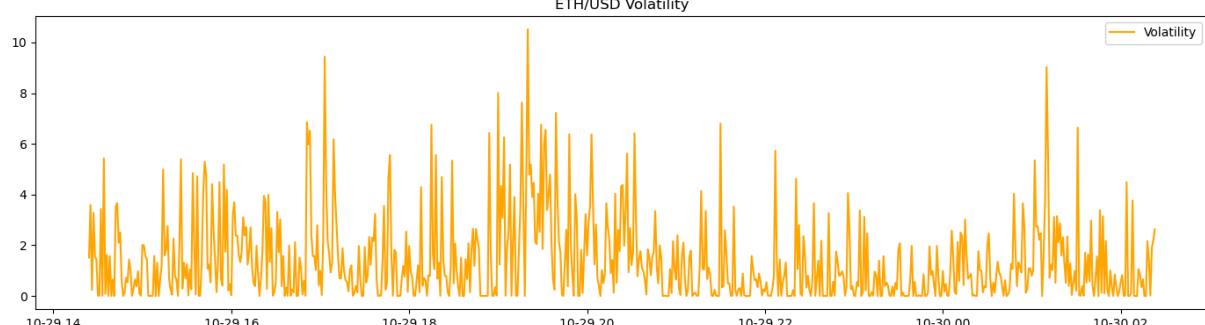
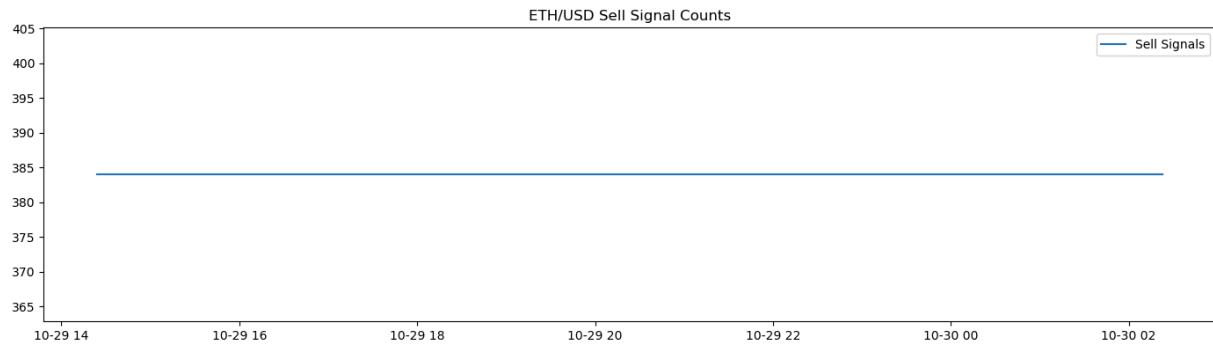
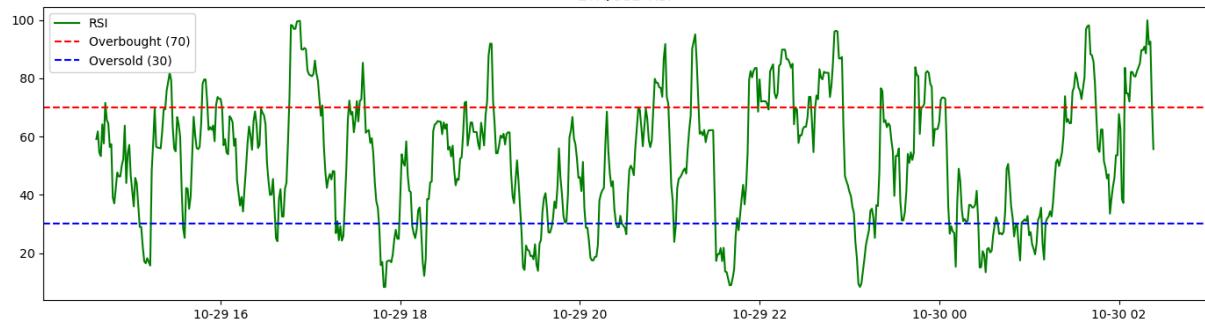
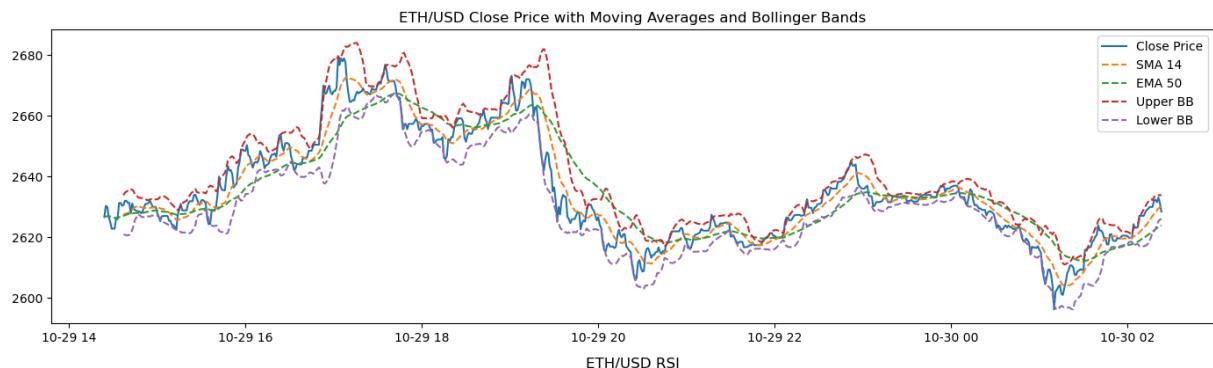
Data saved to CSV file: CPOOL_USD_data.csv



Data successfully saved to SQL table: ETH_USD_data

SQL connection closed.

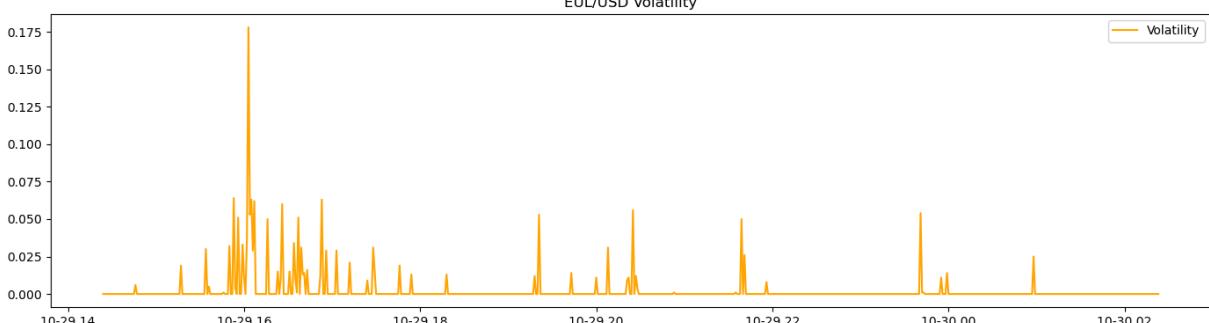
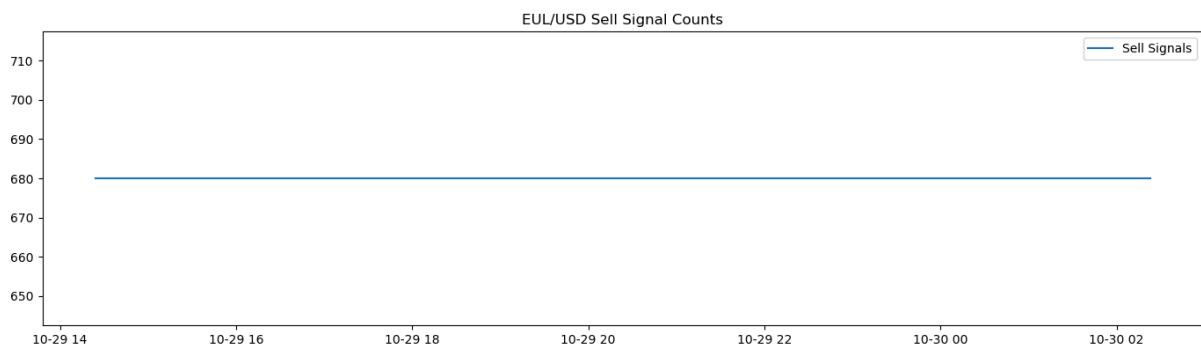
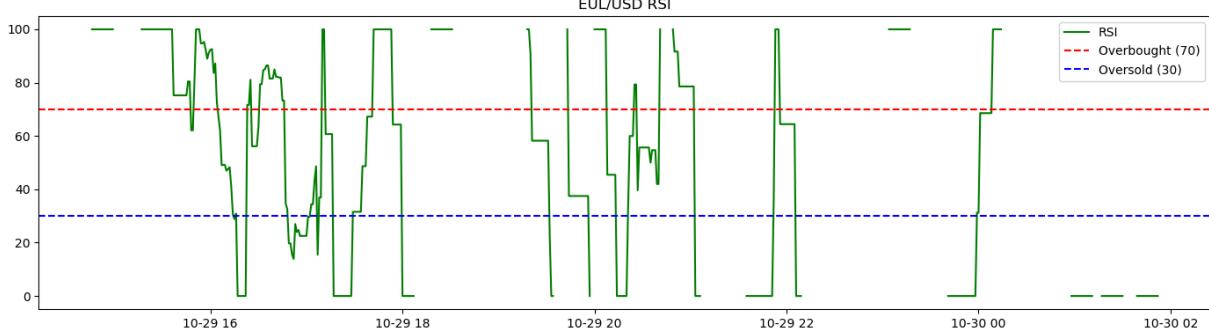
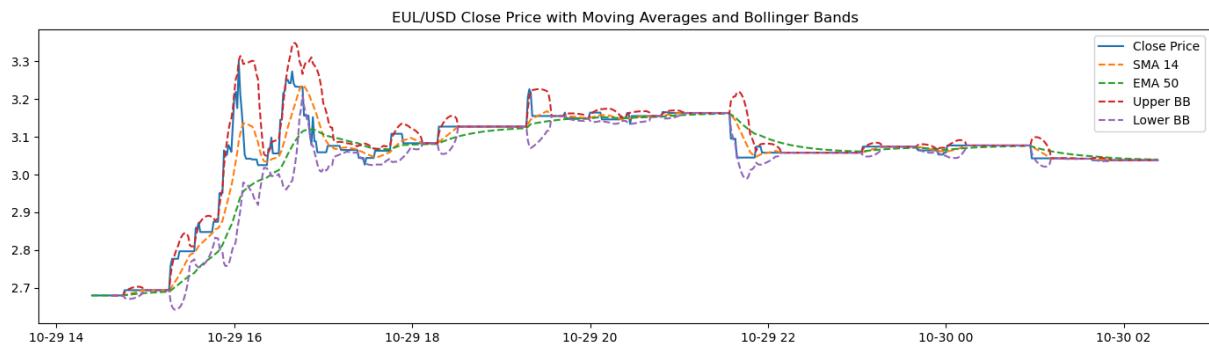
Data saved to CSV file: ETH_USD_data.csv



Data successfully saved to SQL table: EUL_USD_data

SQL connection closed.

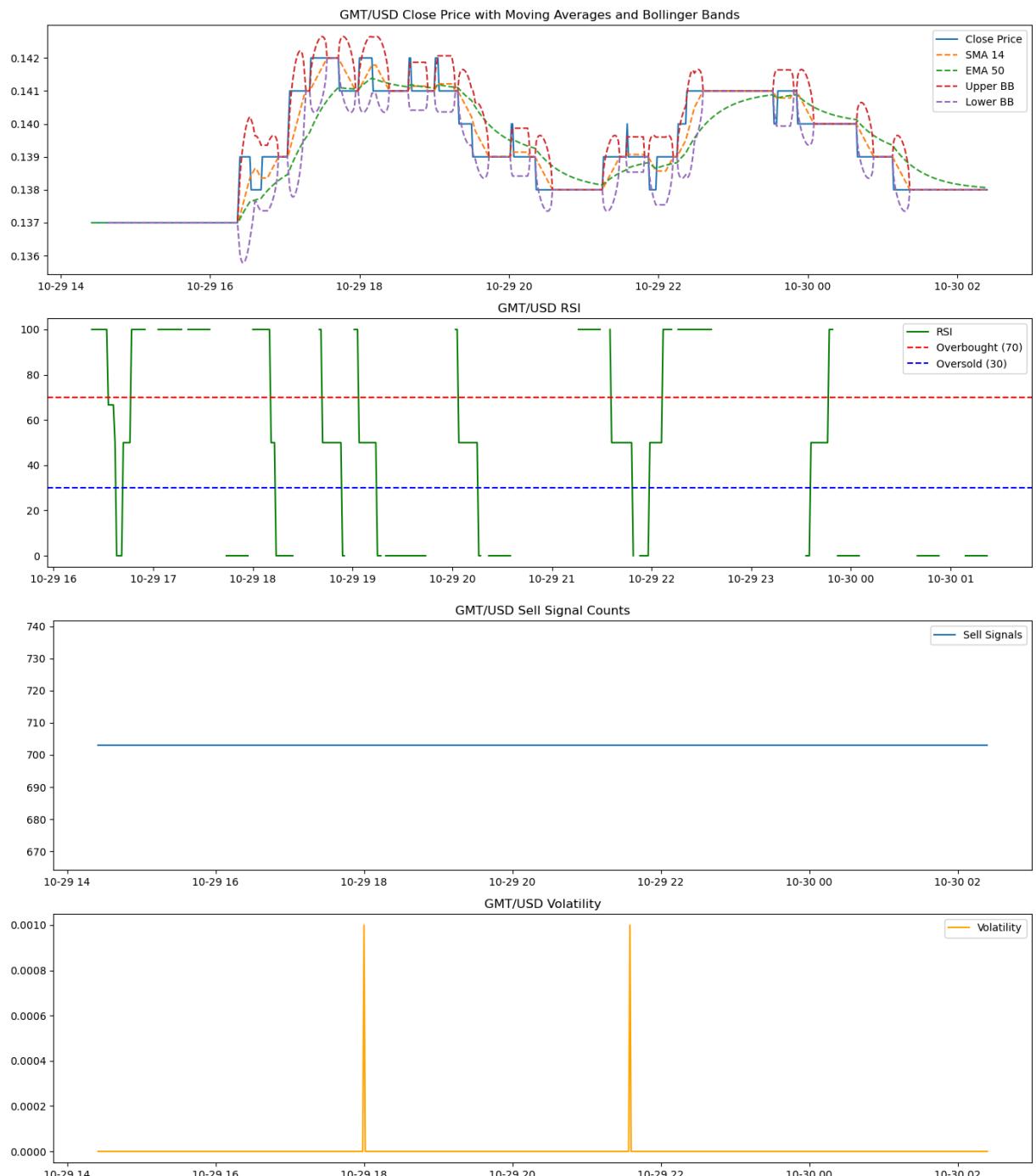
Data saved to CSV file: EUL_USD_data.csv



Data successfully saved to SQL table: GMT_USD_data

SQL connection closed.

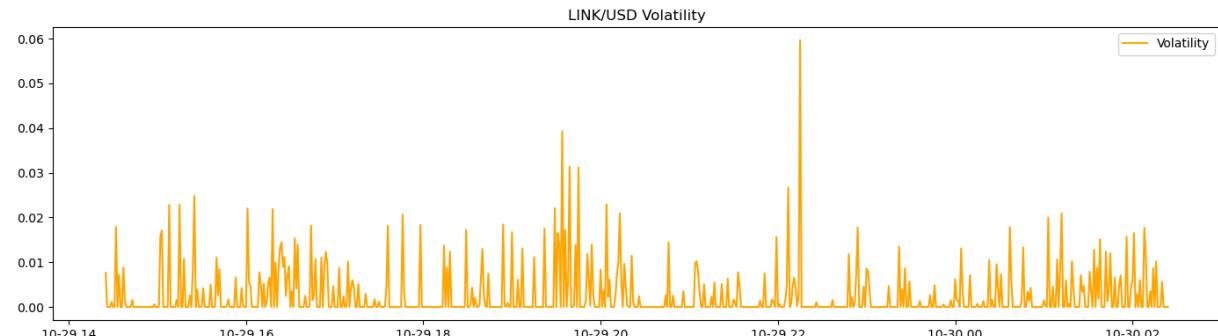
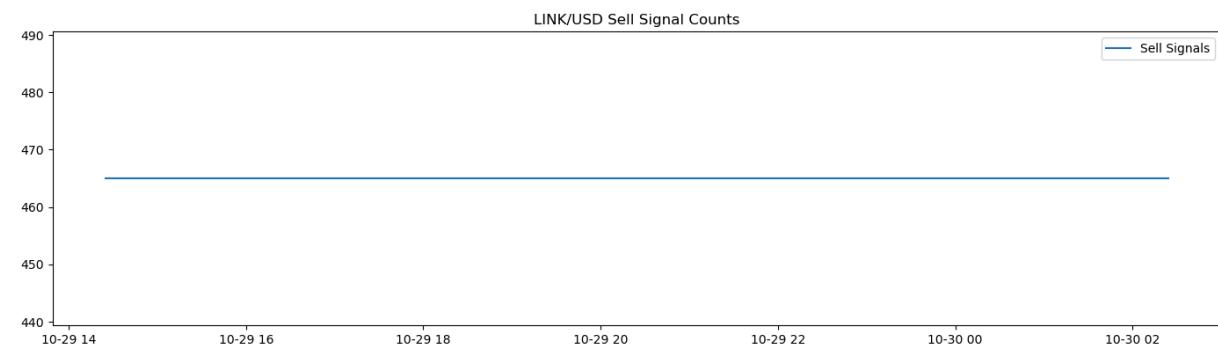
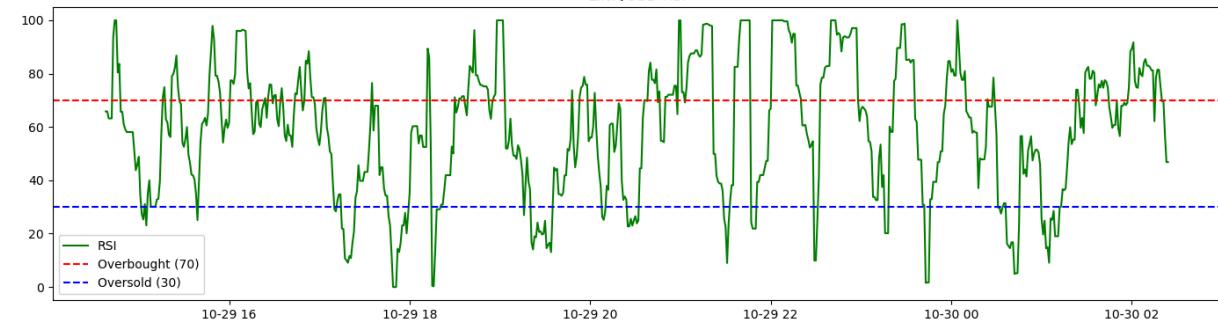
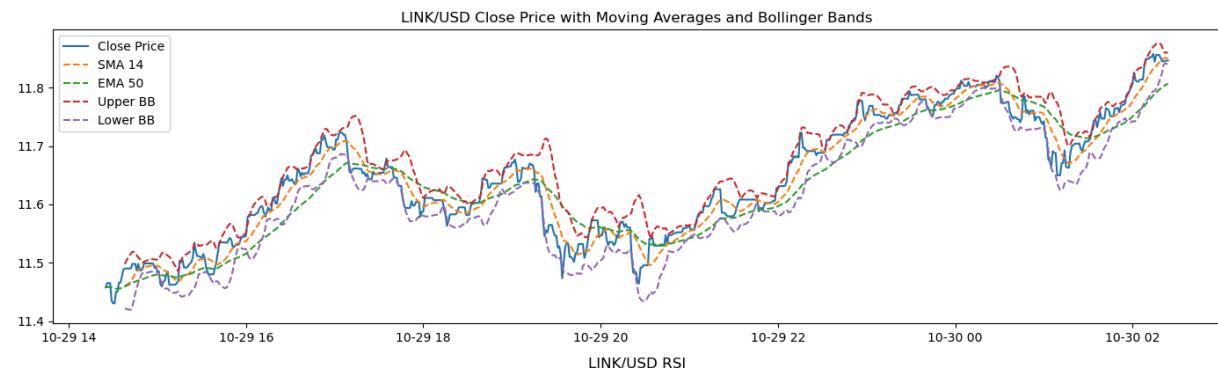
Data saved to CSV file: GMT_USD_data.csv



Data successfully saved to SQL table: LINK_USD_data

SQL connection closed.

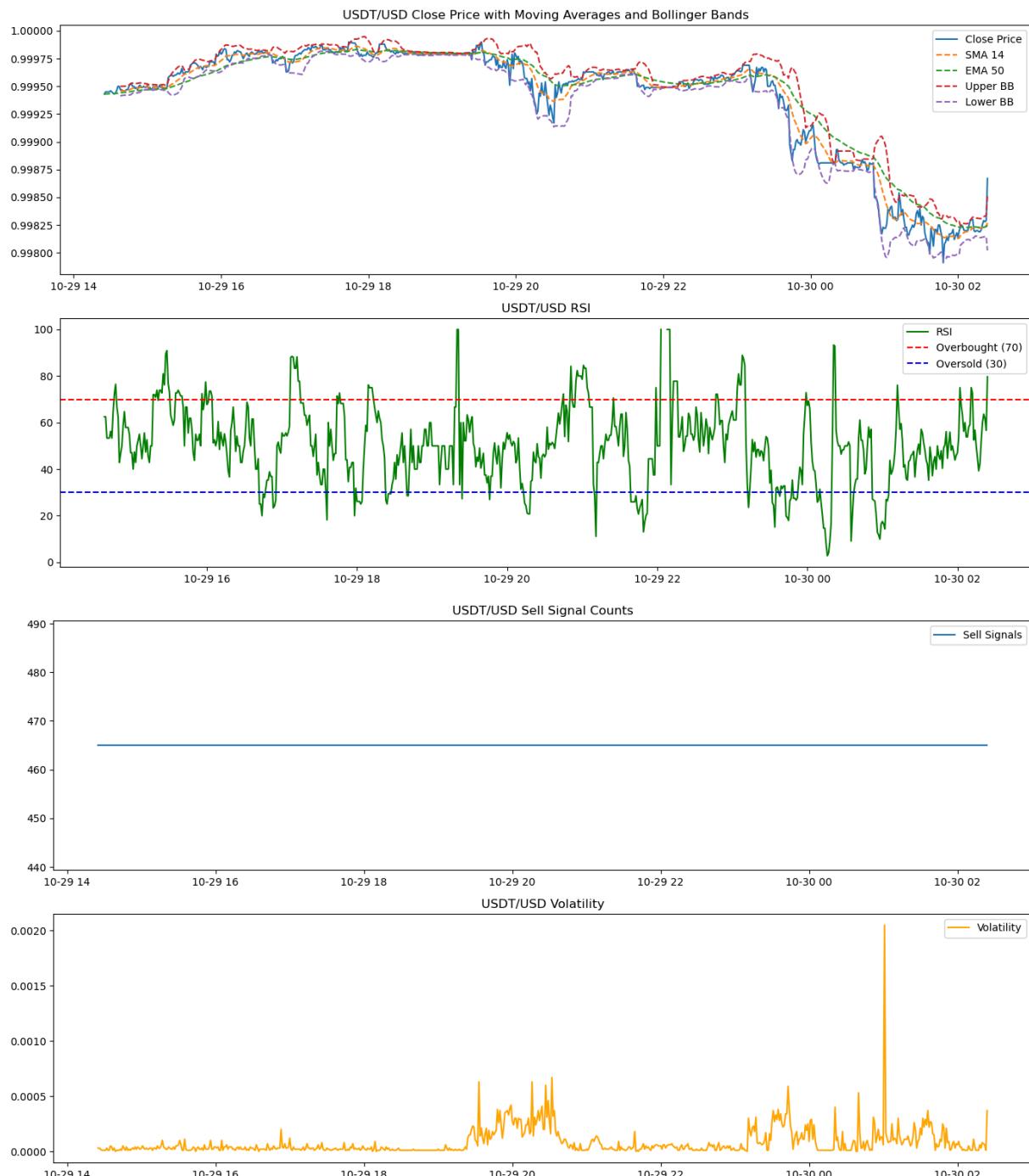
Data saved to CSV file: LINK_USD_data.csv



Data successfully saved to SQL table: USDT_USD_data

SQL connection closed.

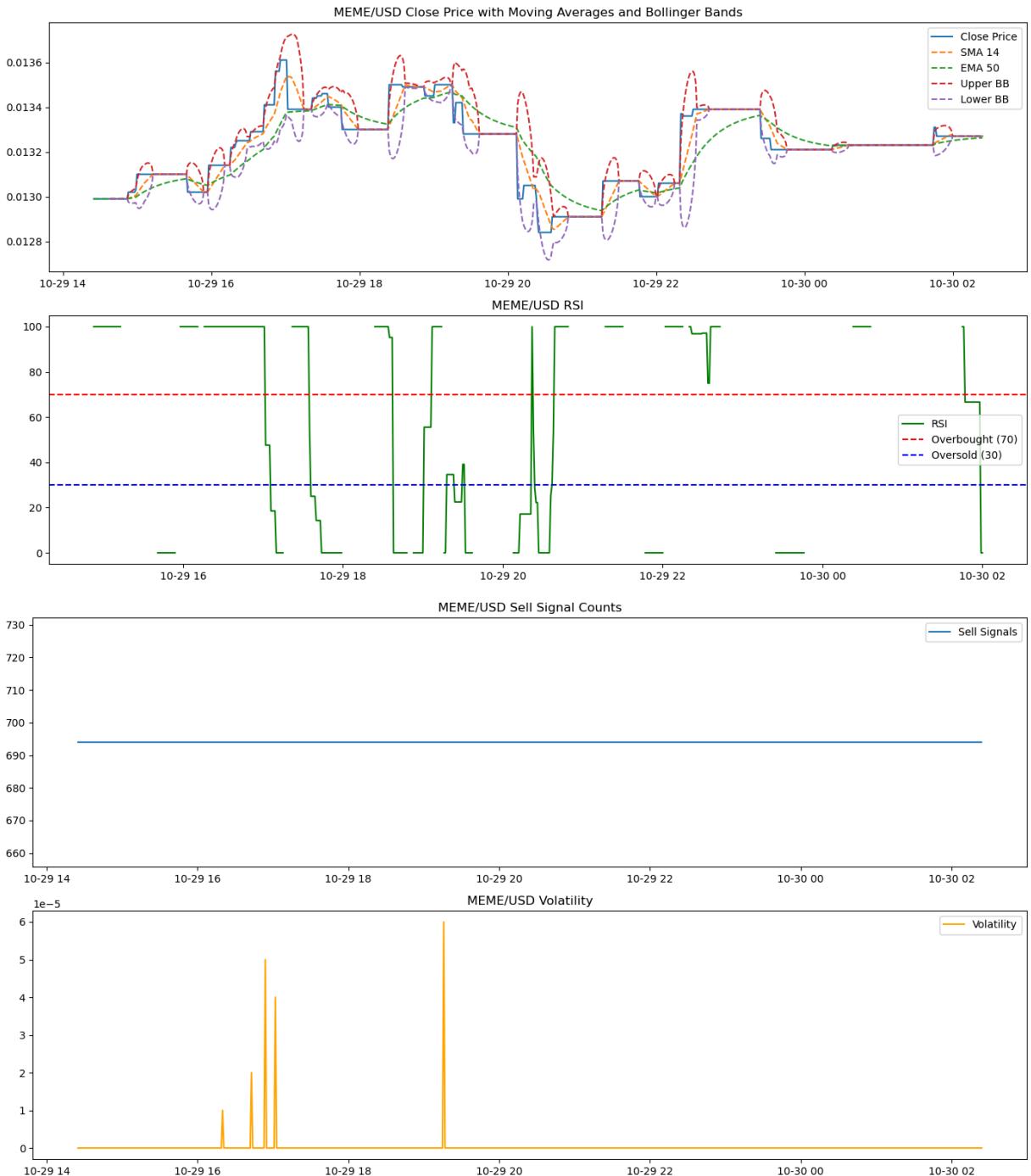
Data saved to CSV file: USDT_USD_data.csv



Data successfully saved to SQL table: MEME_USD_data

SQL connection closed.

Data saved to CSV file: MEME_USD_data.csv



Data successfully saved to SQL table: MNT_USD_data

SQL connection closed.

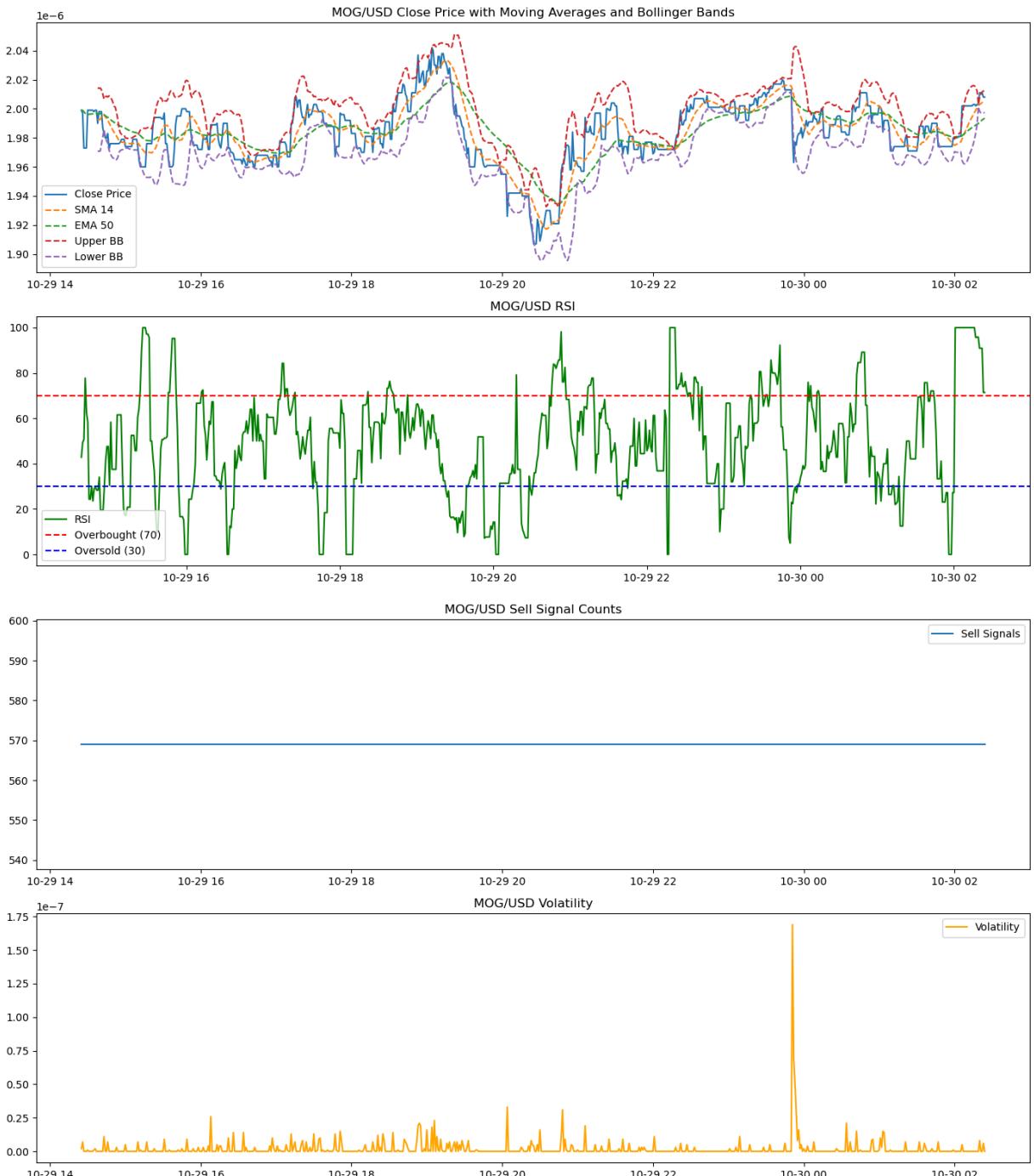
Data saved to CSV file: MNT_USD_data.csv



Data successfully saved to SQL table: MOG_USD_data

SQL connection closed.

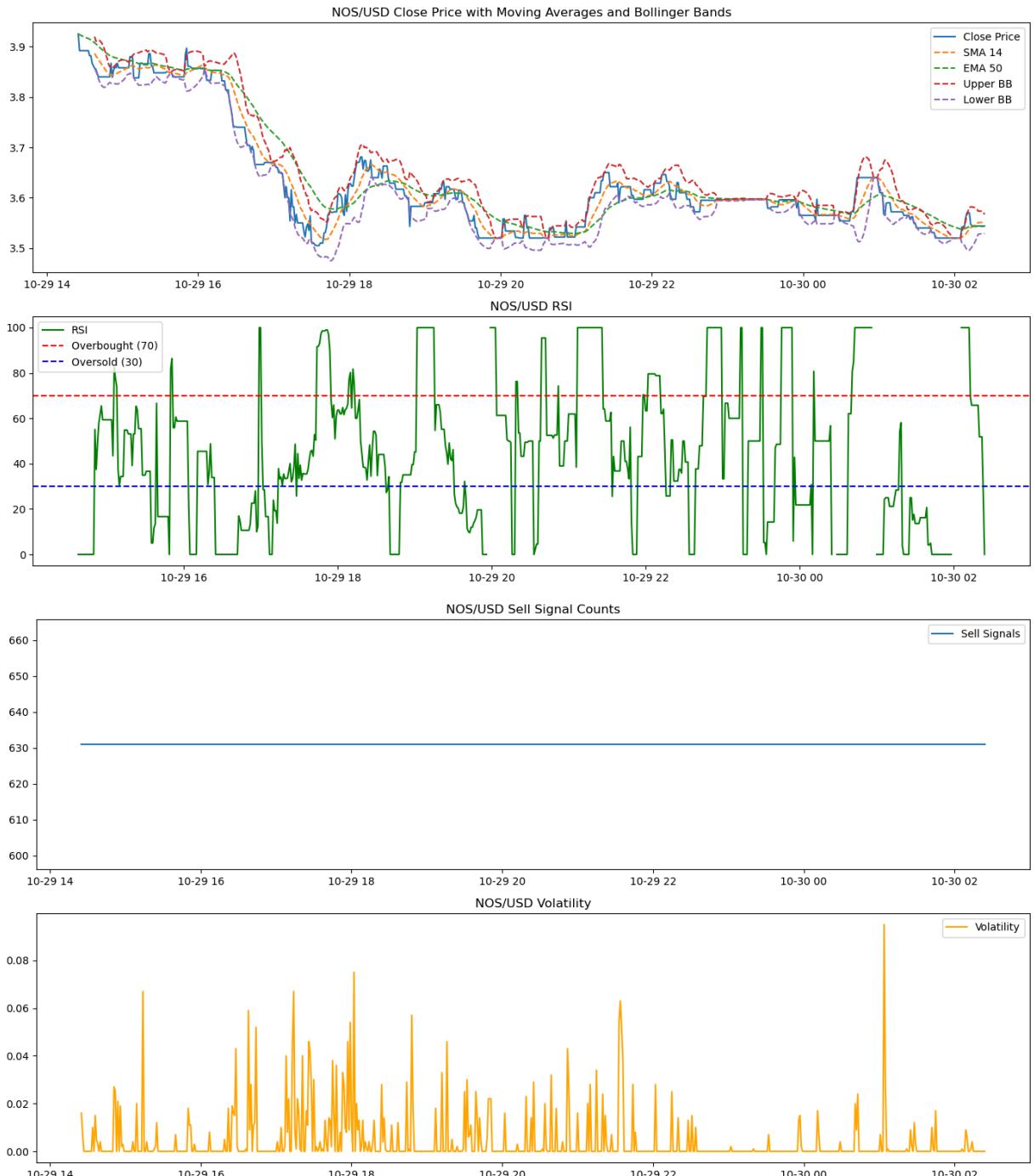
Data saved to CSV file: MOG_USD_data.csv



Data successfully saved to SQL table: NOS_USD_data

SQL connection closed.

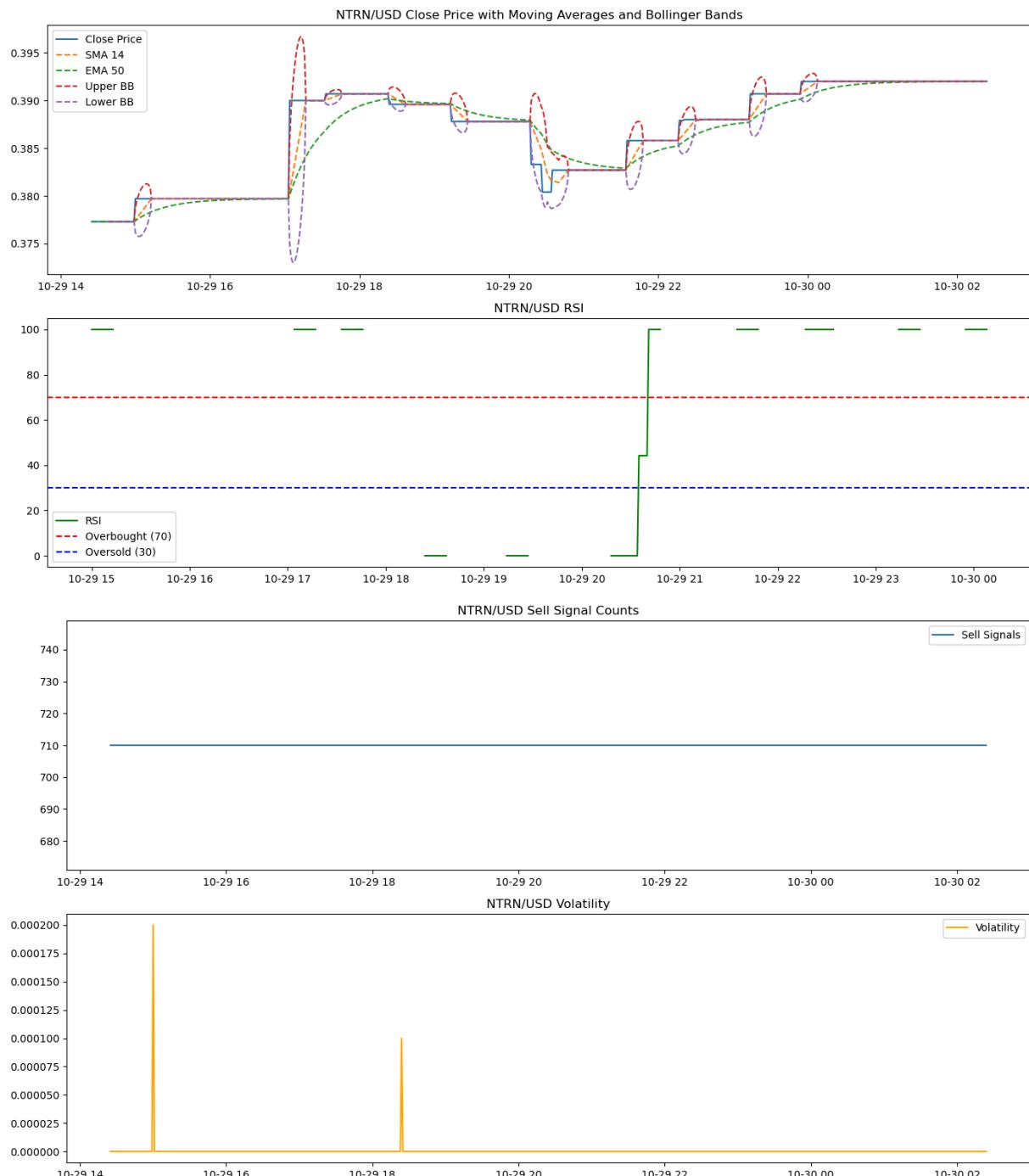
Data saved to CSV file: NOS_USD_data.csv



Data successfully saved to SQL table: NTRN_USD_data

SQL connection closed.

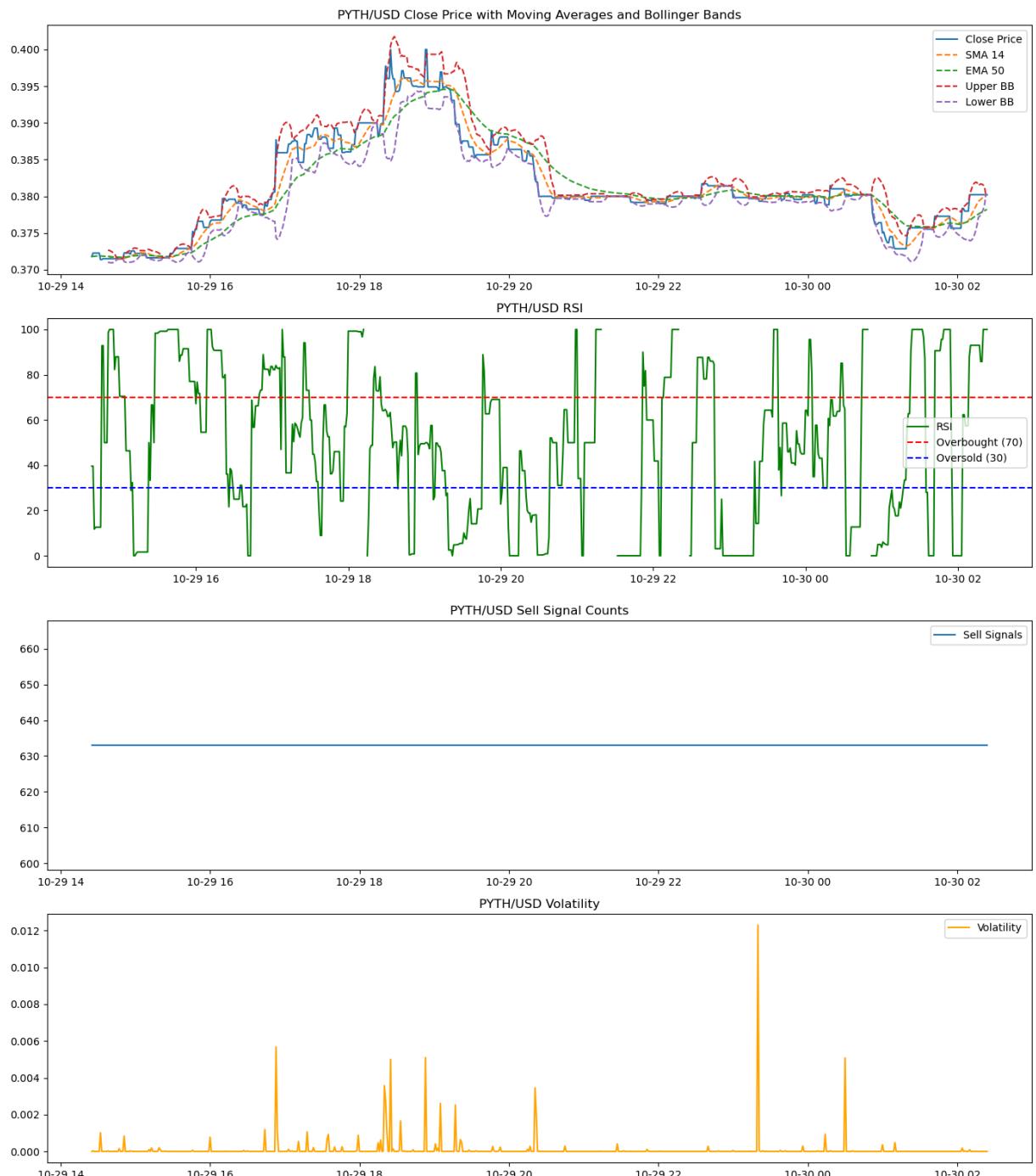
Data saved to CSV file: NTRN_USD_data.csv



Data successfully saved to SQL table: PYTH_USD_data

SQL connection closed.

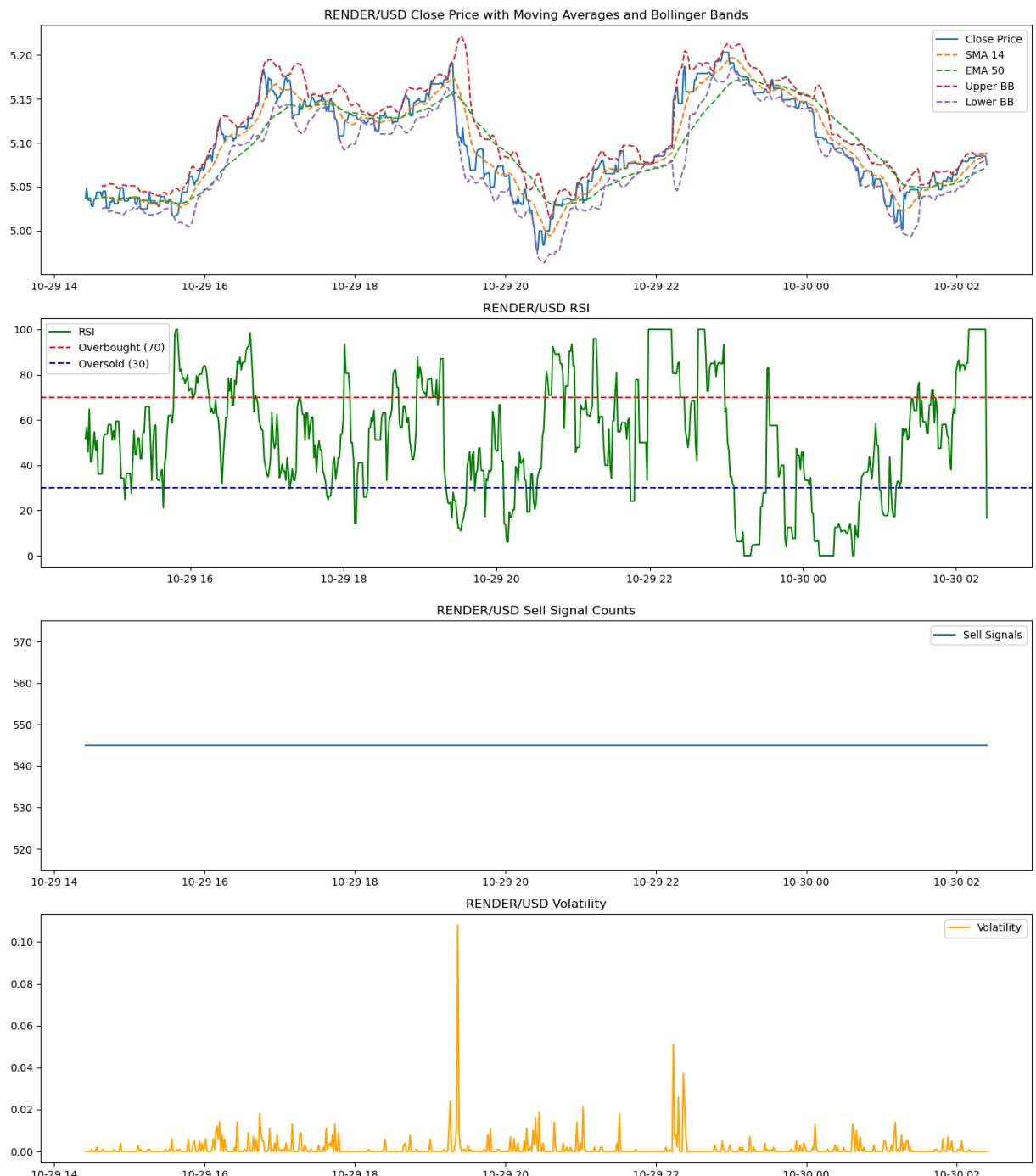
Data saved to CSV file: PYTH_USD_data.csv



Data successfully saved to SQL table: RENDER_USD_data

SQL connection closed.

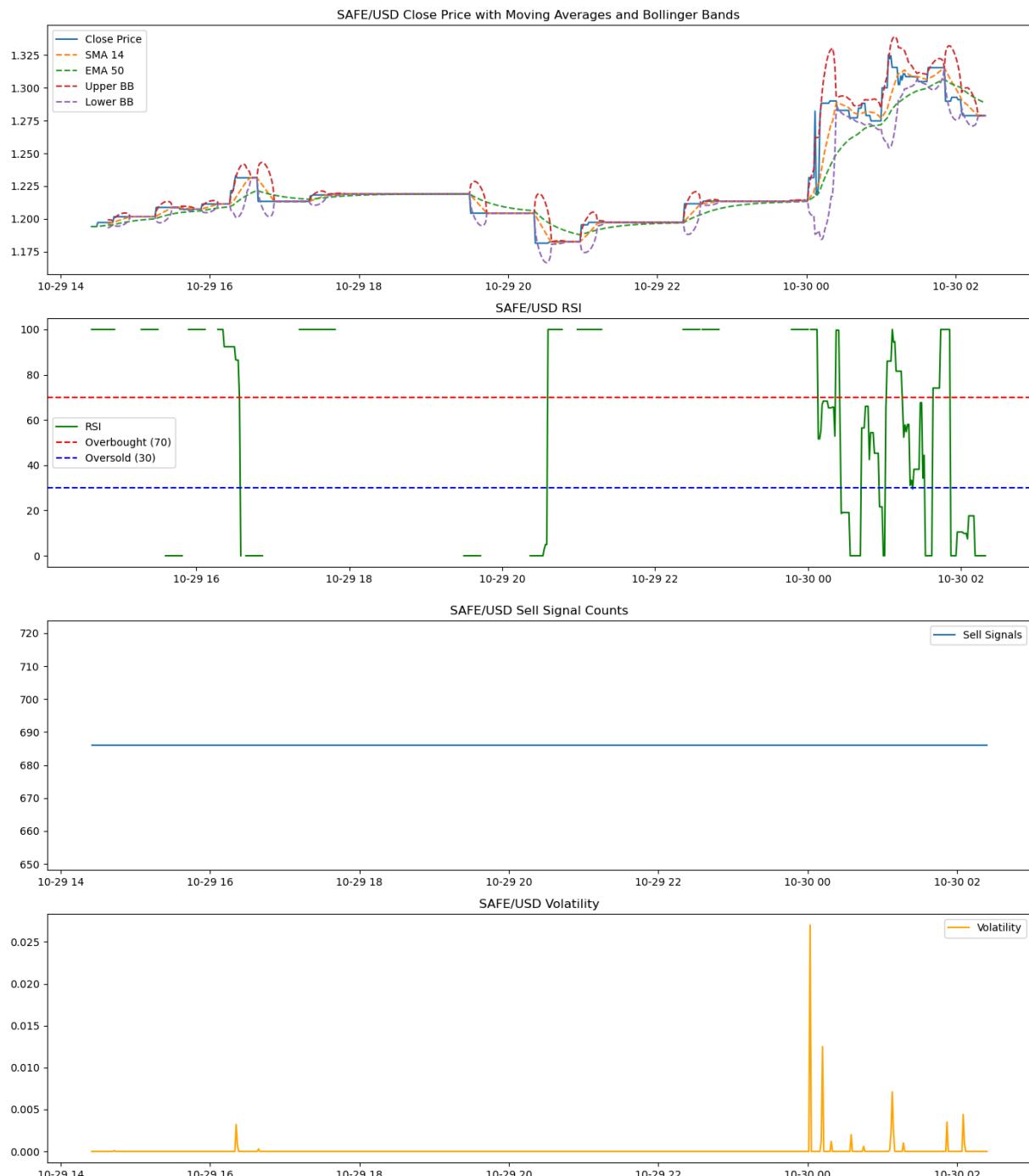
Data saved to CSV file: RENDER_USD_data.csv



Data successfully saved to SQL table: SAFE_USD_data

SQL connection closed.

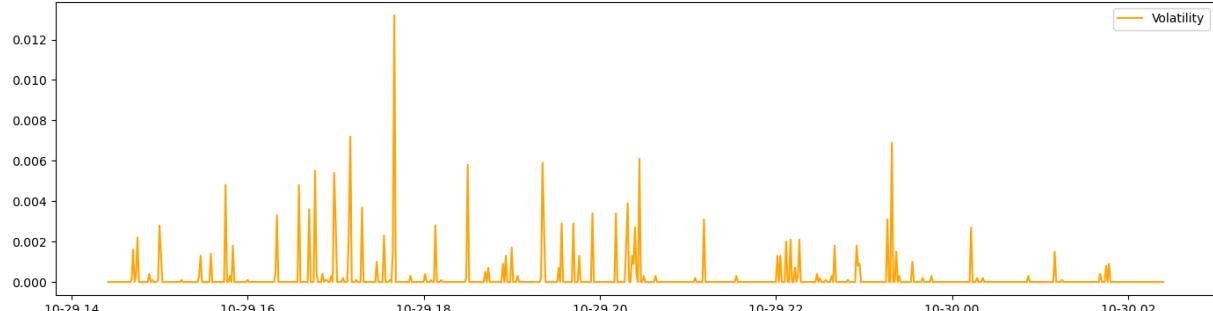
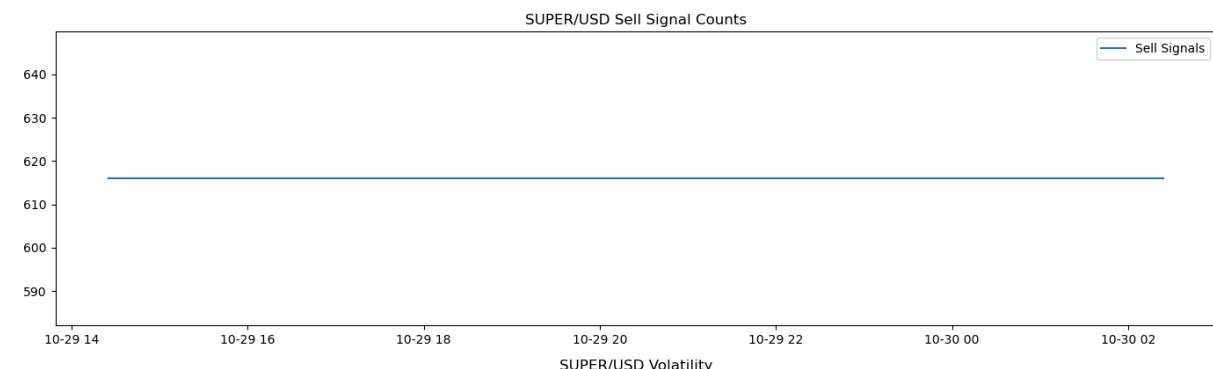
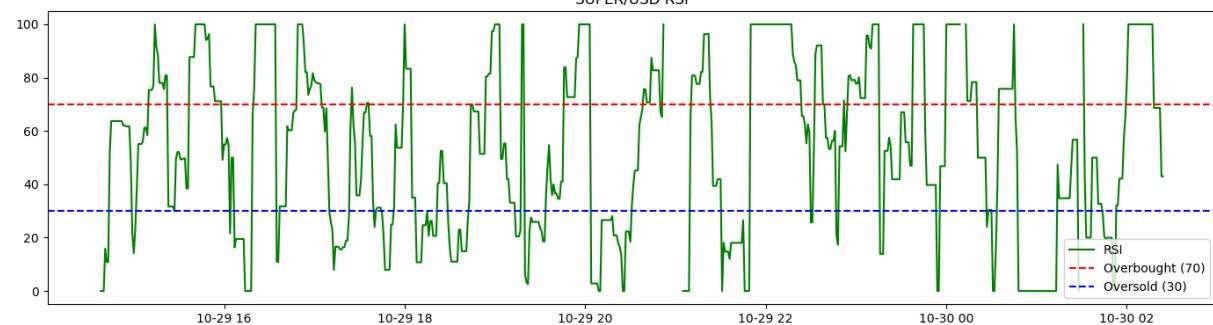
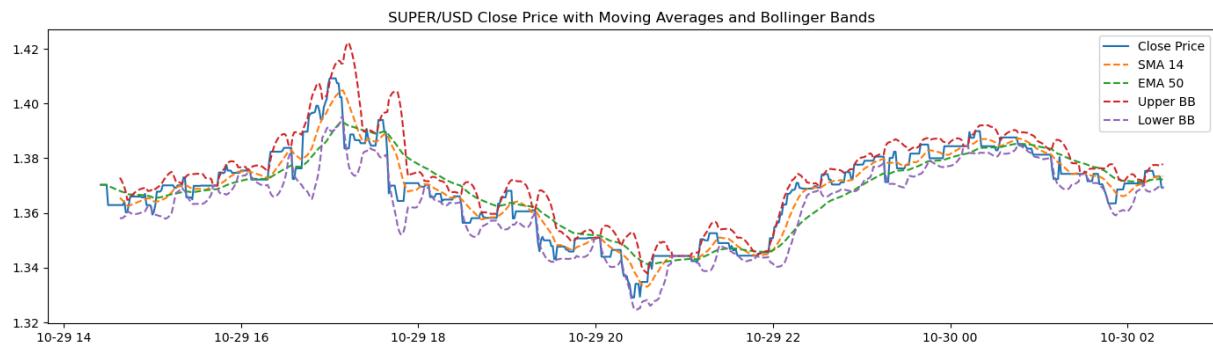
Data saved to CSV file: SAFE_USD_data.csv



Data successfully saved to SQL table: SUPER_USD_data

SQL connection closed.

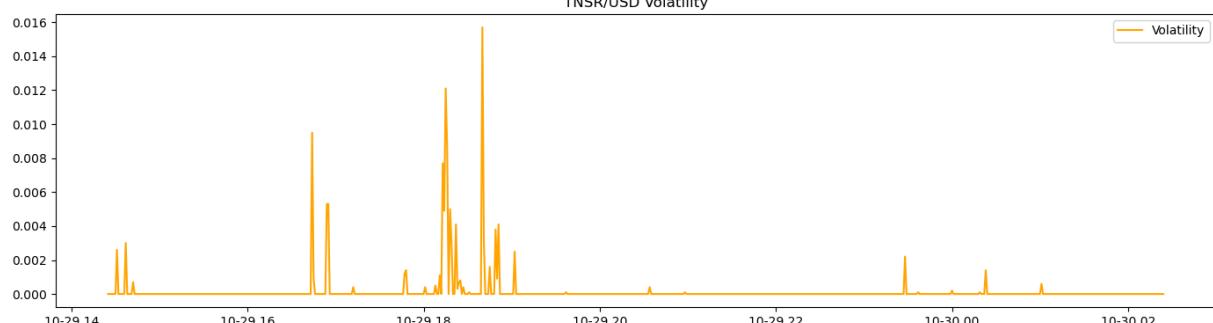
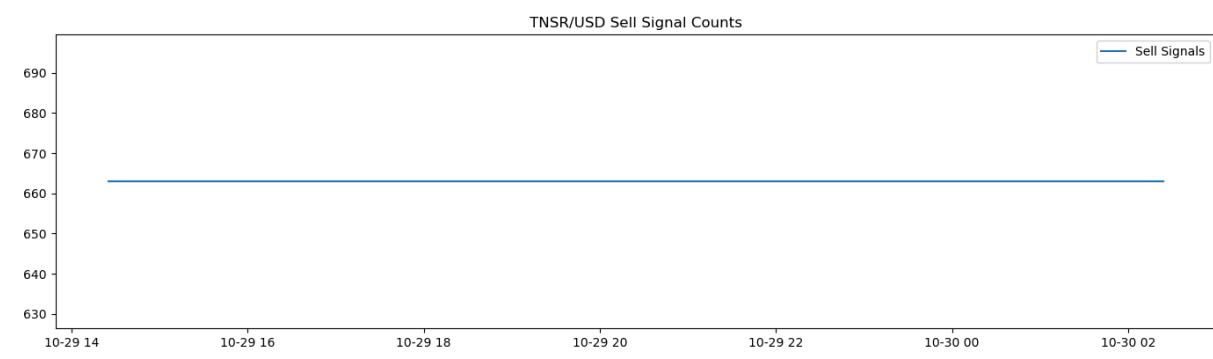
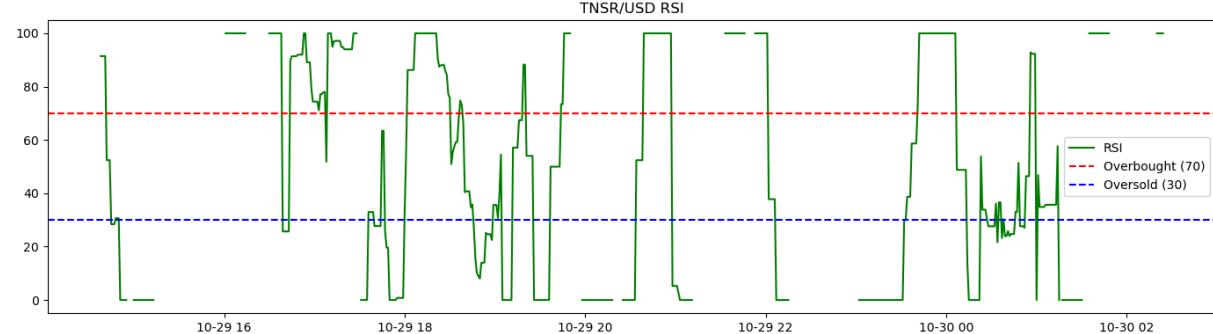
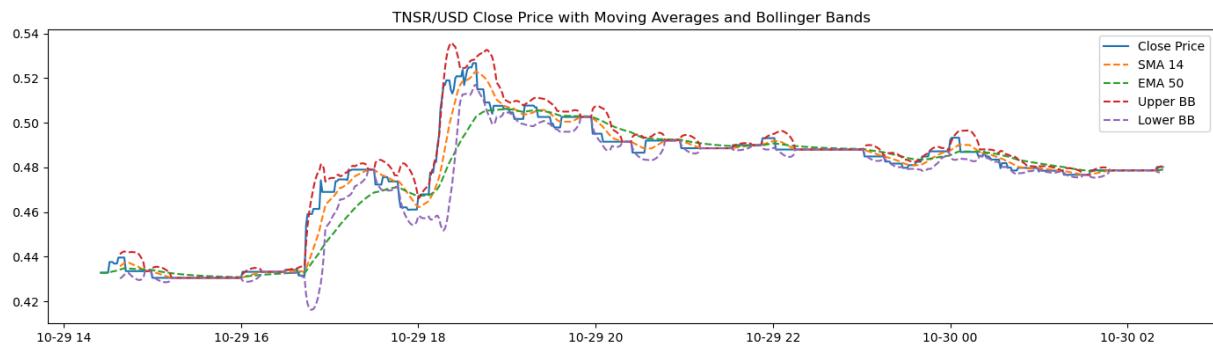
Data saved to CSV file: SUPER_USD_data.csv



Data successfully saved to SQL table: TNSR_USD_data

SQL connection closed.

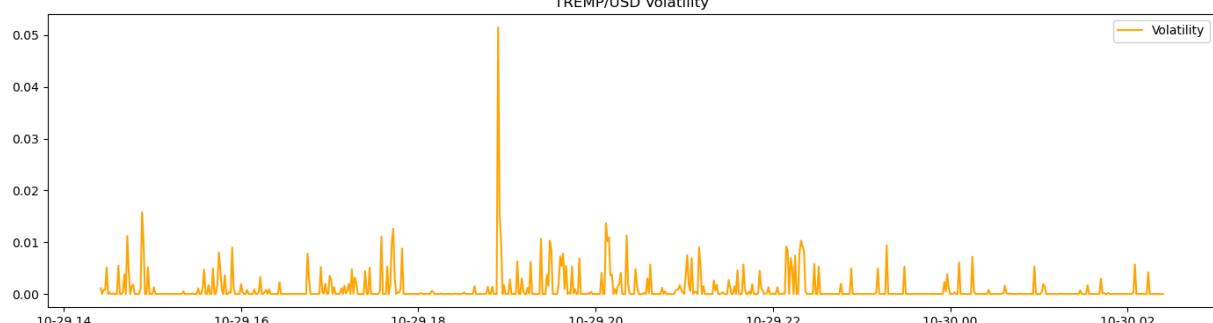
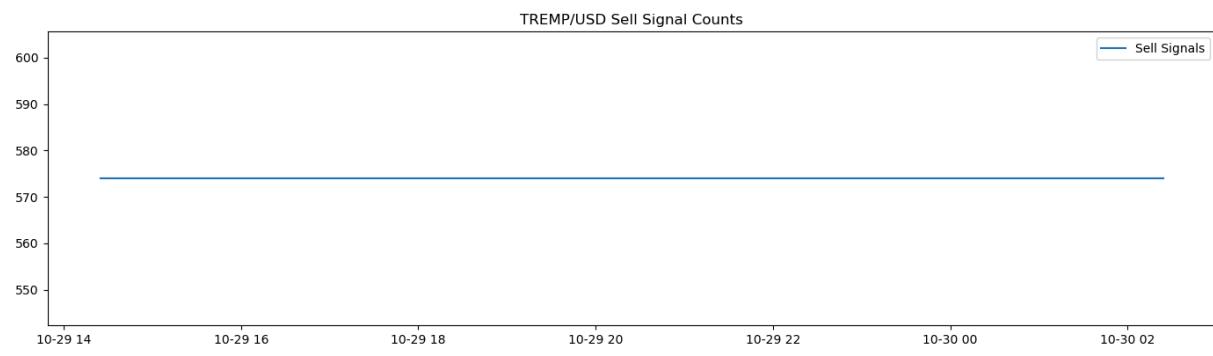
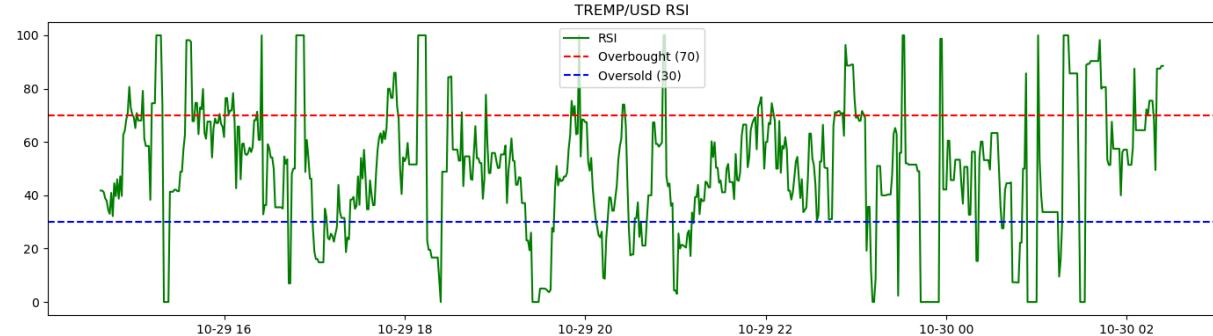
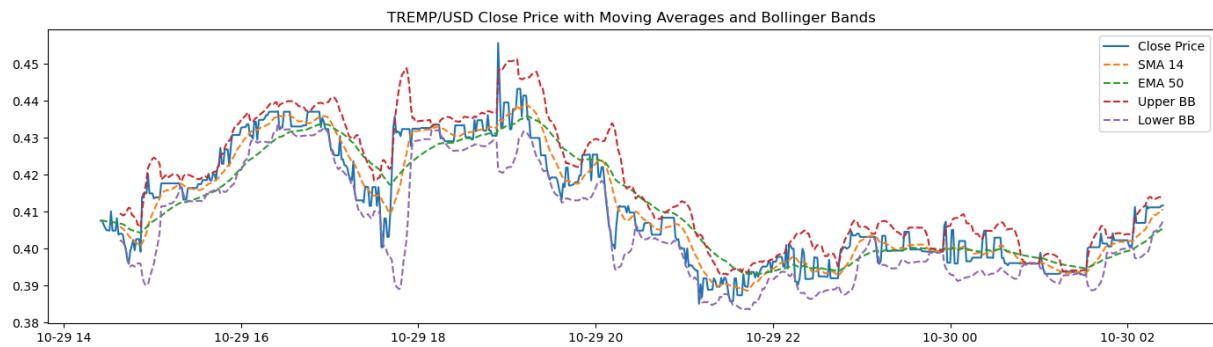
Data saved to CSV file: TNSR_USD_data.csv



Data successfully saved to SQL table: TREMP_USD_data

SQL connection closed.

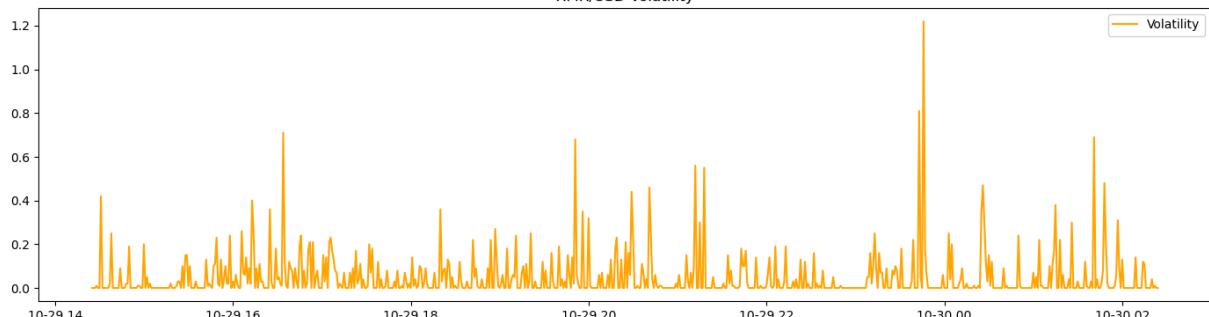
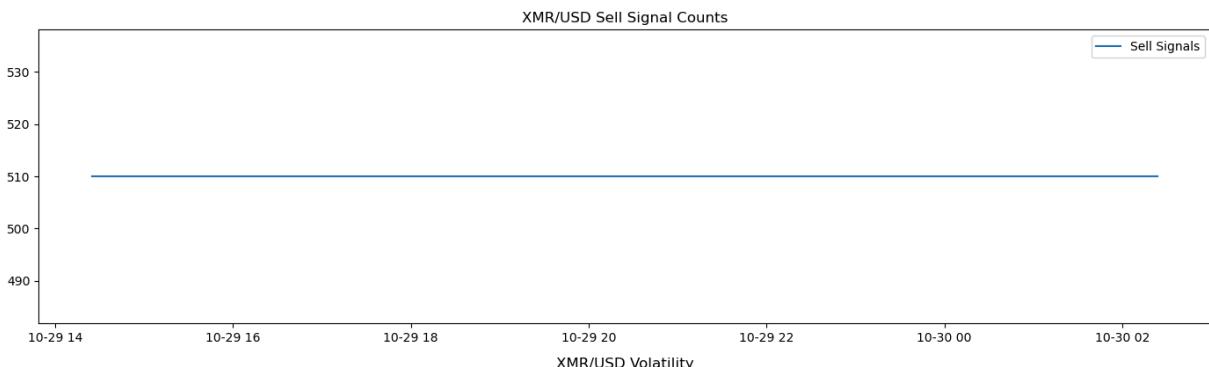
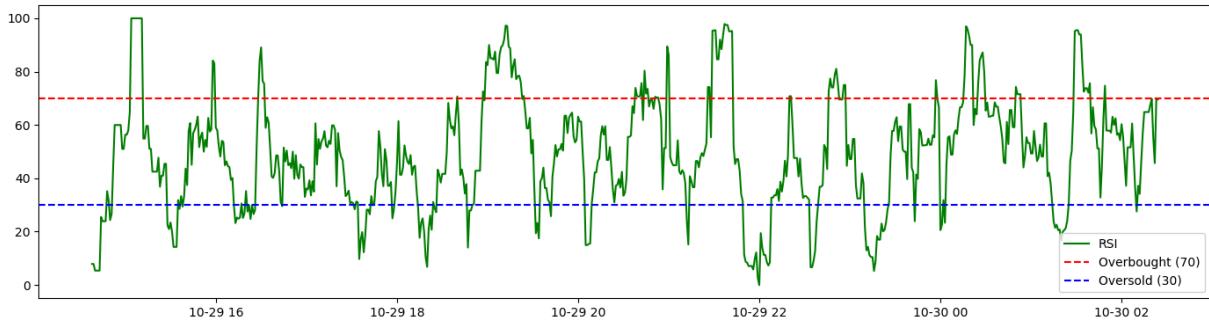
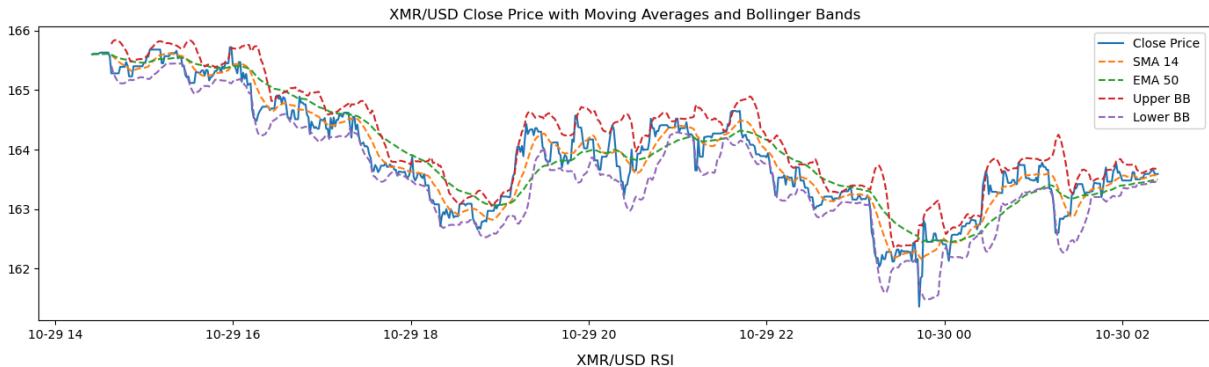
Data saved to CSV file: TREMP_USD_data.csv



Data successfully saved to SQL table: XMR_USD_data

SQL connection closed.

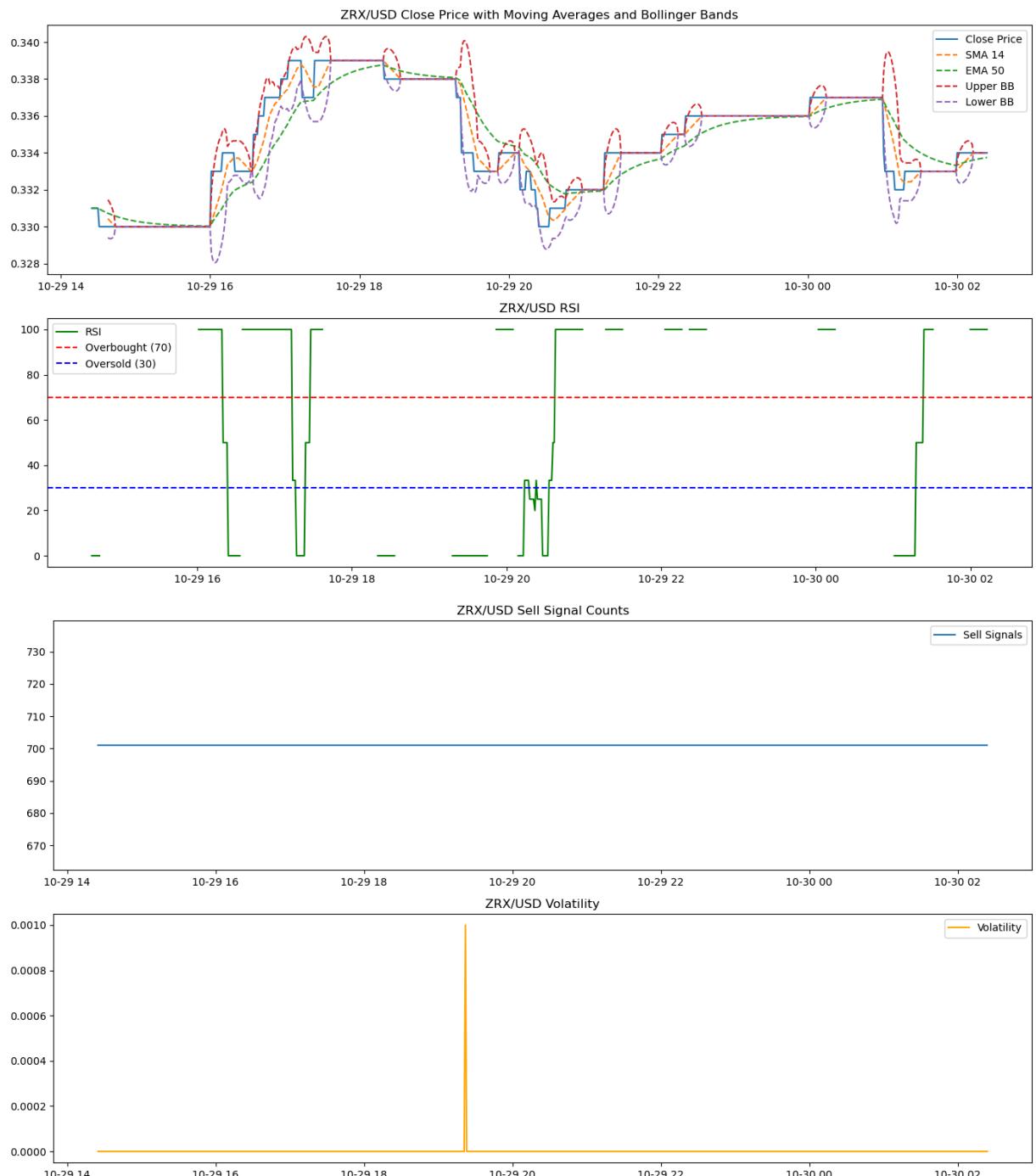
Data saved to CSV file: XMR_USD_data.csv



Data successfully saved to SQL table: ZRX_USD_data

SQL connection closed.

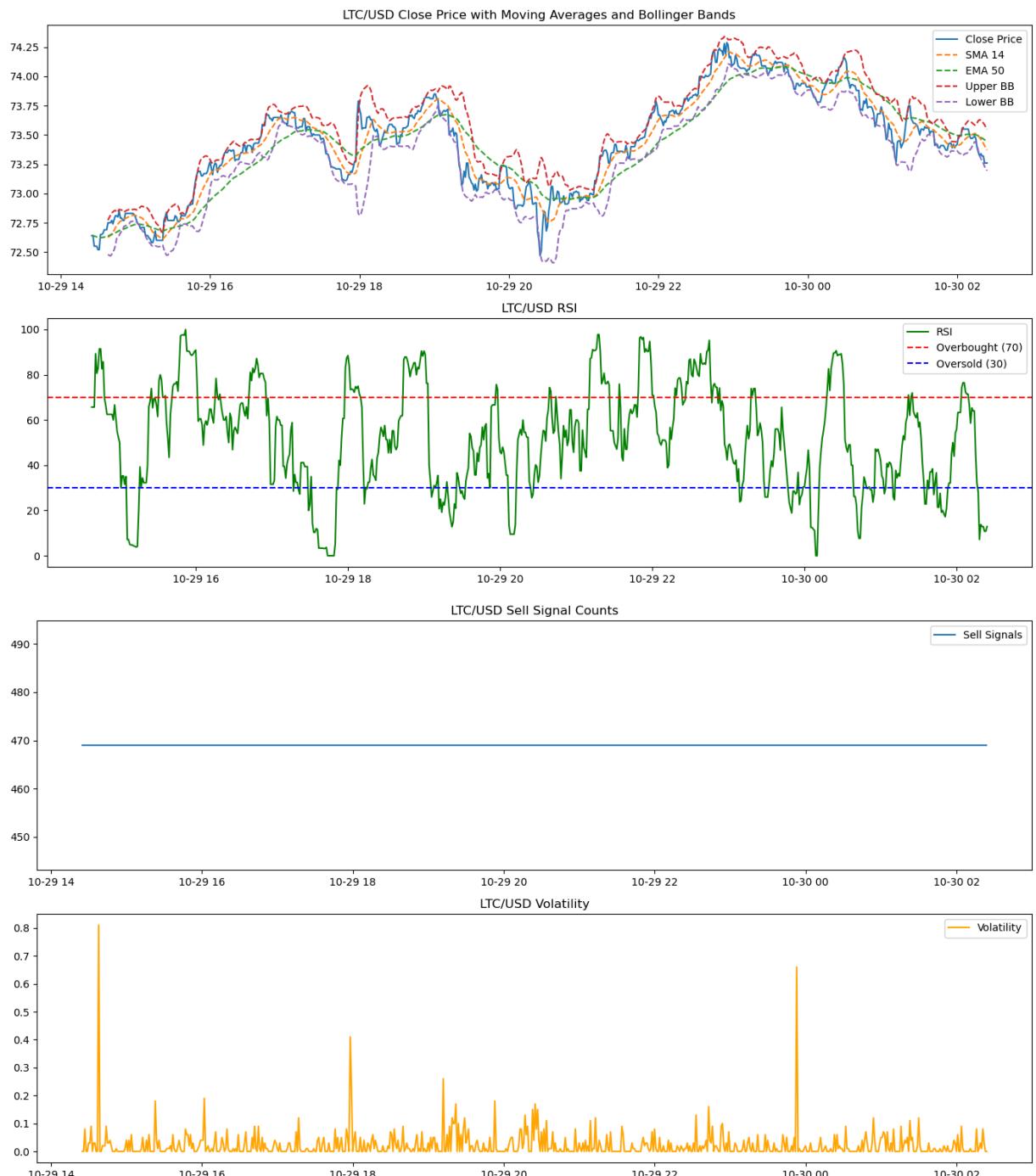
Data saved to CSV file: ZRX_USD_data.csv



Data successfully saved to SQL table: LTC_USD_data

SQL connection closed.

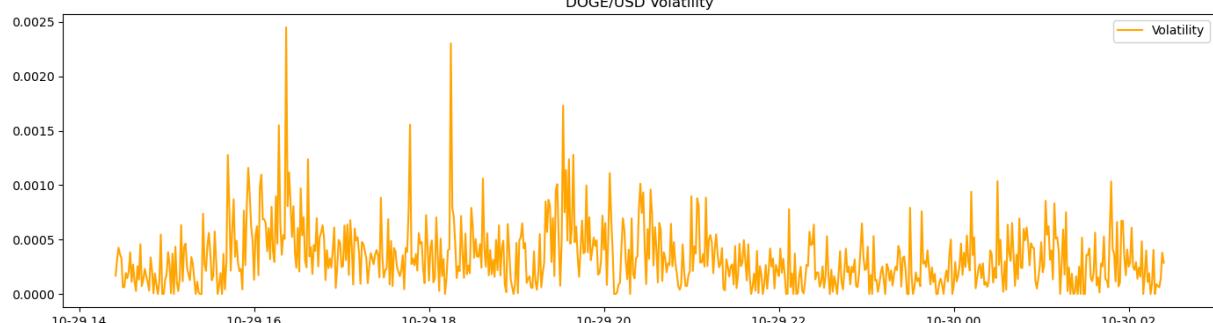
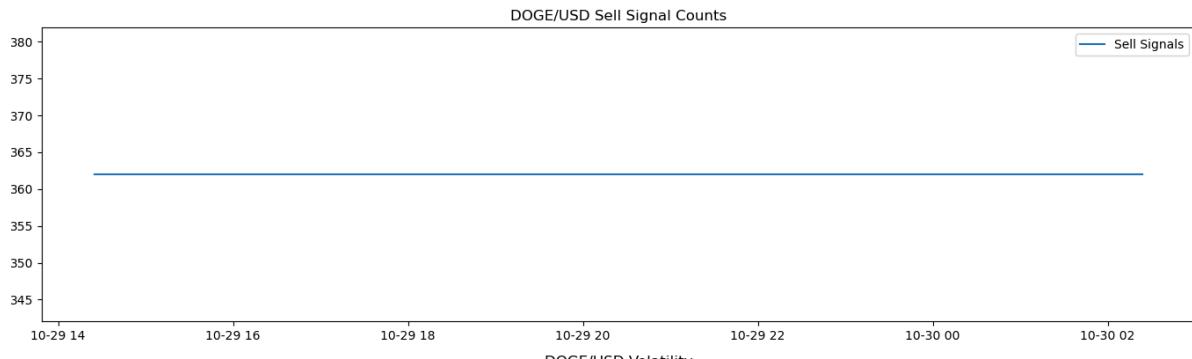
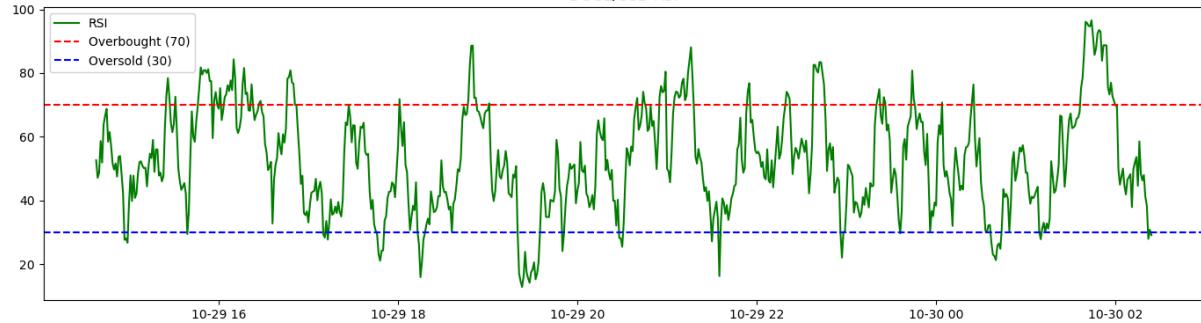
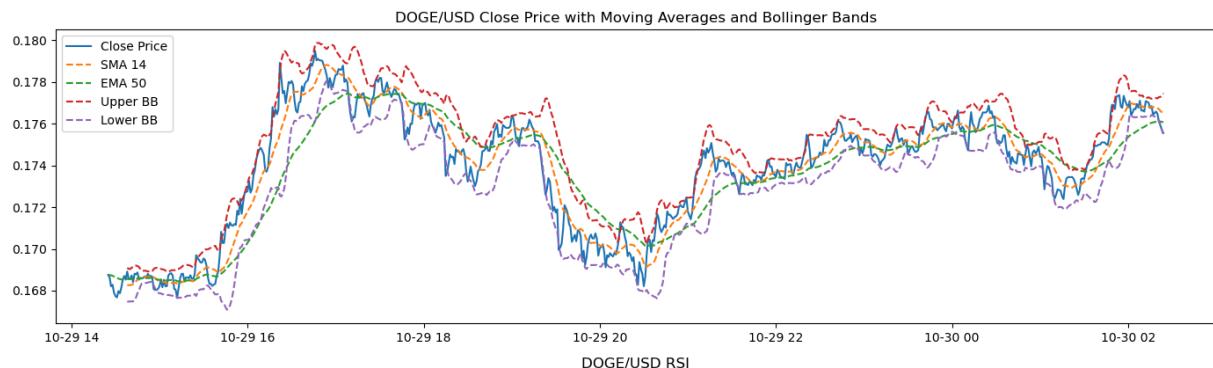
Data saved to CSV file: LTC_USD_data.csv



Data successfully saved to SQL table: DOGE_USD_data

SQL connection closed.

Data saved to CSV file: DOGE_USD_data.csv



Top 10 Symbols with Most 'SELL' Signals:

AUCTION/USD: 713 'SELL' signals
NTRN/USD: 710 'SELL' signals
MNT/USD: 709 'SELL' signals
GMT/USD: 703 'SELL' signals
ZRX/USD: 701 'SELL' signals
BODEN/USD: 695 'SELL' signals
MEME/USD: 694 'SELL' signals
SAFE/USD: 686 'SELL' signals
EUL/USD: 680 'SELL' signals
TNSR/USD: 663 'SELL' signals

Risk Classification:

ADA/USD: Low Risk
APE/USD: Low Risk
AUCTION/USD: Unknown Risk
BODEN/USD: Low Risk
BTC/USD: High Risk
CPOOL/USD: Low Risk
ETH/USD: High Risk
EUL/USD: Low Risk
GMT/USD: Low Risk
LINK/USD: Low Risk
USDT/USD: Low Risk
MEME/USD: Low Risk
MNT/USD: Low Risk
MOG/USD: Low Risk
NOS/USD: Low Risk
NTRN/USD: Low Risk
PYTH/USD: Low Risk
RENDER/USD: Low Risk
SAFE/USD: Low Risk
SUPER/USD: Low Risk
TNSR/USD: Low Risk
TREMP/USD: Low Risk
XMR/USD: Low Risk
ZRX/USD: Low Risk
LTC/USD: Low Risk
DOGE/USD: Low Risk

In [222...]

```
import dash
from dash import dcc, html
import plotly.express as px
import plotly.graph_objects as go
import pandas as pd

# Load the crypto data and handle the possibility of None
all_crypto_data = main() # Load data from the main script
if not isinstance(all_crypto_data, dict):
    all_crypto_data = {} # Ensure all_crypto_data is a dictionary if main() fails

# function to calculate volatility (standard deviation of returns)
def calculate_volatility(df):
    df['Returns'] = df['Close'].pct_change()
    return df['Returns'].std()
```

```

# Preprocess data to add volatility, if data is available
for crypto_name, df in all_crypto_data.items():
    if 'Close' in df.columns: # Ensure 'Close' column exists before calculation
        df['Volatility'] = calculate_volatility(df)

app = dash.Dash(__name__)

# Check if all_crypto_data is empty and set default value
crypto_names = list(all_crypto_data.keys())
default_crypto = crypto_names[0] if crypto_names else None # Fallback if no data

# Function to create line chart for selected cryptocurrency's price data
def create_crypto_line_chart(df, crypto_name):
    if df.empty:
        return go.Figure() # Return an empty figure if there's no data
    return px.line(df, x=df.index, y='Close', title=f'{crypto_name.capitalize()} Price')

# Function to create bar chart for buy/sell counts
def create_buy_sell_chart(df, crypto_name):
    if df.empty:
        return go.Figure() # Return an empty figure if there's no data
    buy_count = (df['Signal'] == 'BUY').sum()
    sell_count = (df['Signal'] == 'SELL').sum()
    fig = go.Figure(data=[
        go.Bar(name='BUY', x=[crypto_name], y=[buy_count], marker_color='green'),
        go.Bar(name='SELL', x=[crypto_name], y=[sell_count], marker_color='red')
    ])
    fig.update_layout(barmode='group', title=f'{crypto_name.capitalize()} Buy/Sell')
    return fig

# Dashboard Layout
app.layout = html.Div([
    html.H1("Cryptocurrency Volatility Dashboard"),

    # Dropdown for selecting cryptocurrency
    dcc.Dropdown(id='crypto-selector',
                 options=[{'label': name, 'value': name} for name in crypto_names],
                 value=default_crypto, # Set default to first symbol or None
                 style={'width': '50%'}),

    # Show a message if no cryptocurrencies are available
    html.Div(id='no-data-message', style={'color': 'red', 'display': 'none'}),

    # Dropdown for selecting volatility level
    dcc.Dropdown(id='volatility-selector',
                 options=[{'label': 'All', 'value': 'All'},
                          {'label': 'High Volatility', 'value': 'High'},
                          {'label': 'Low Volatility', 'value': 'Low'}],
                 value='All', # Default value
                 style={'width': '50%', 'margin-top': '10px'}),

    # Graph to display selected cryptocurrency's price data
    dcc.Graph(id='price-chart'),

    # Graph to display Buy/Sell signal count
    dcc.Graph(id='buy-sell-chart')
])

```

```

])

# Callback to update charts based on selected cryptocurrency and volatility Level
@app.callback(
    [dash.dependencies.Output('price-chart', 'figure'),
     dash.dependencies.Output('buy-sell-chart', 'figure'),
     dash.dependencies.Output('no-data-message', 'style')],
    [dash.dependencies.Input('crypto-selector', 'value'),
     dash.dependencies.Input('volatility-selector', 'value')])
)
def update_charts(crypto_name, volatility_level):
    if crypto_name is None:
        return go.Figure(), go.Figure(), {'display': 'block'} # Show no data message

    # Filter the DataFrame for the selected cryptocurrency
    df = all_crypto_data.get(crypto_name, pd.DataFrame()).copy() # Ensure df is a DataFrame

    # Filter by volatility level if selected
    if volatility_level == 'High':
        df = df[df['Volatility'] > df['Volatility'].median()]
    elif volatility_level == 'Low':
        df = df[df['Volatility'] <= df['Volatility'].median()]

    # Generate charts
    price_chart = create_crypto_line_chart(df, crypto_name)
    buy_sell_chart = create_buy_sell_chart(df, crypto_name)

    return price_chart, buy_sell_chart, {'display': 'none'} # Hide no data message

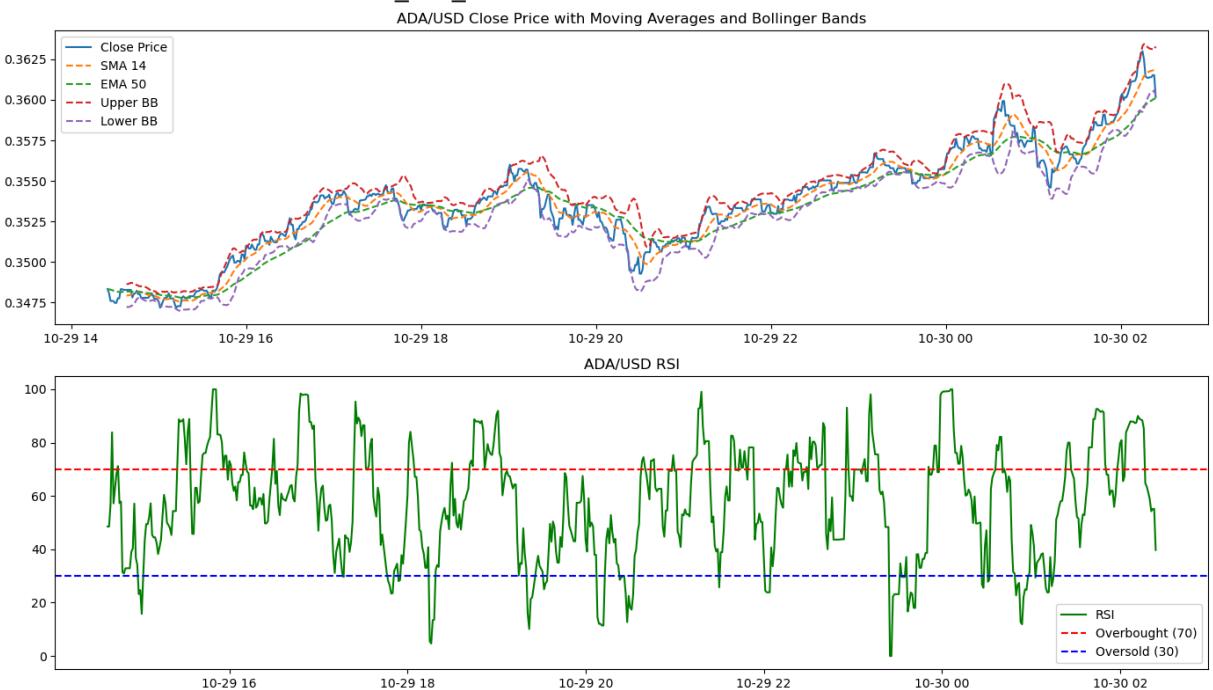
if __name__ == '__main__':
    app.run_server(debug=True, port=8054)

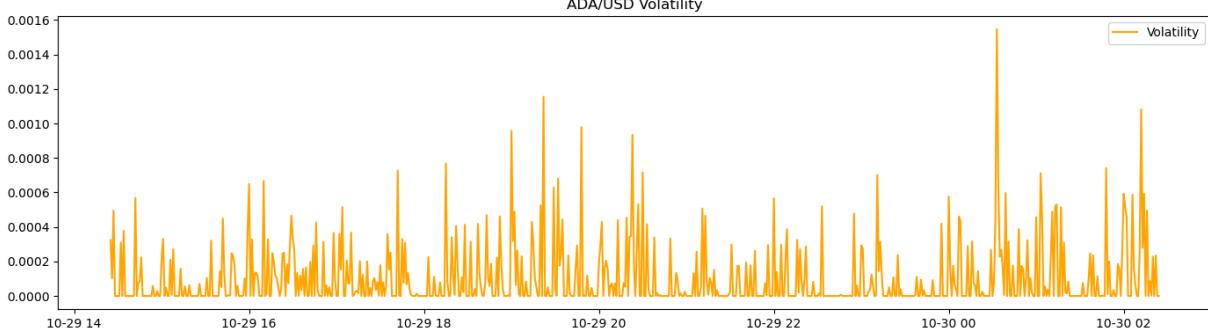
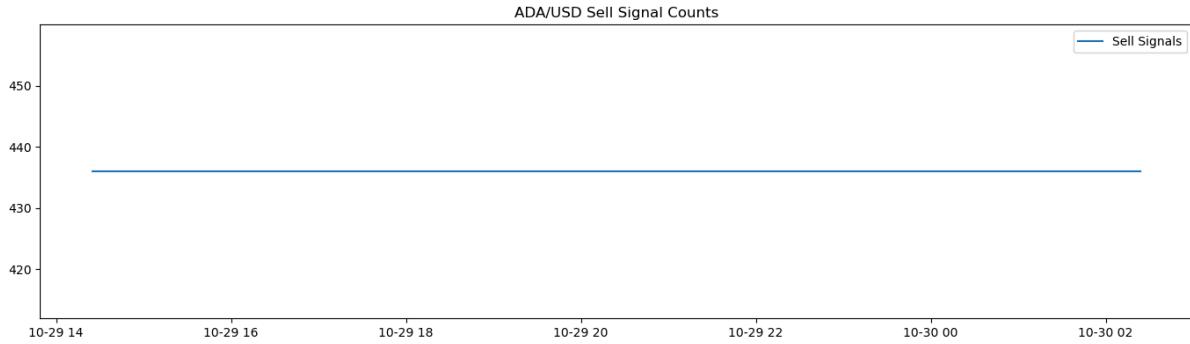
```

Data successfully saved to SQL table: ADA_USD_data

SQL connection closed.

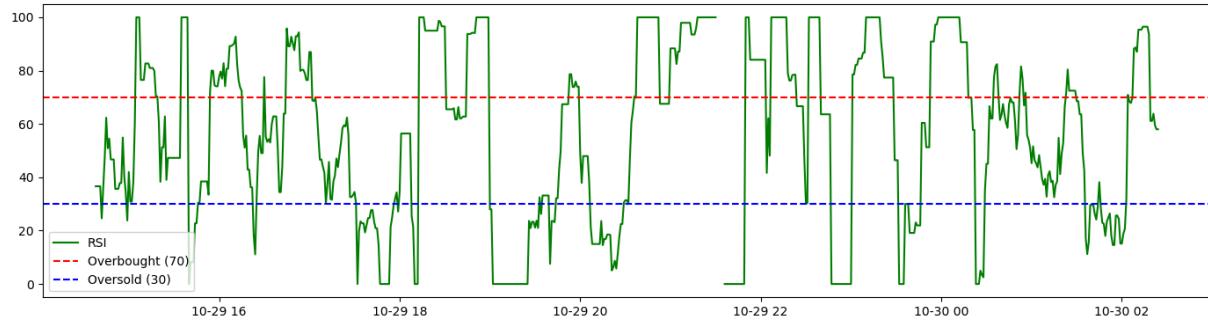
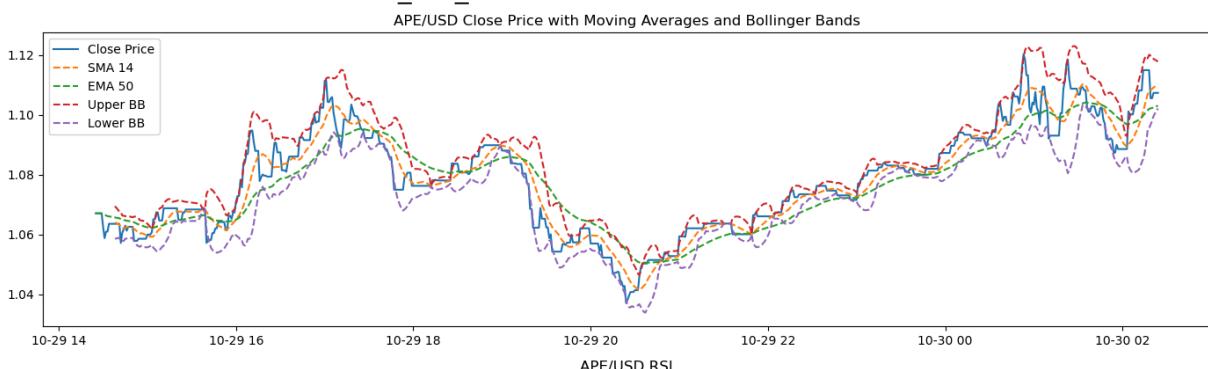
Data saved to CSV file: ADA_USD_data.csv

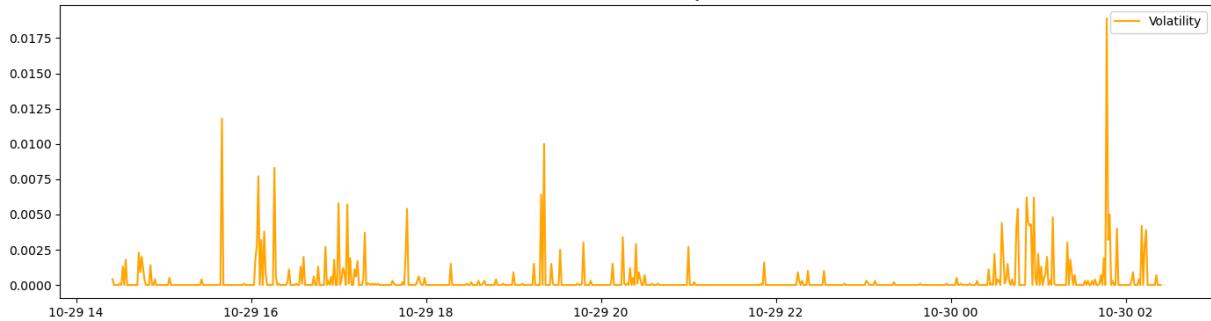
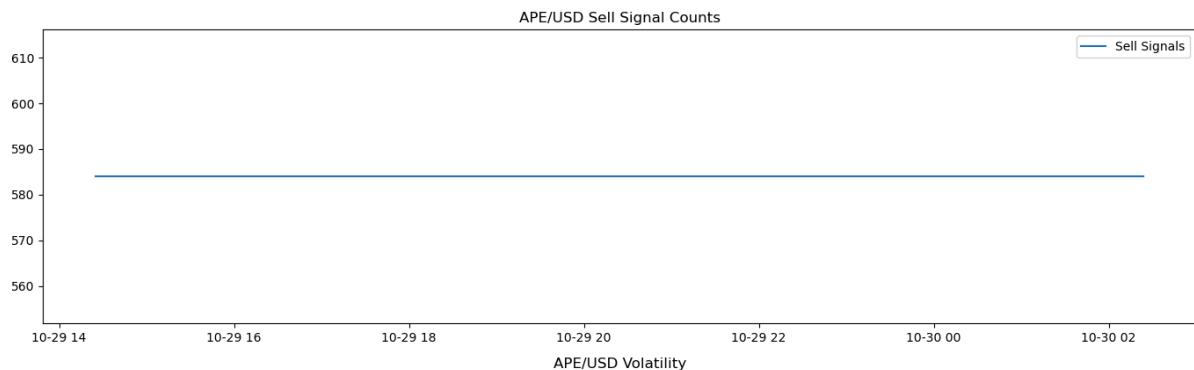




Data successfully saved to SQL table: APE_USD_data
SQL connection closed.

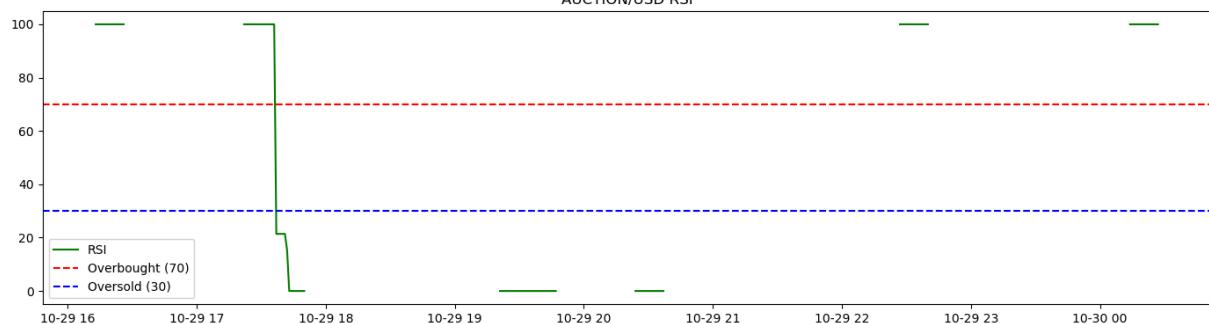
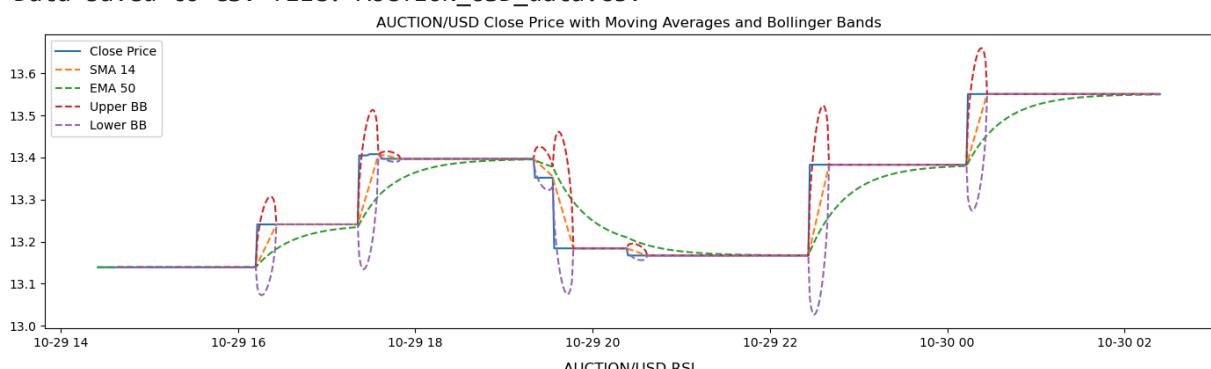
Data saved to CSV file: APE_USD_data.csv

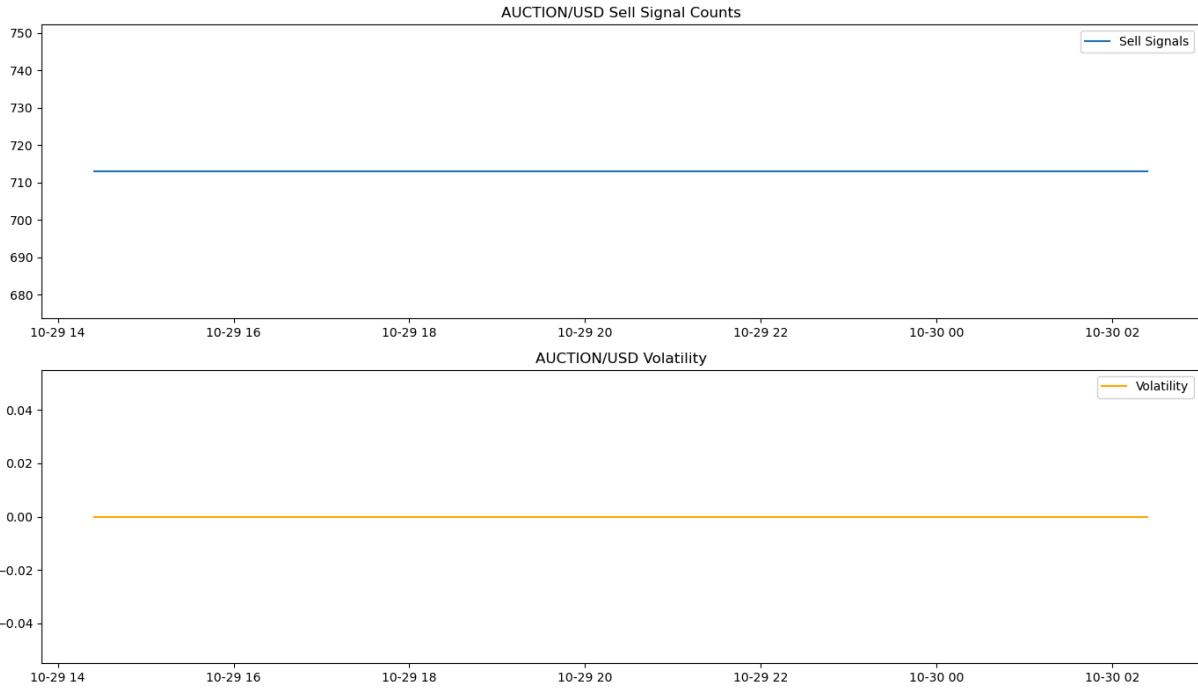




Data successfully saved to SQL table: AUCTION_USD_data
SQL connection closed.

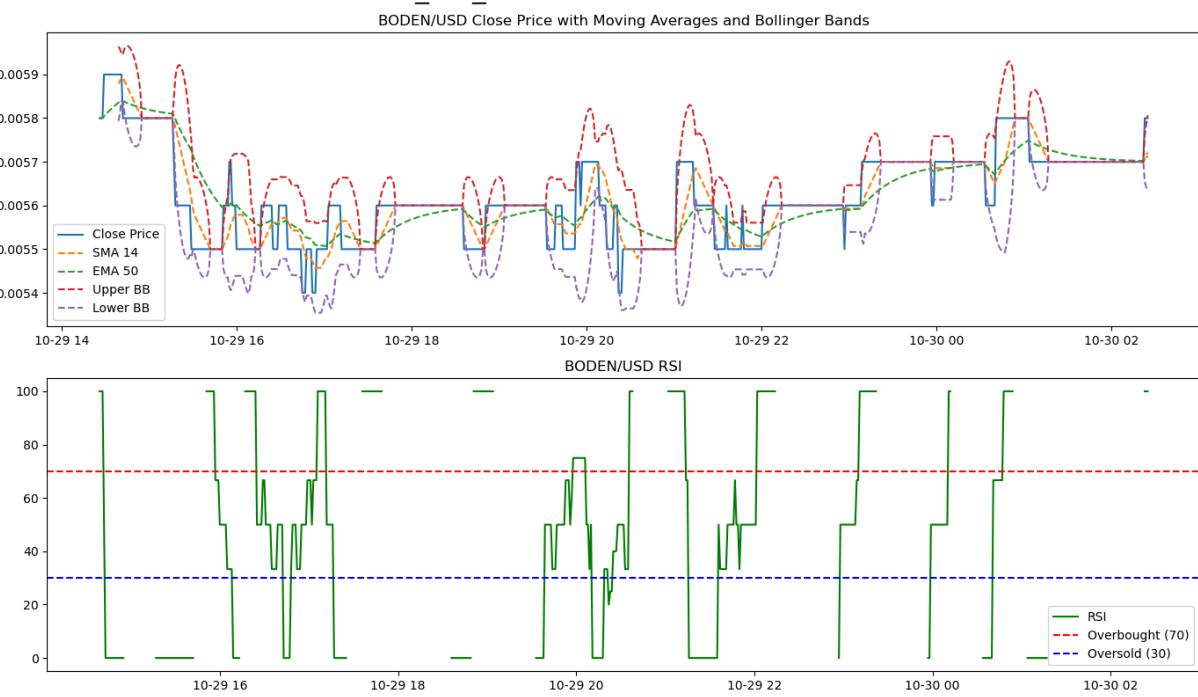
Data saved to CSV file: AUCTION_USD_data.csv

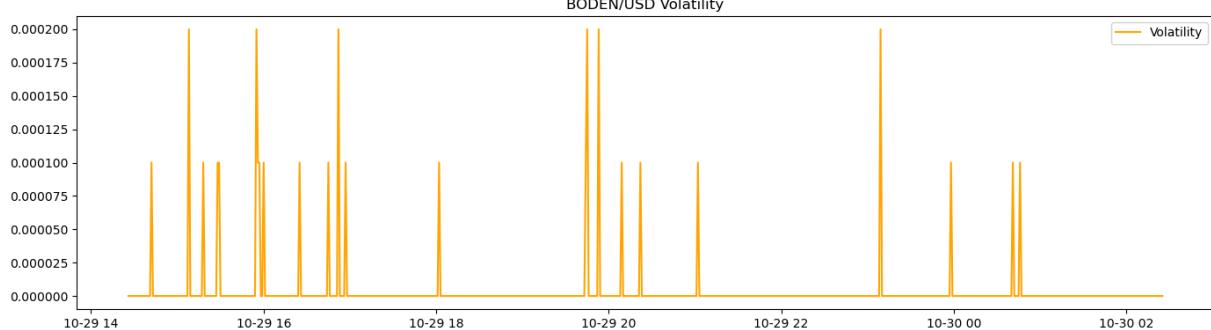
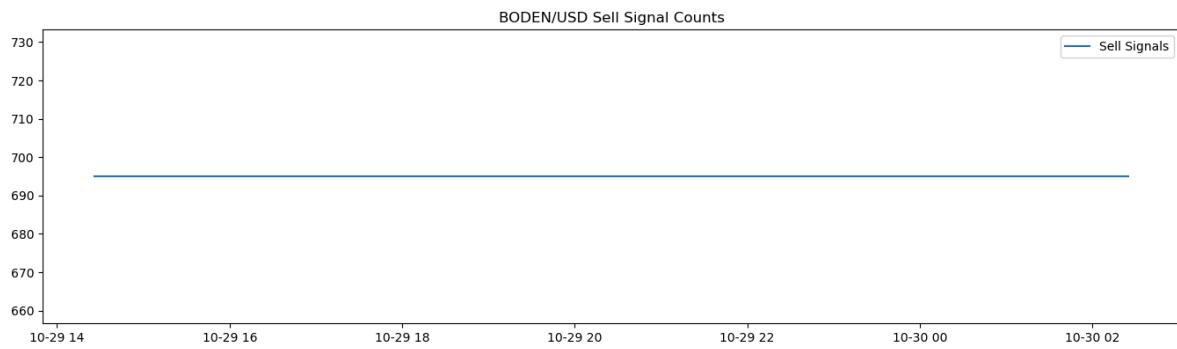




Data successfully saved to SQL table: BODEN_USD_data
SQL connection closed.

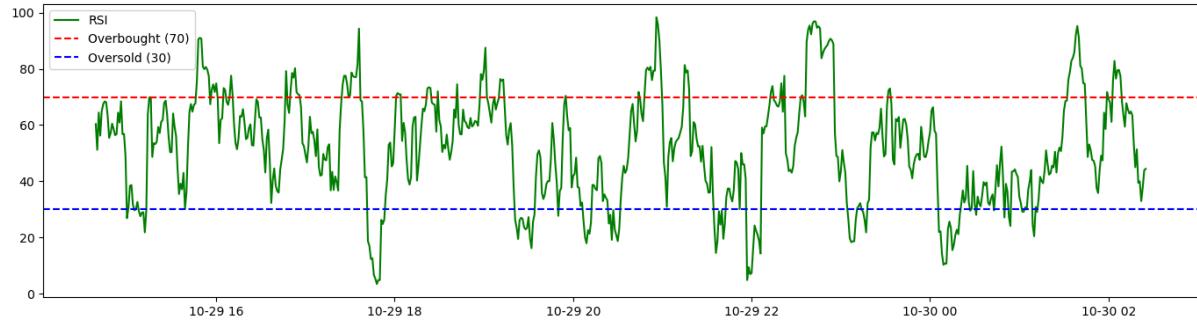
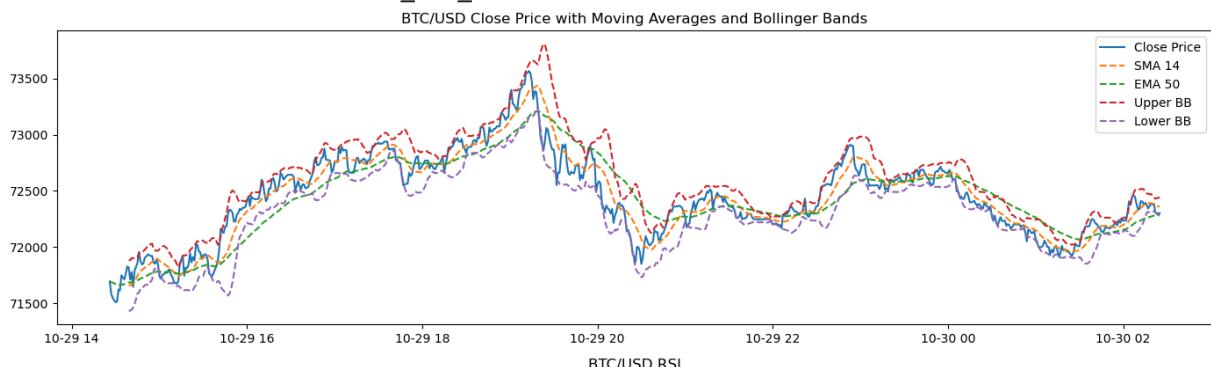
Data saved to CSV file: BODEN_USD_data.csv

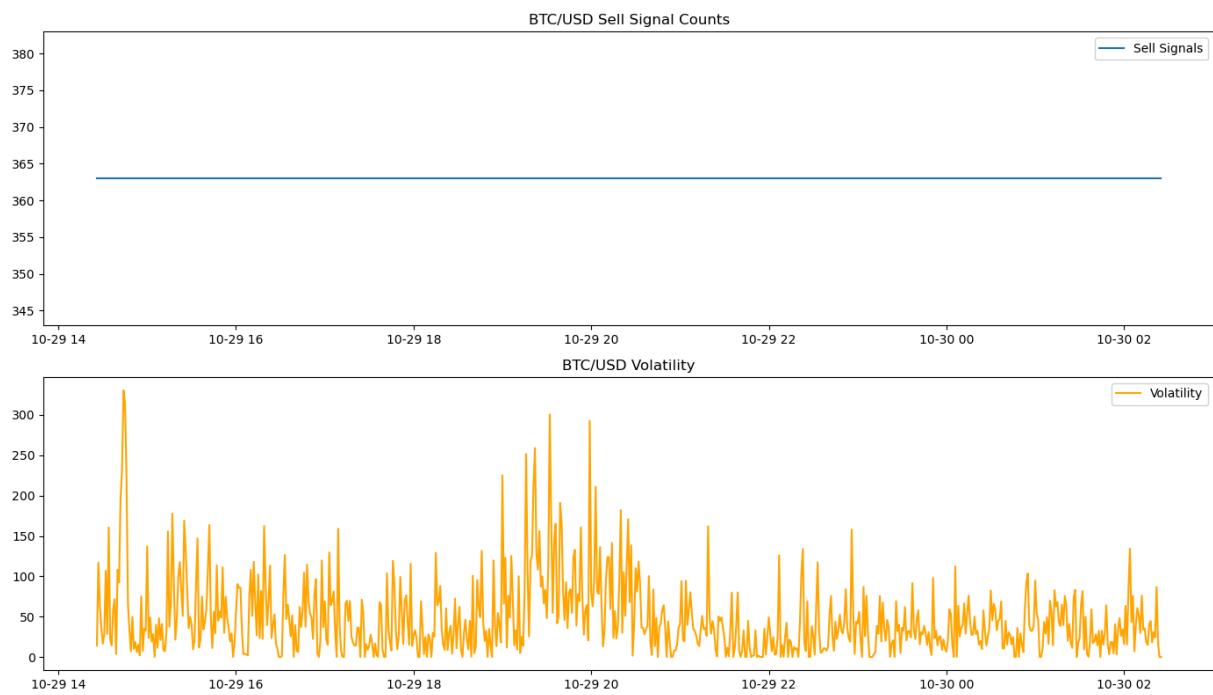




Data successfully saved to SQL table: BTC_USD_data
SQL connection closed.

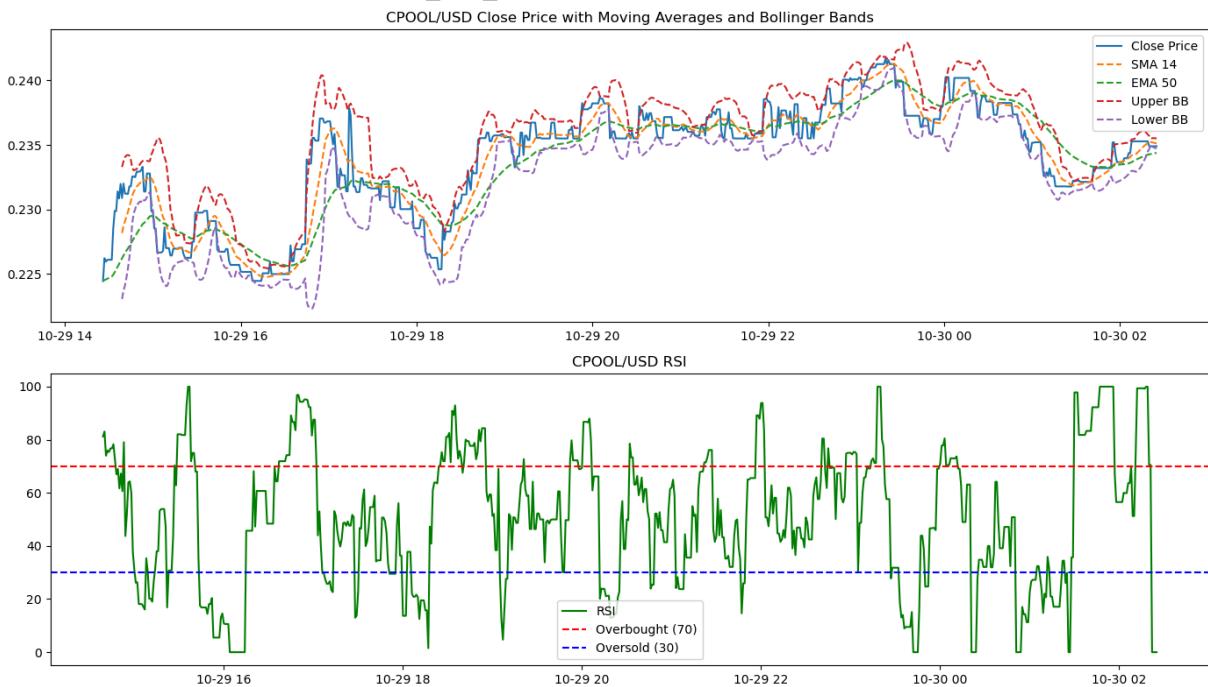
Data saved to CSV file: BTC_USD_data.csv

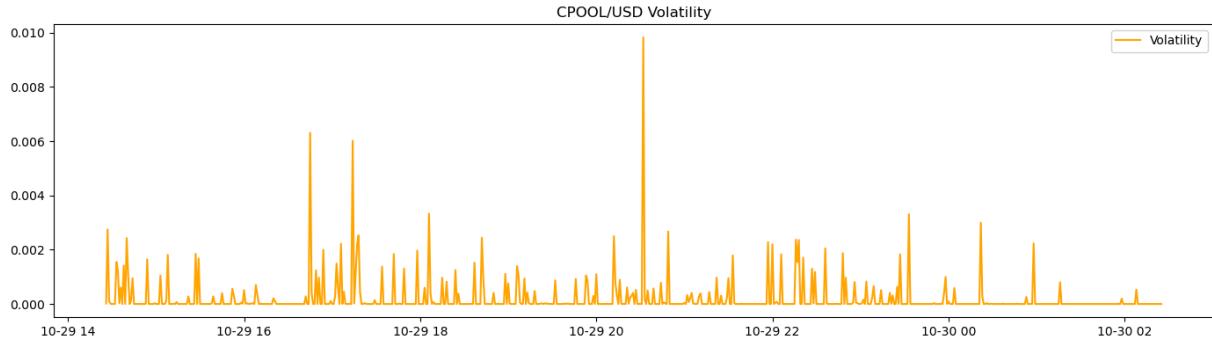
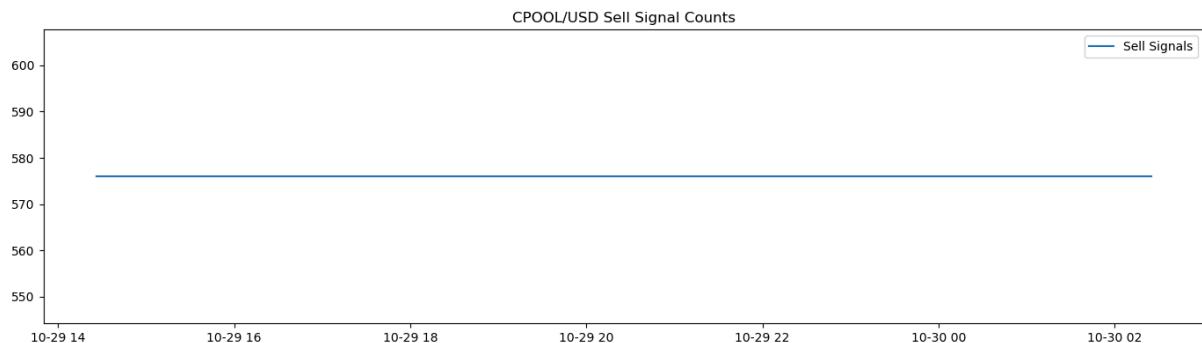




Data successfully saved to SQL table: CPOOL_USD_data
SQL connection closed.

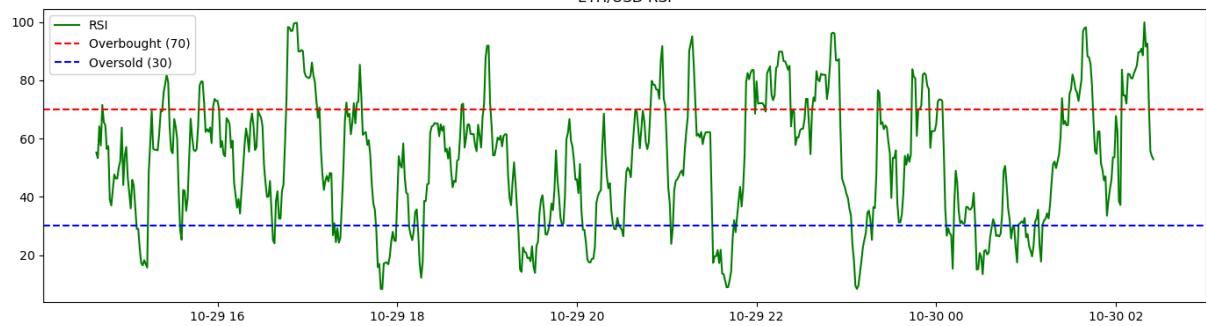
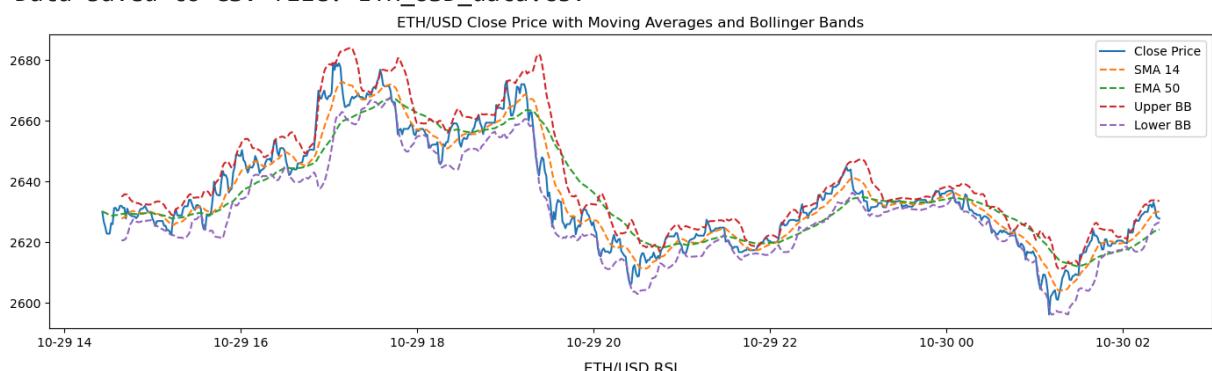
Data saved to CSV file: CPOOL_USD_data.csv

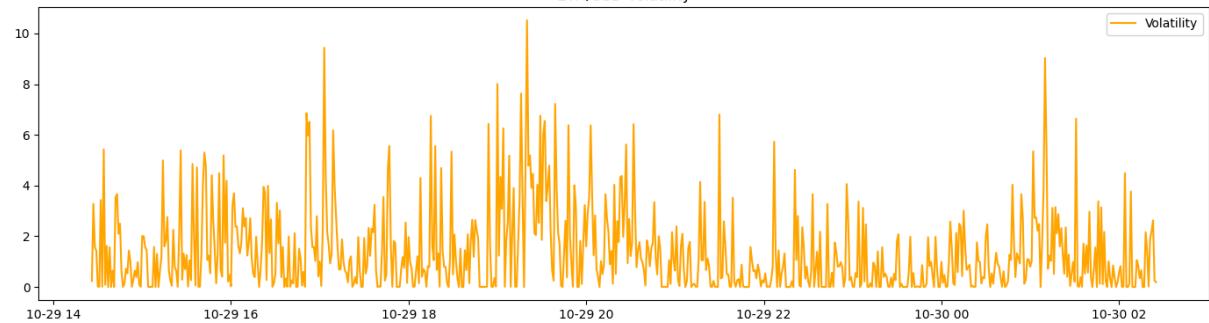
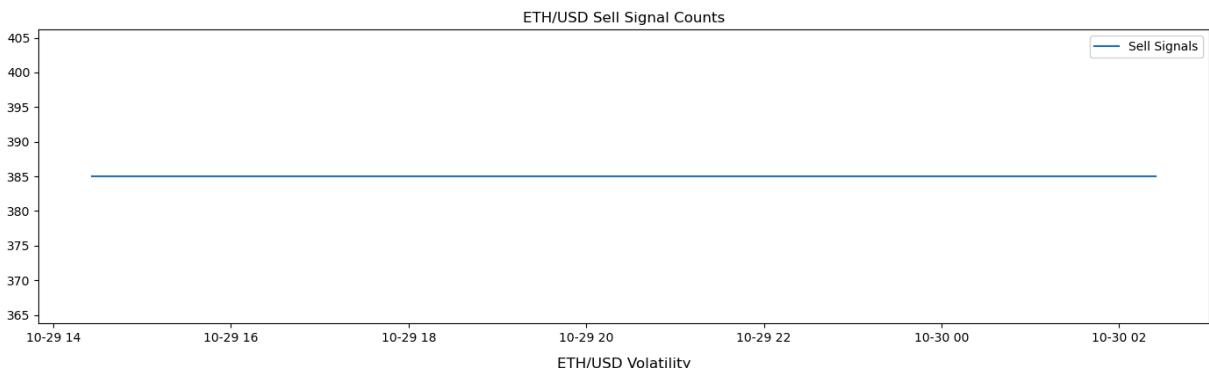




Data successfully saved to SQL table: ETH_USD_data
SQL connection closed.

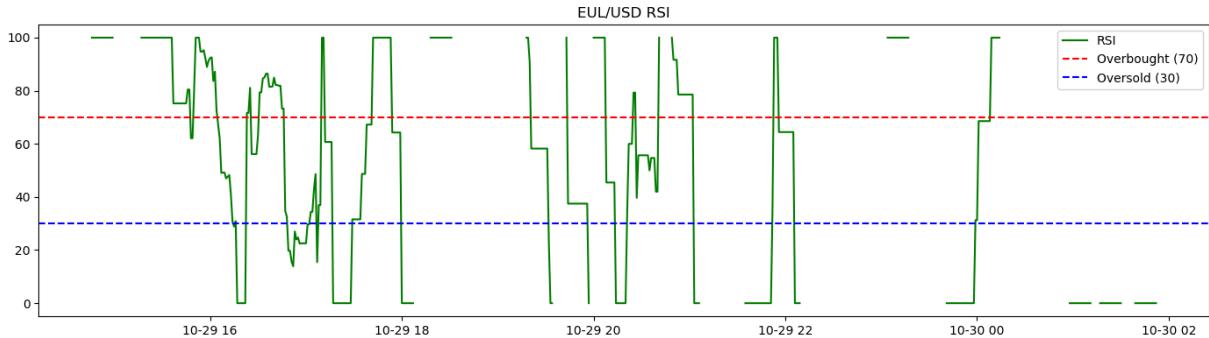
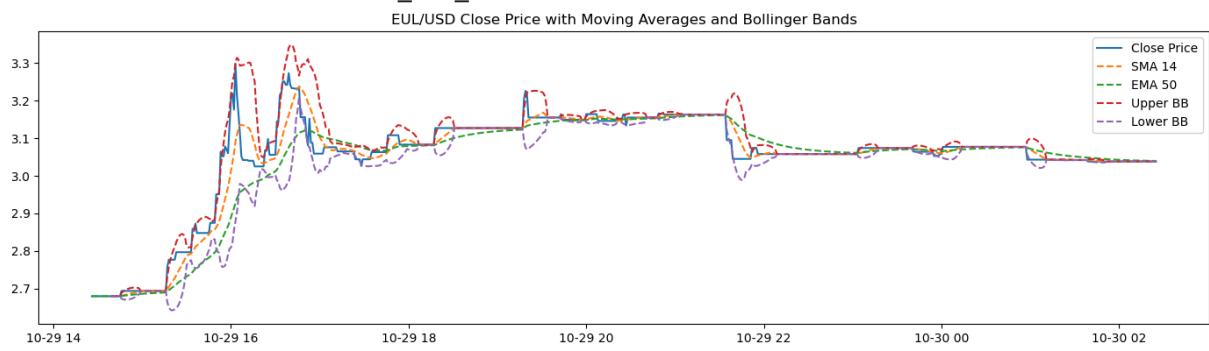
Data saved to CSV file: ETH_USD_data.csv

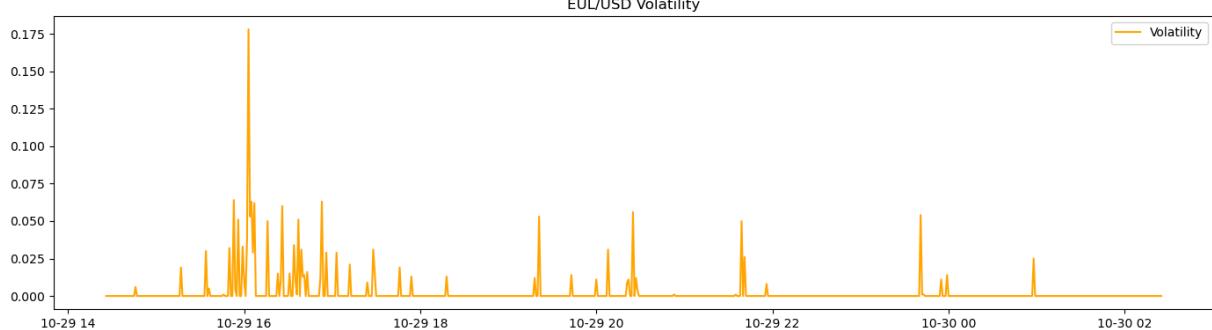
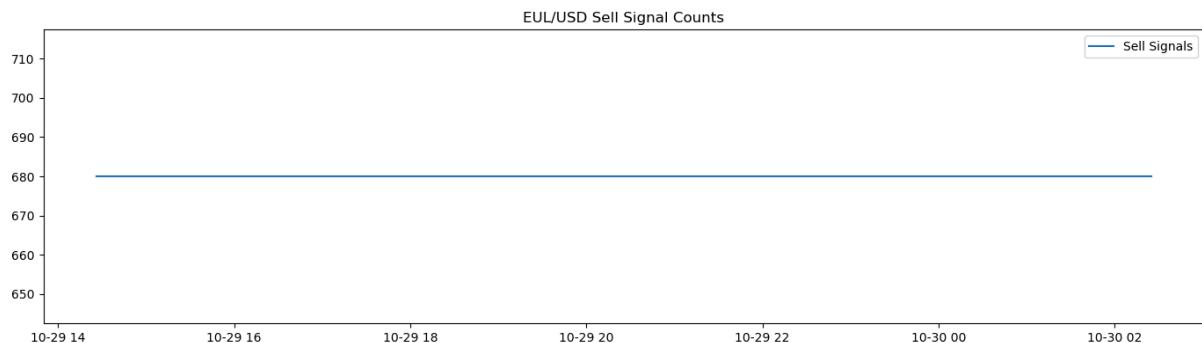




Data successfully saved to SQL table: EUL_USD_data
SQL connection closed.

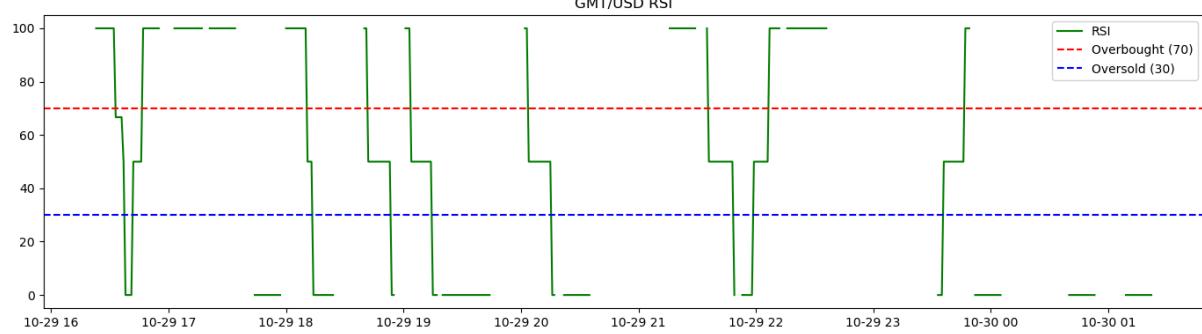
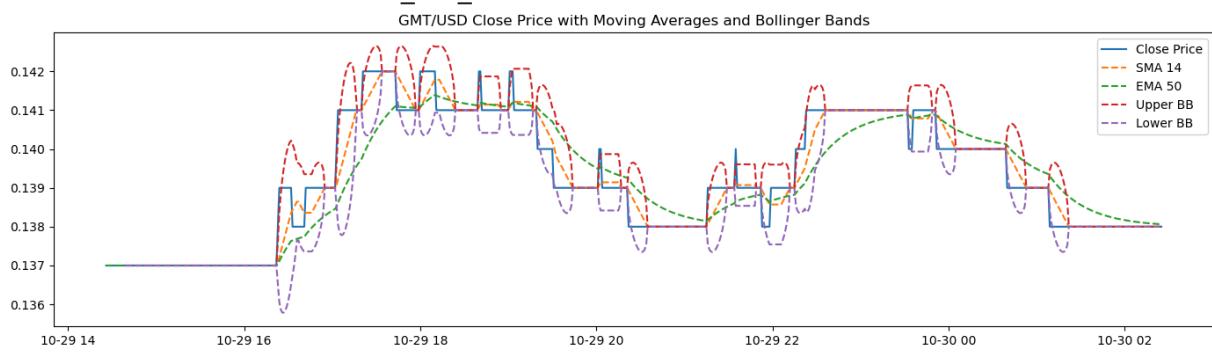
Data saved to CSV file: EUL_USD_data.csv

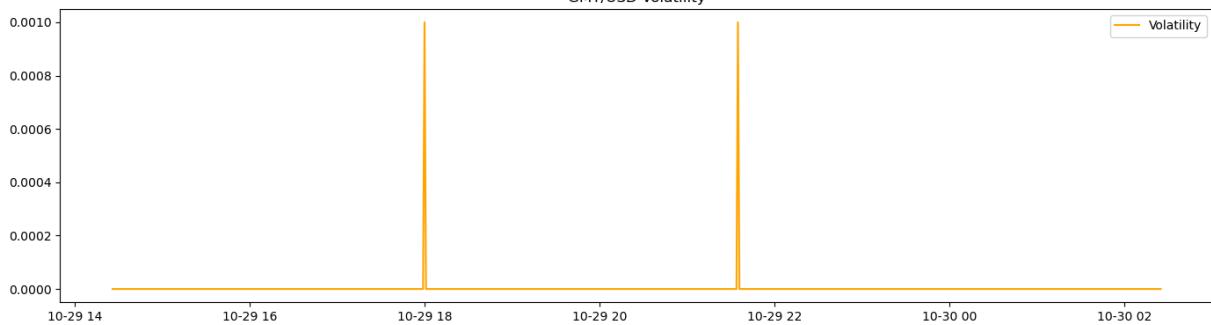
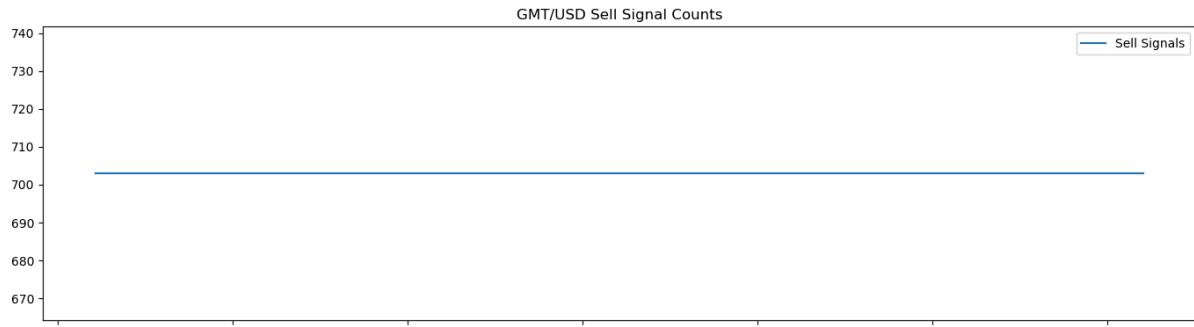




Data successfully saved to SQL table: GMT_USD_data
SQL connection closed.

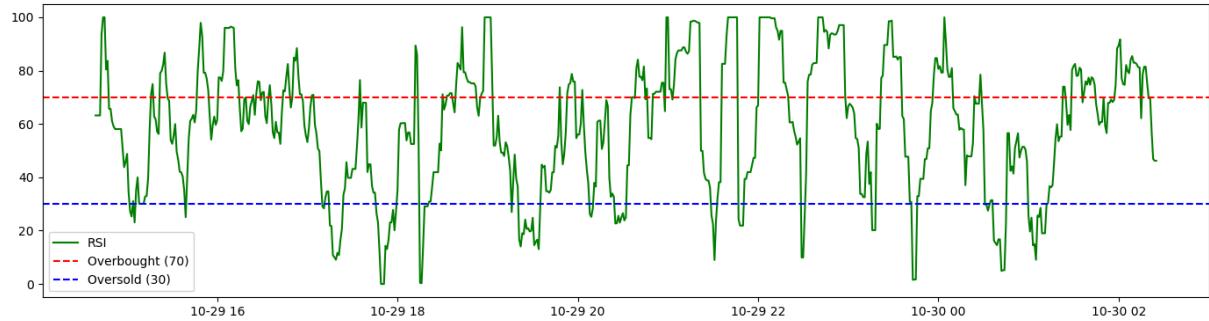
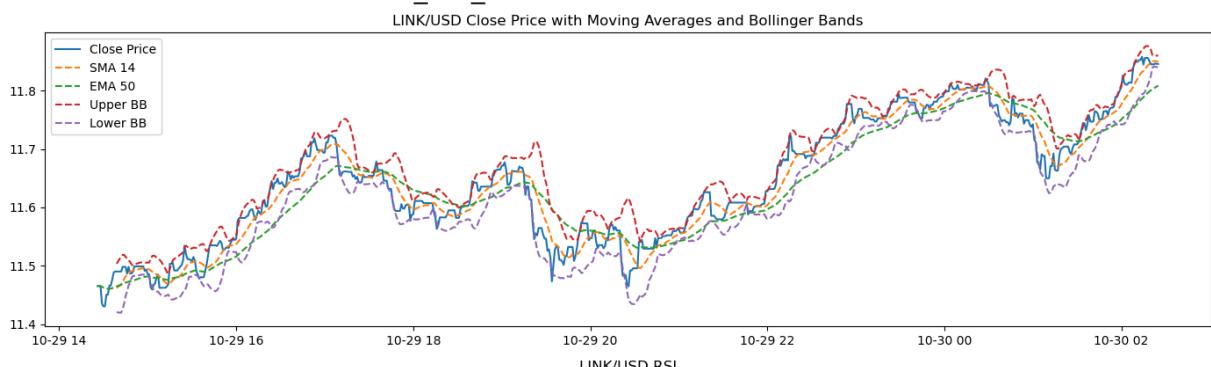
Data saved to CSV file: GMT_USD_data.csv

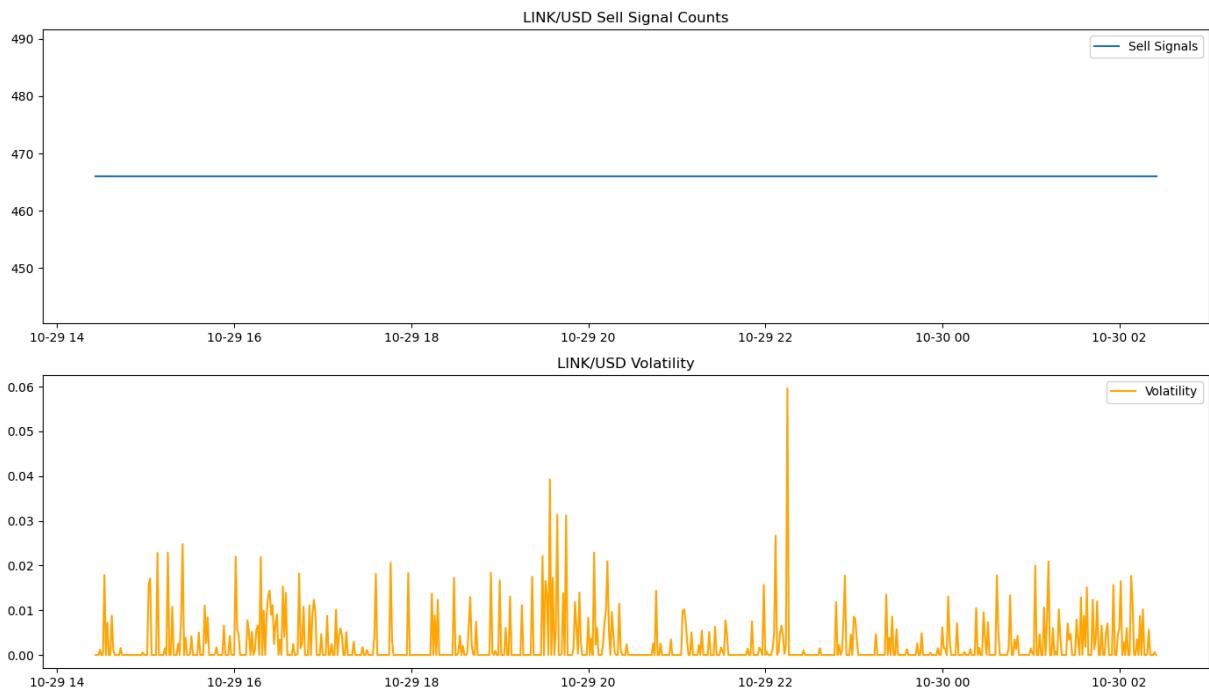




Data successfully saved to SQL table: LINK_USD_data
SQL connection closed.

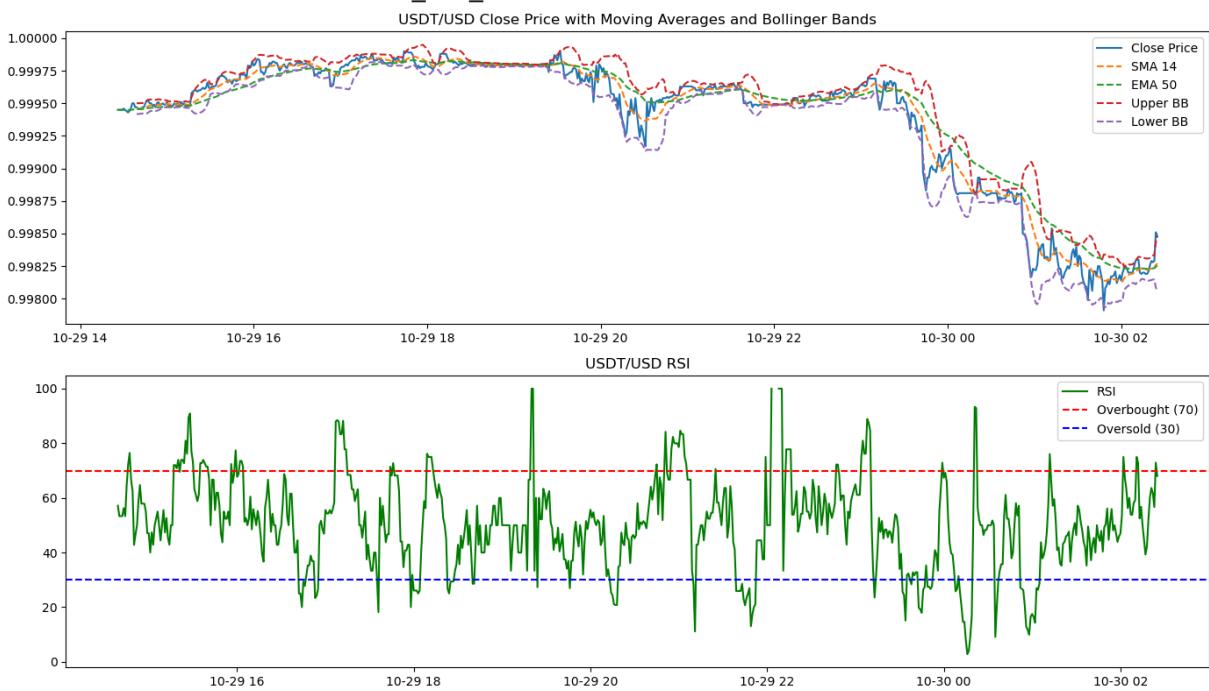
Data saved to CSV file: LINK_USD_data.csv

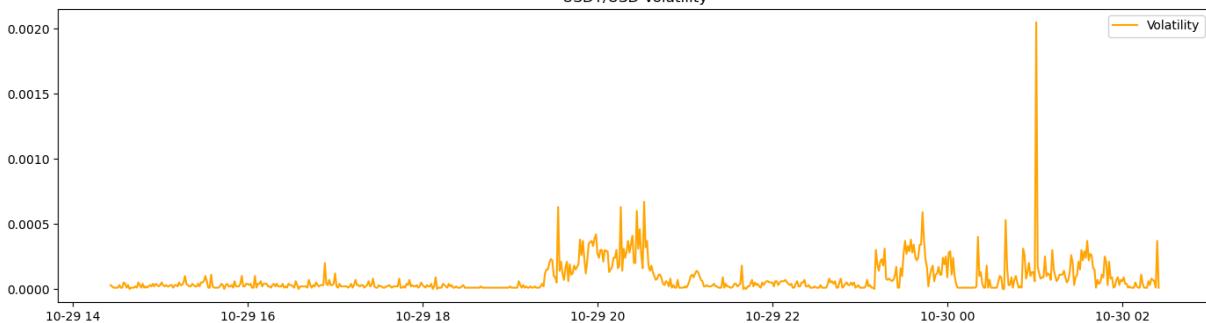
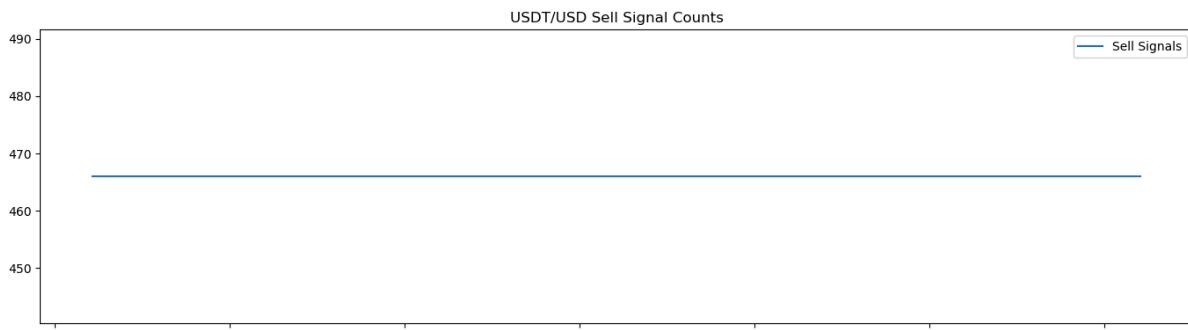




Data successfully saved to SQL table: USDT_USD_data
SQL connection closed.

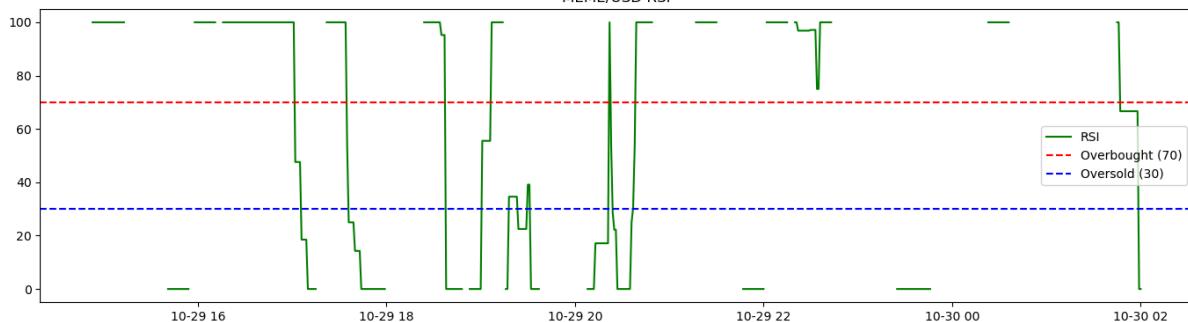
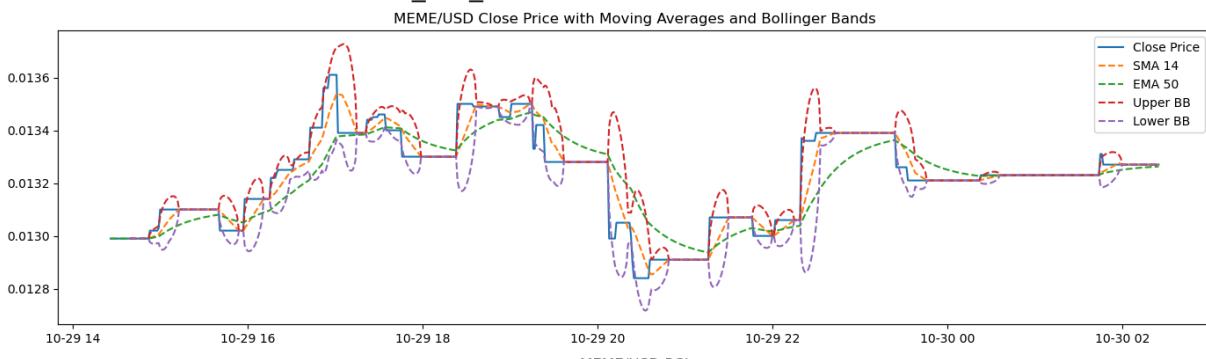
Data saved to CSV file: USDT_USD_data.csv

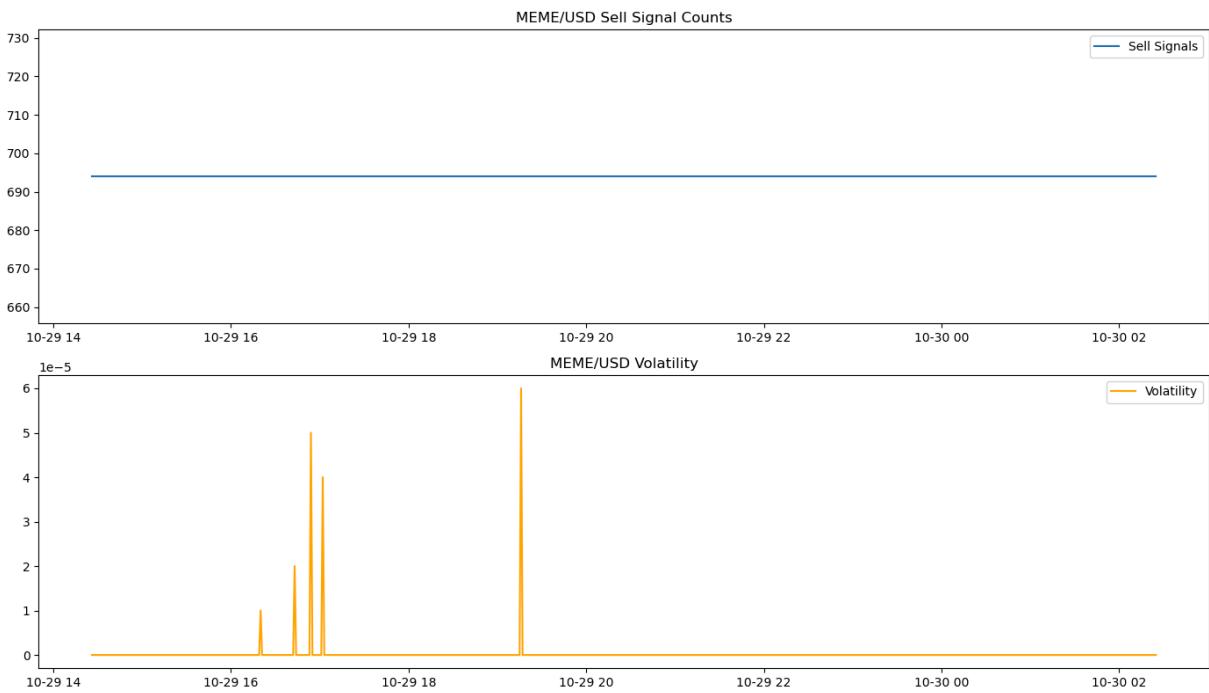




Data successfully saved to SQL table: MEME_USD_data
SQL connection closed.

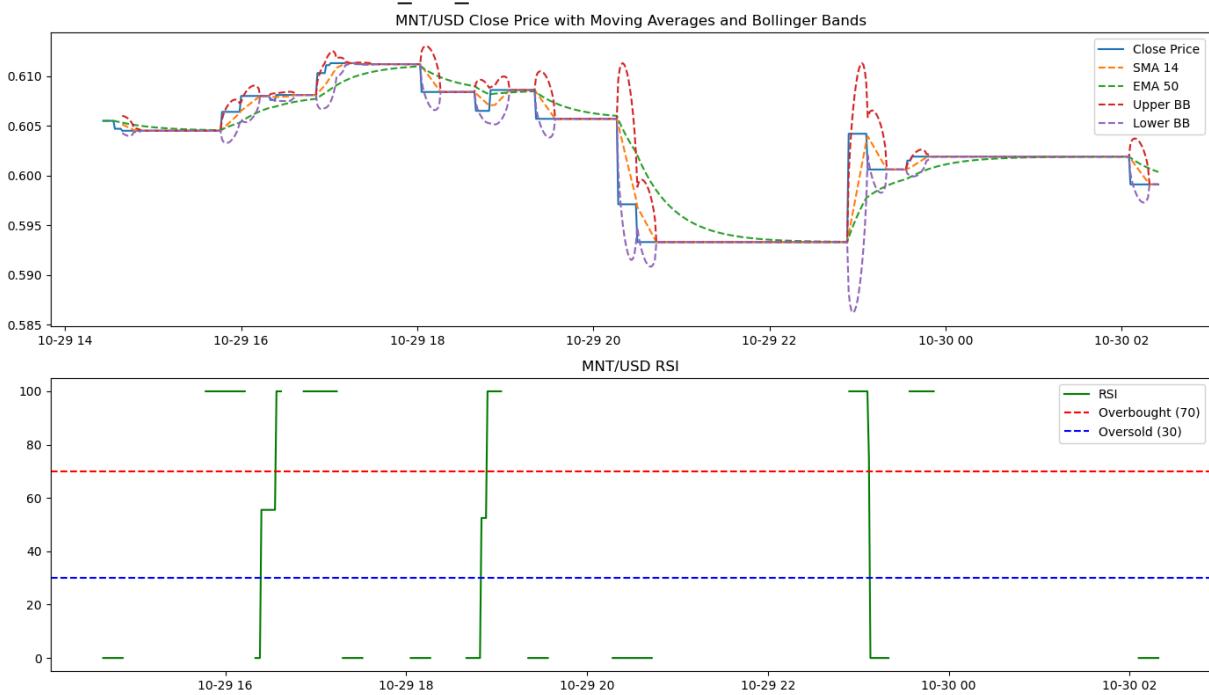
Data saved to CSV file: MEME_USD_data.csv

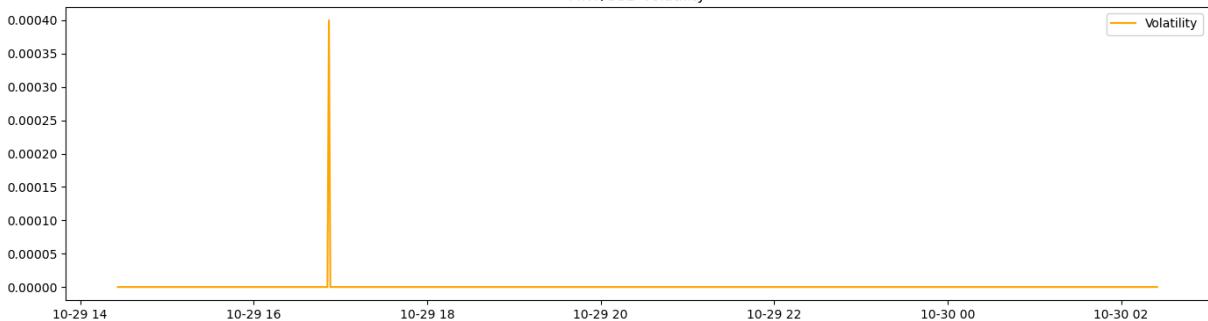
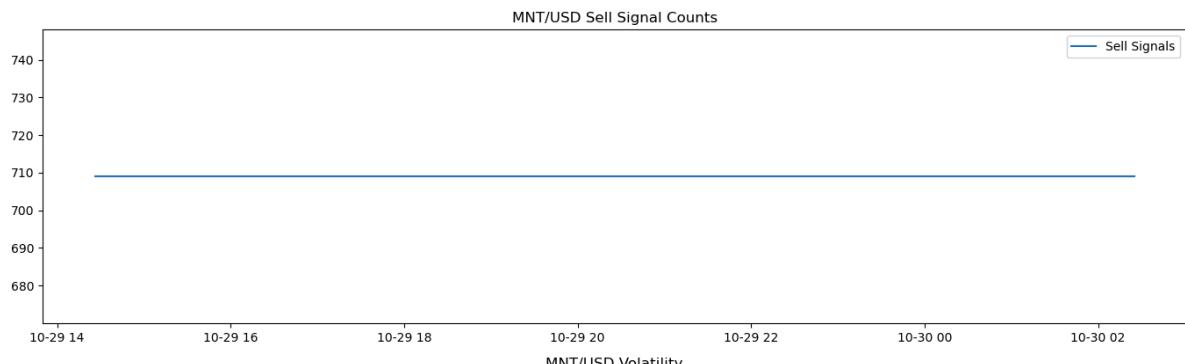




Data successfully saved to SQL table: MNT_USD_data
SQL connection closed.

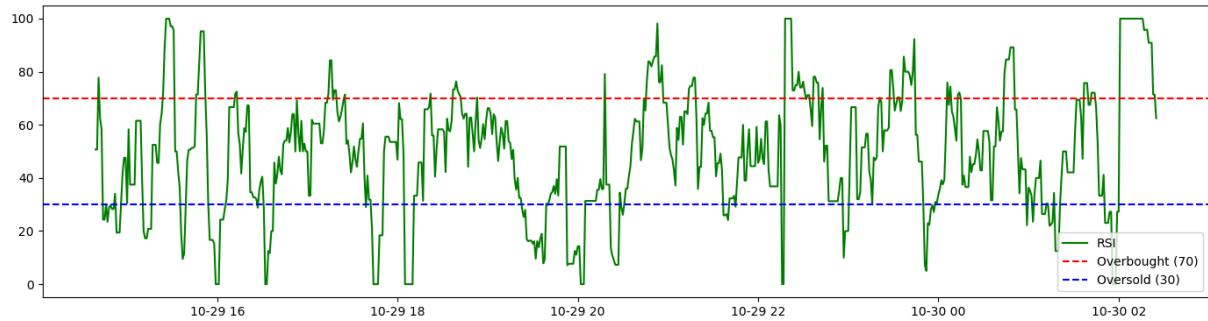
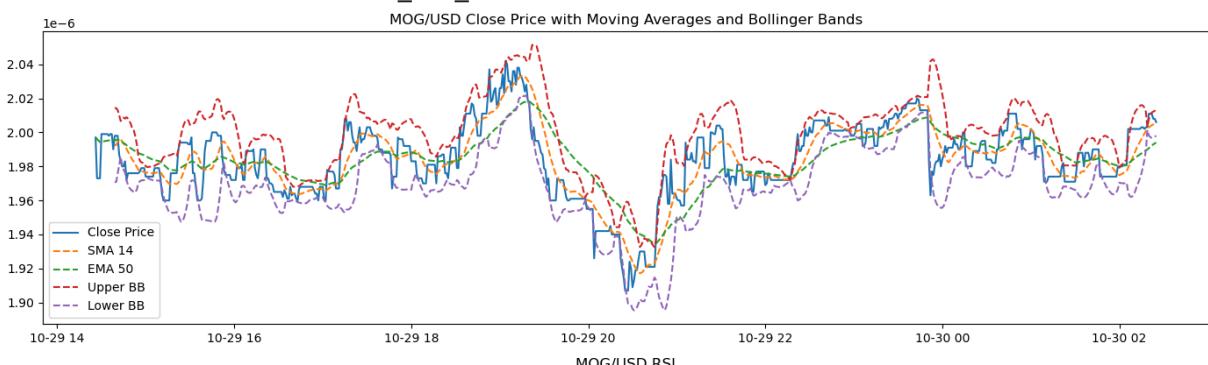
Data saved to CSV file: MNT_USD_data.csv

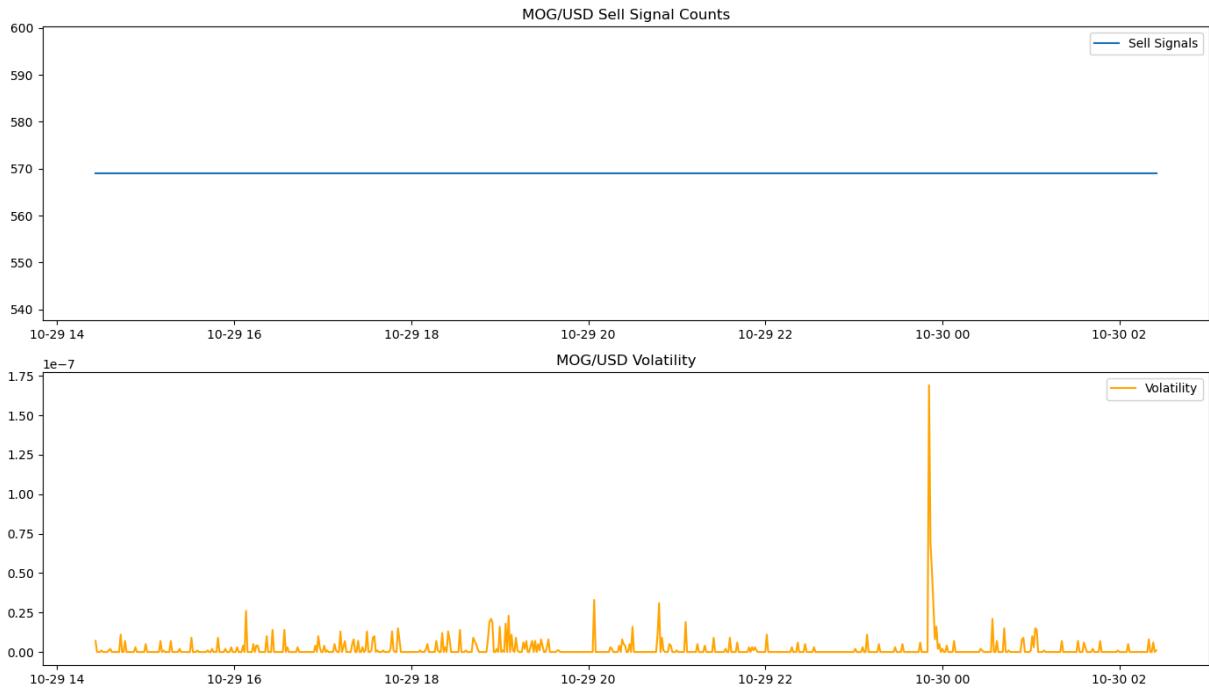




Data successfully saved to SQL table: MOG_USD_data
SQL connection closed.

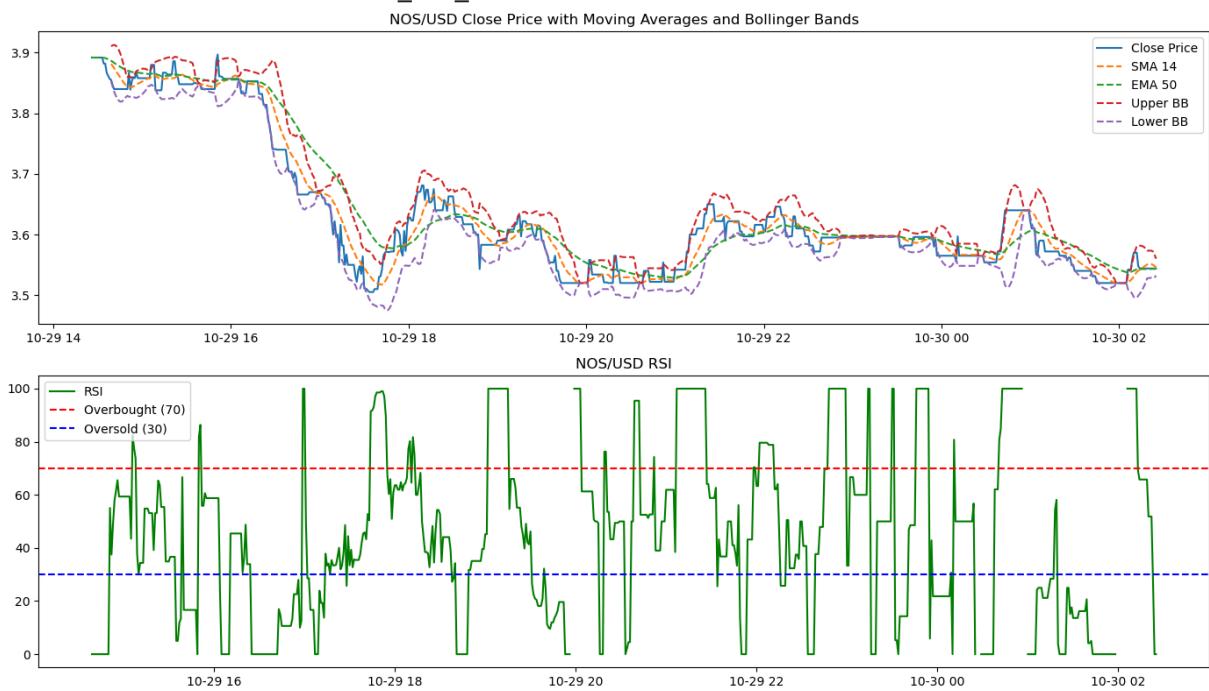
Data saved to CSV file: MOG_USD_data.csv

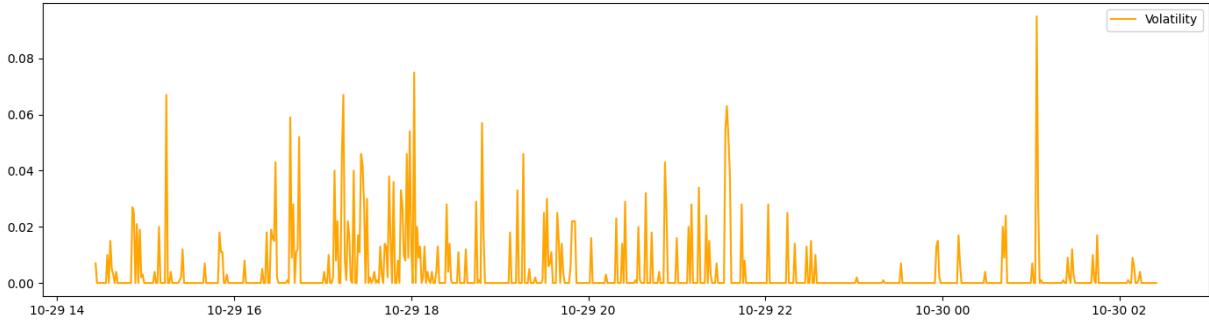
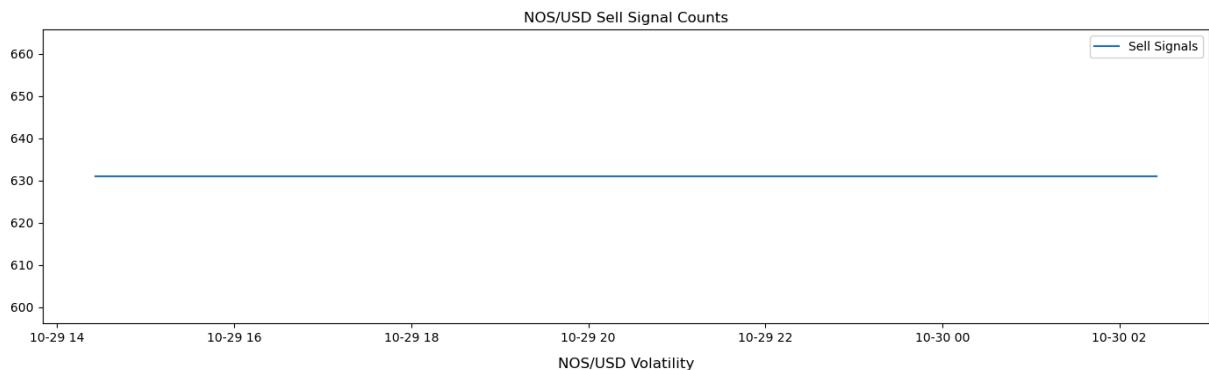




Data successfully saved to SQL table: NOS_USD_data
SQL connection closed.

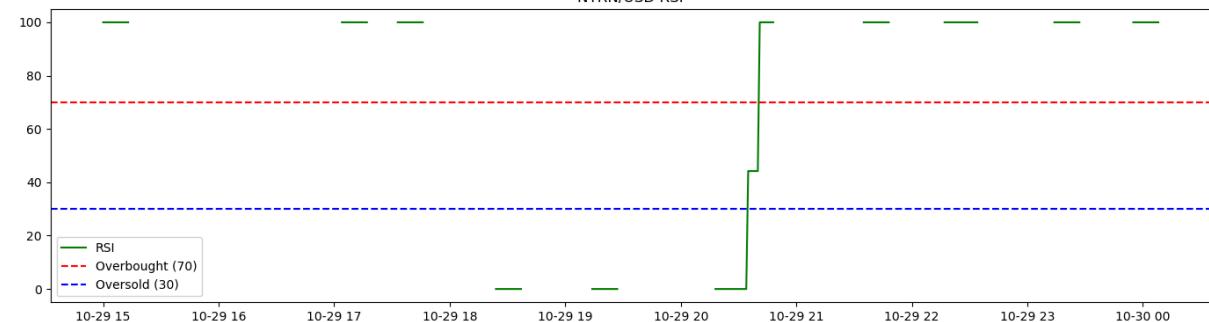
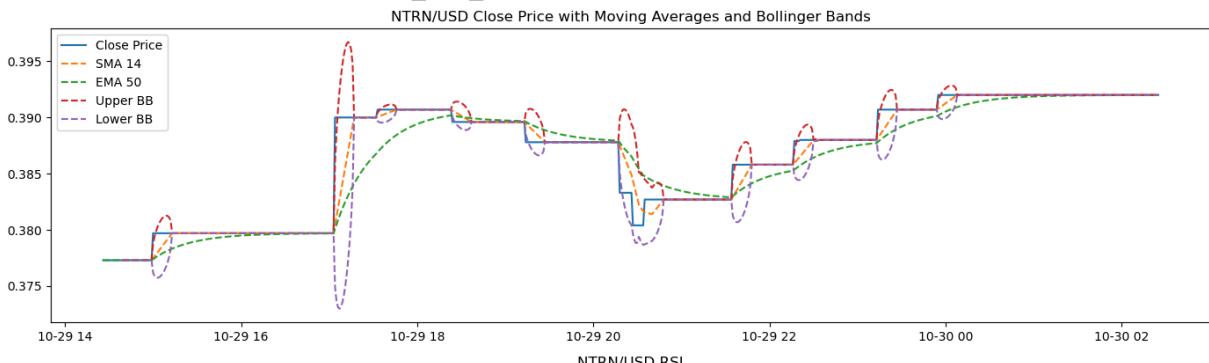
Data saved to CSV file: NOS_USD_data.csv

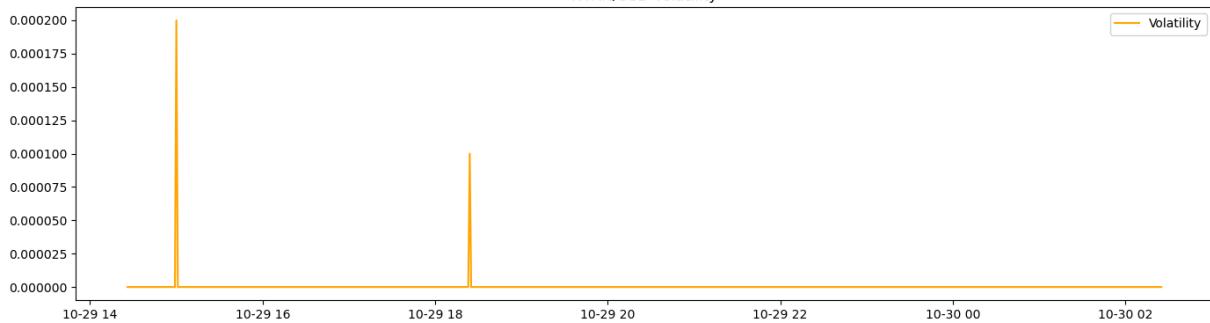
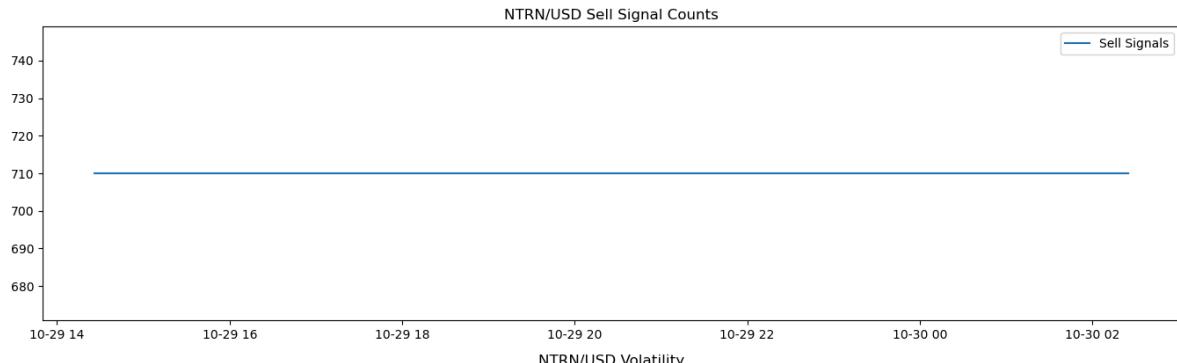




Data successfully saved to SQL table: NTRN_USD_data
SQL connection closed.

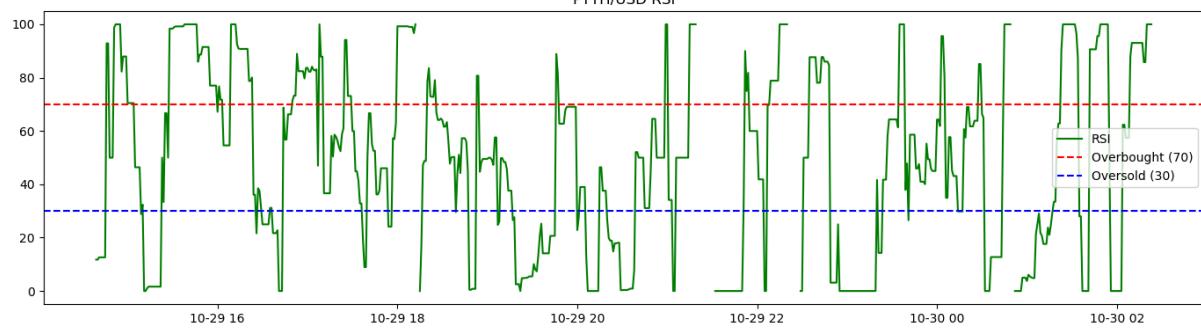
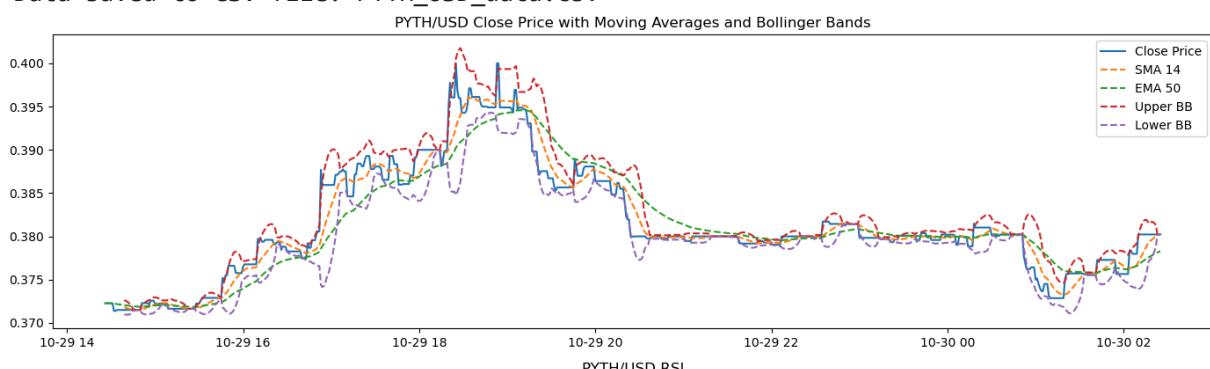
Data saved to CSV file: NTRN_USD_data.csv

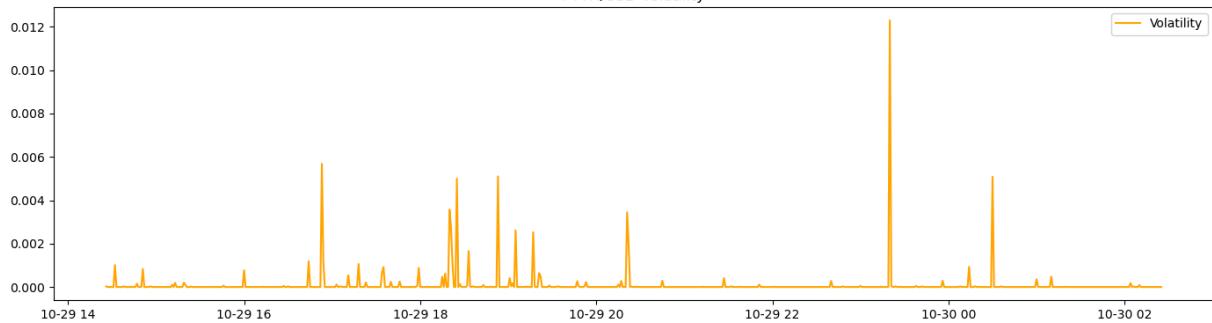
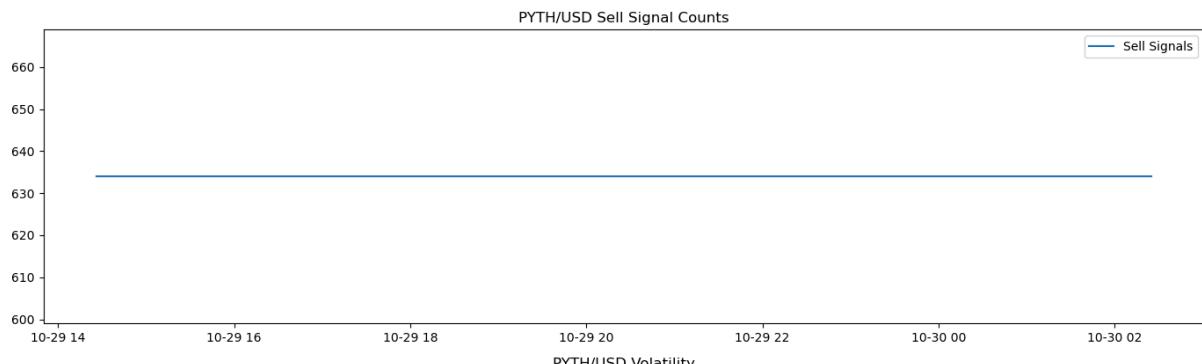




Data successfully saved to SQL table: PYTH_USD_data
SQL connection closed.

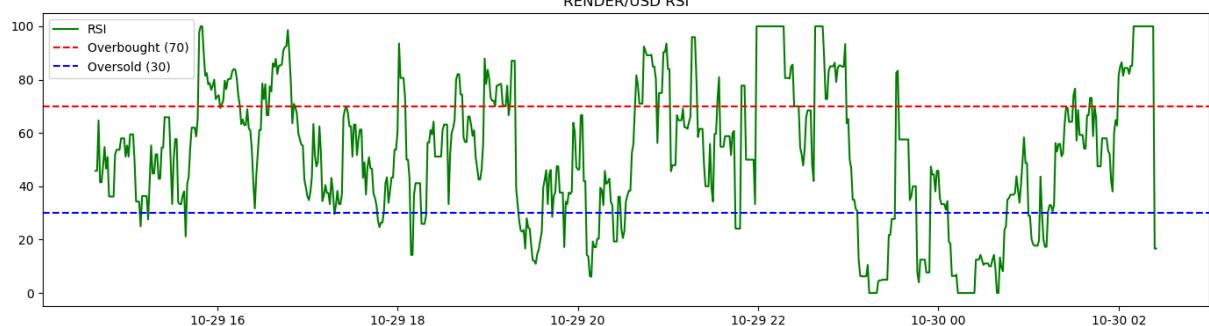
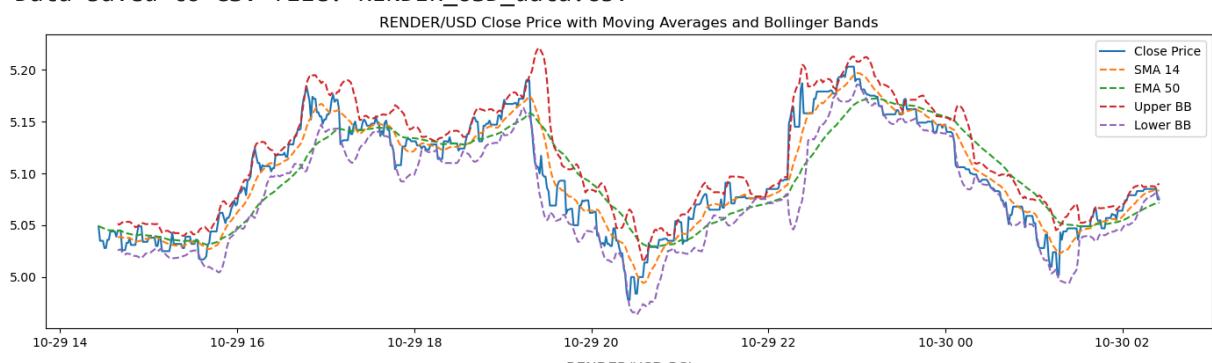
Data saved to CSV file: PYTH_USD_data.csv

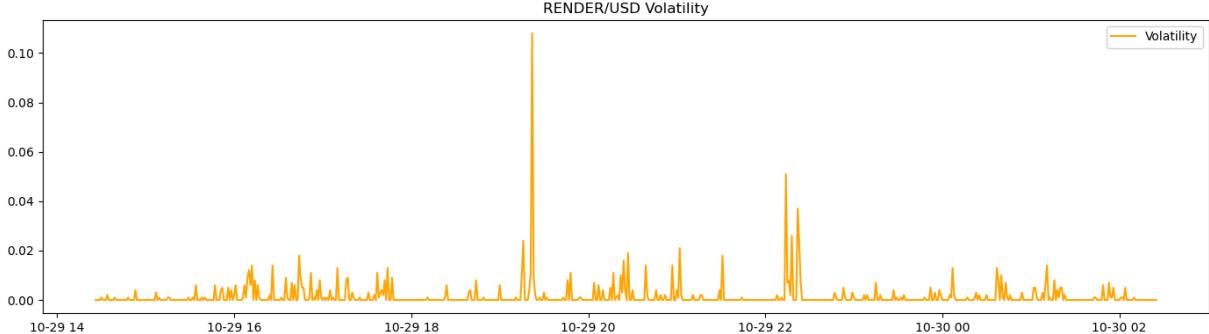
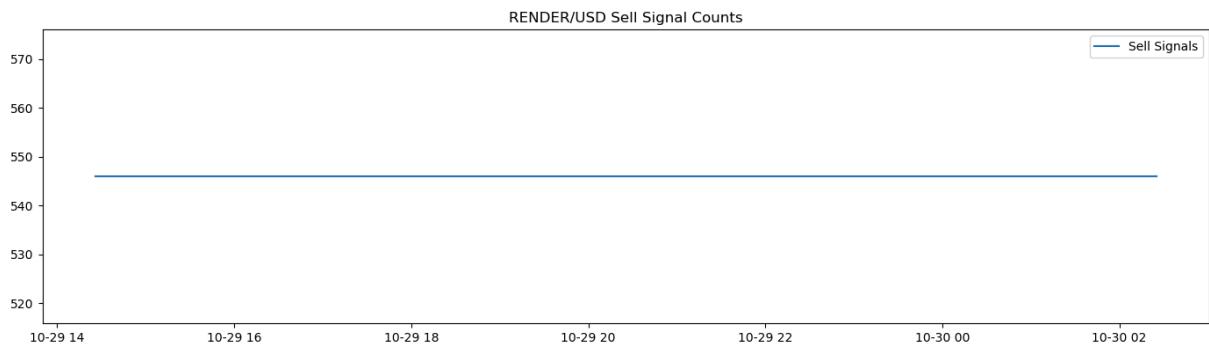




Data successfully saved to SQL table: RENDER_USD_data
SQL connection closed.

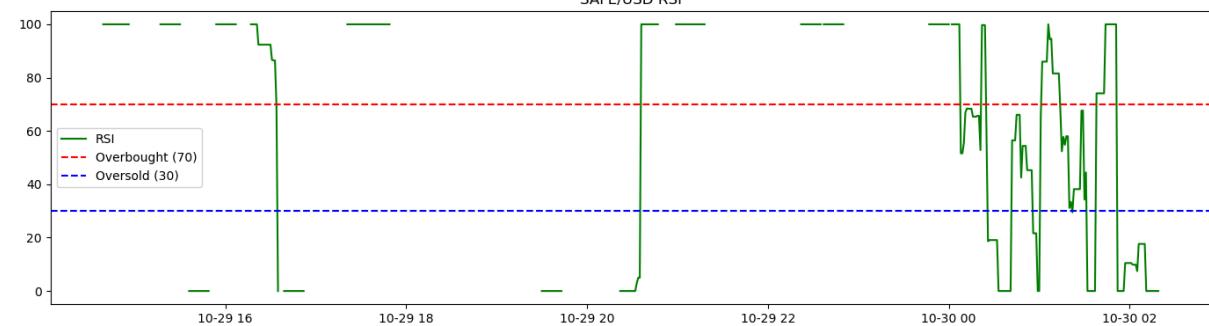
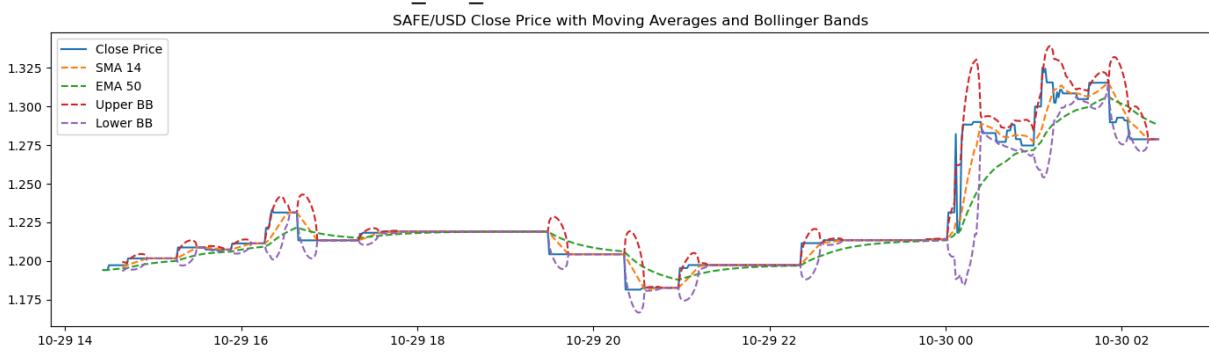
Data saved to CSV file: RENDER_USD_data.csv

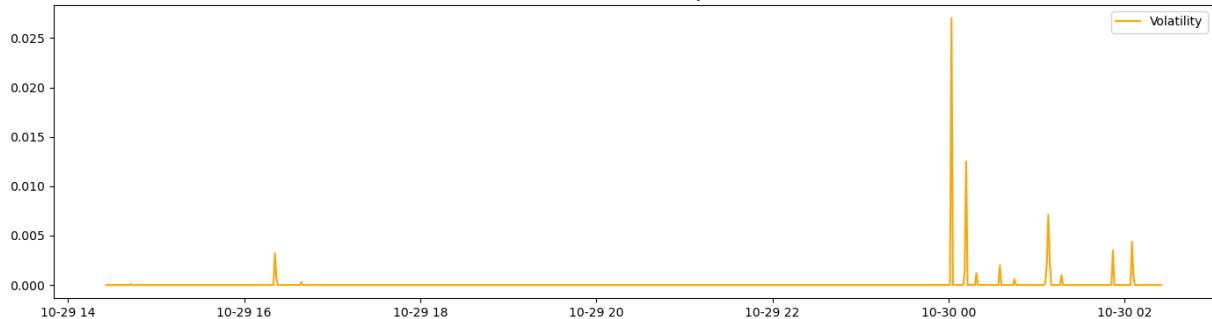
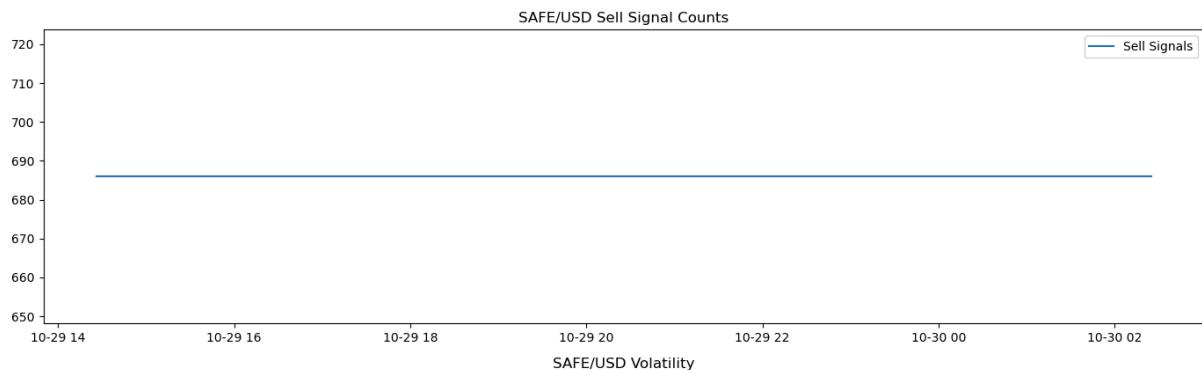




Data successfully saved to SQL table: SAFE_USD_data
SQL connection closed.

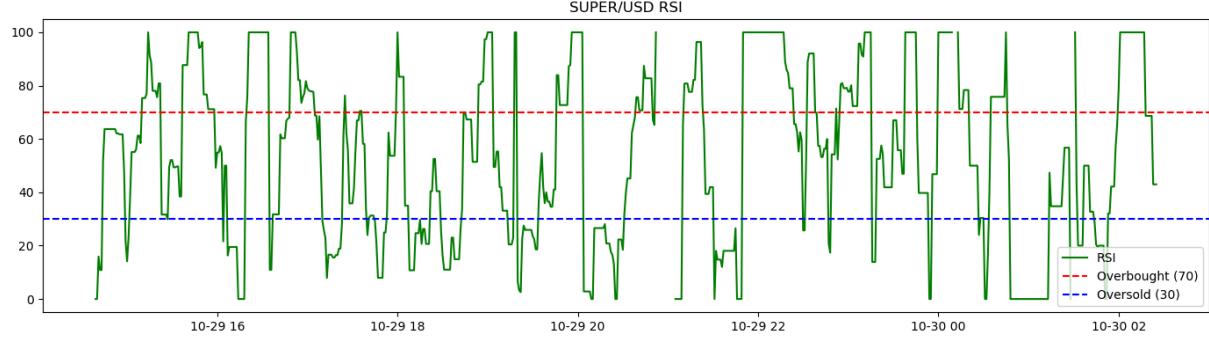
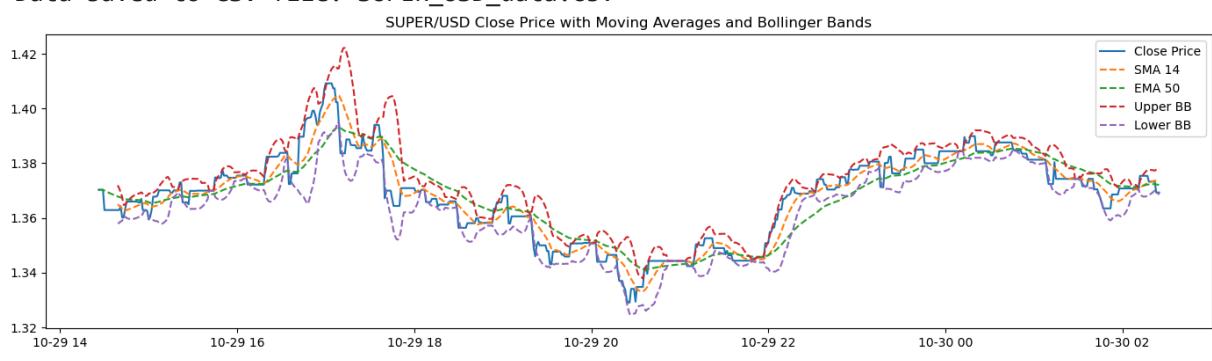
Data saved to CSV file: SAFE_USD_data.csv

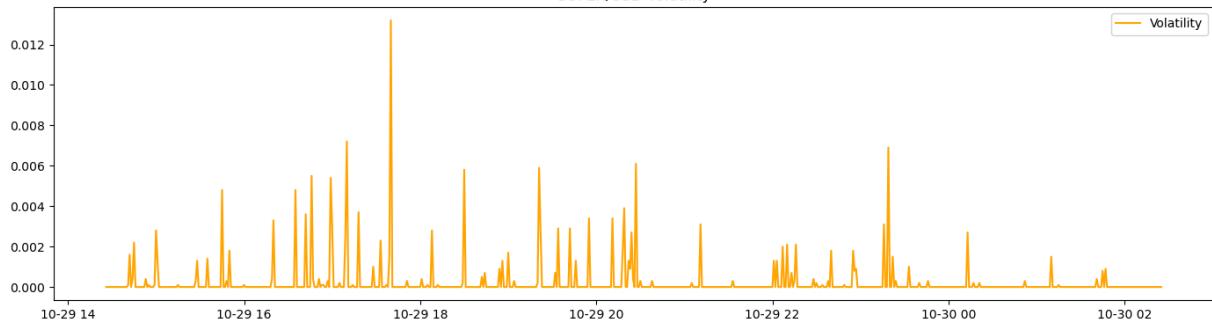
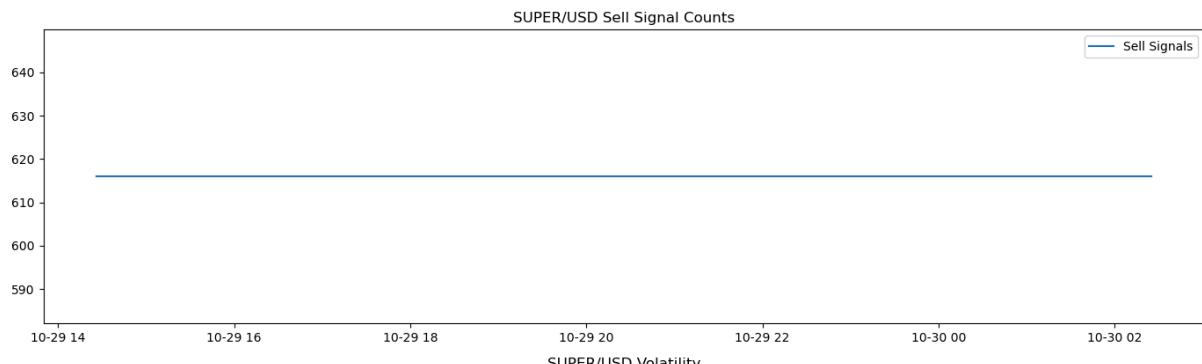




Data successfully saved to SQL table: SUPER_USD_data
SQL connection closed.

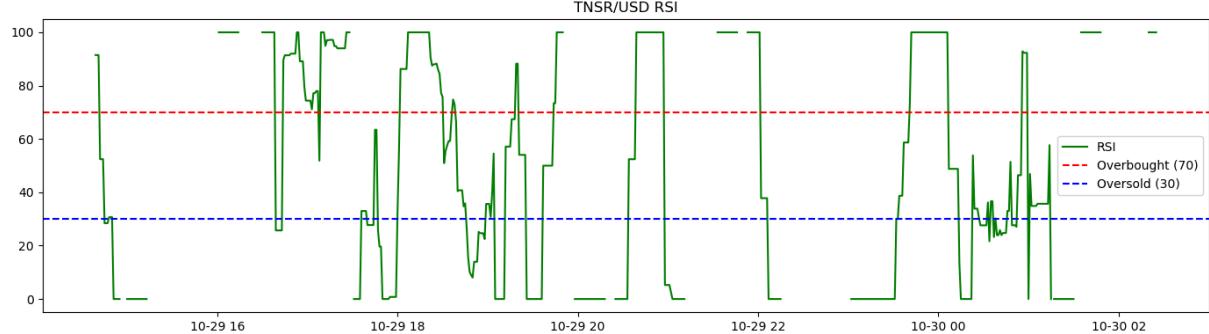
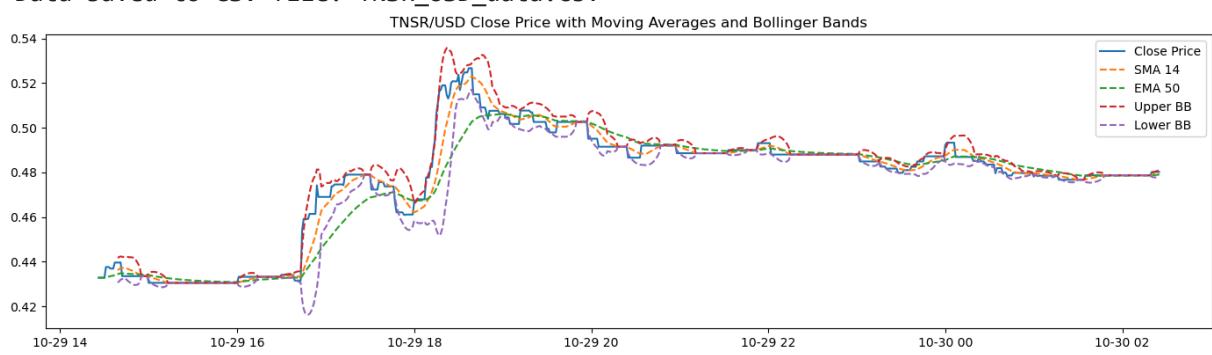
Data saved to CSV file: SUPER_USD_data.csv

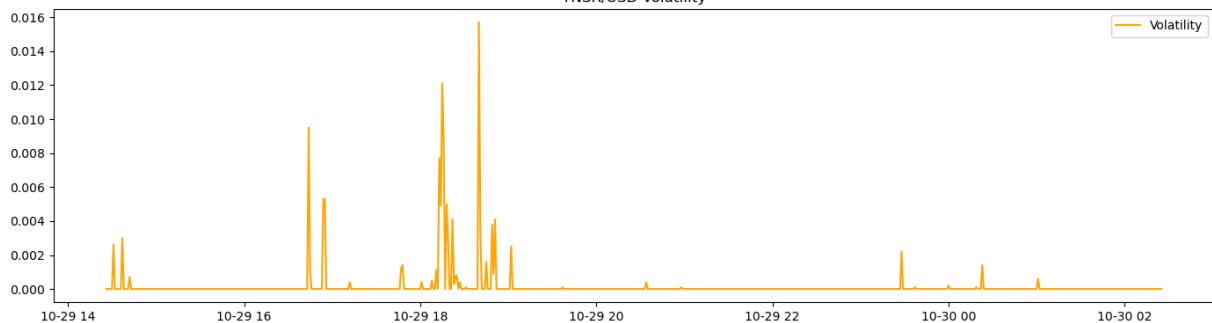
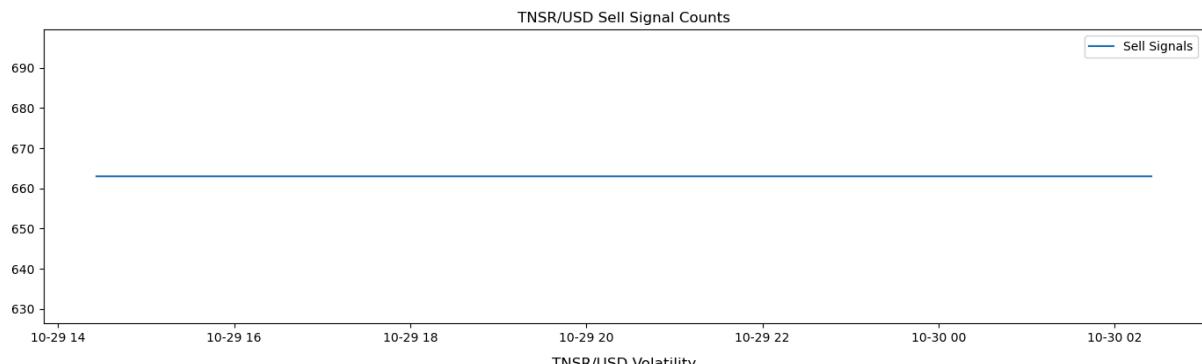




Data successfully saved to SQL table: TNSR_USD_data
SQL connection closed.

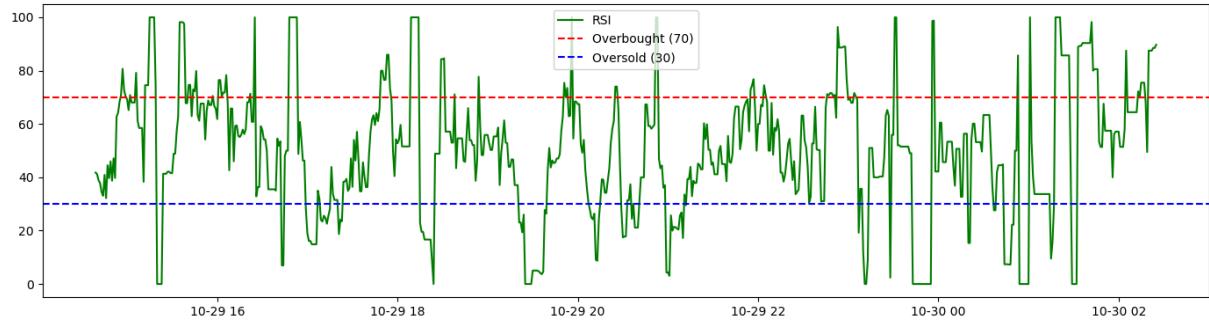
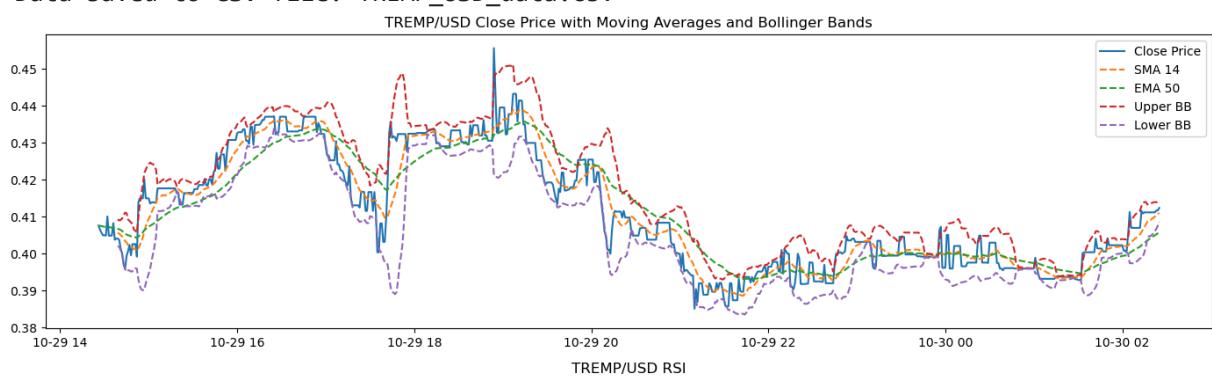
Data saved to CSV file: TNSR_USD_data.csv

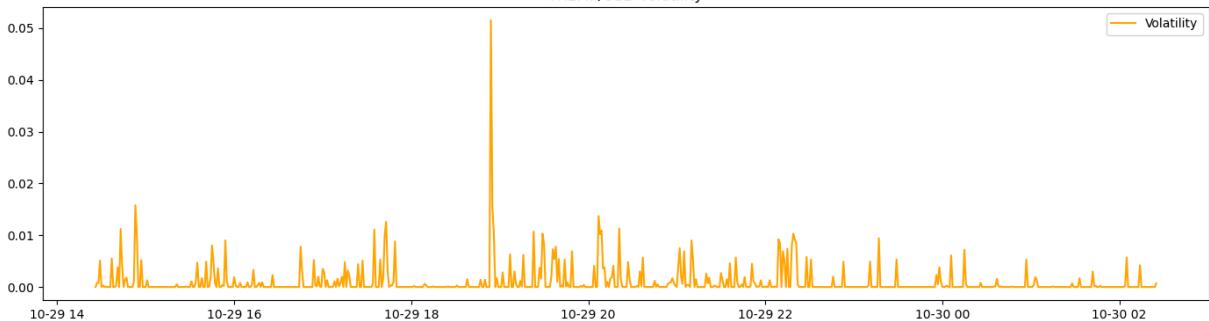
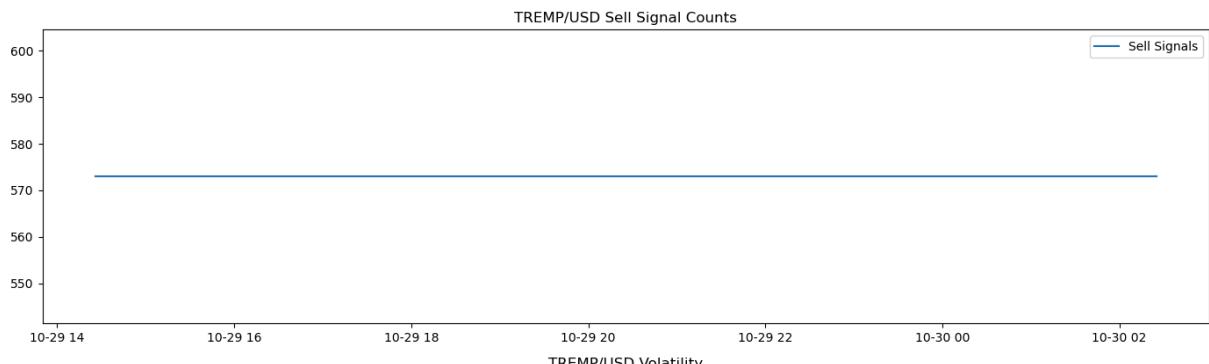




Data successfully saved to SQL table: TREMP_USD_data
SQL connection closed.

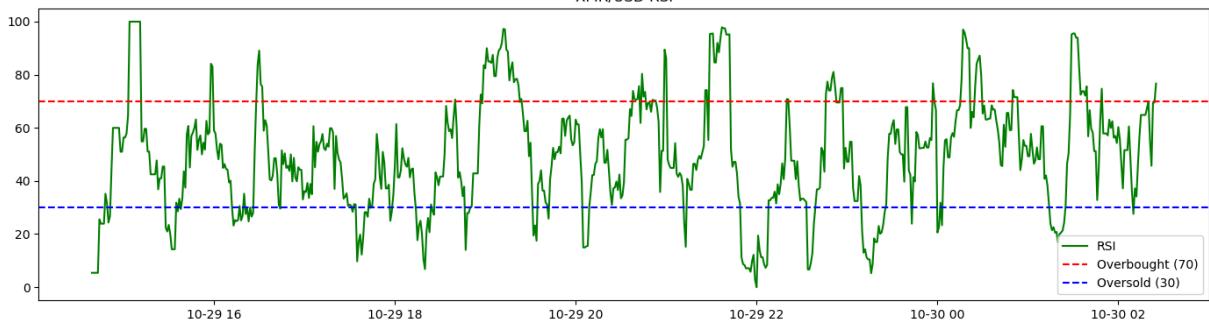
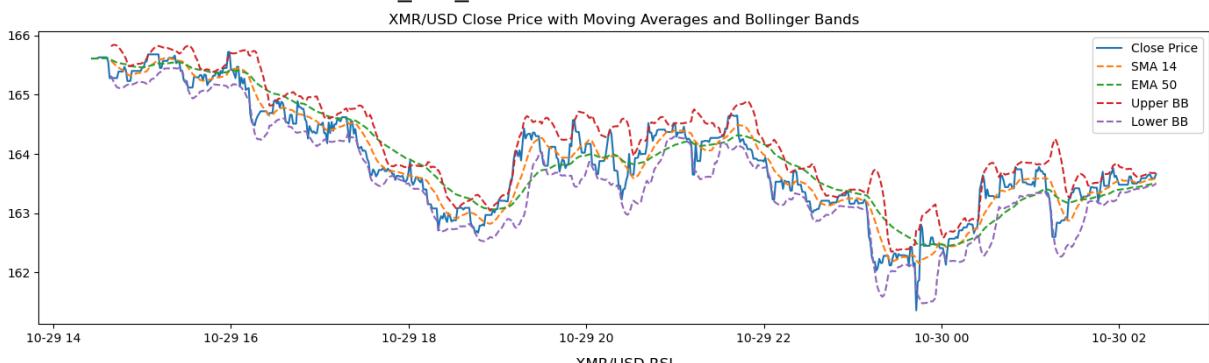
Data saved to CSV file: TREMP_USD_data.csv

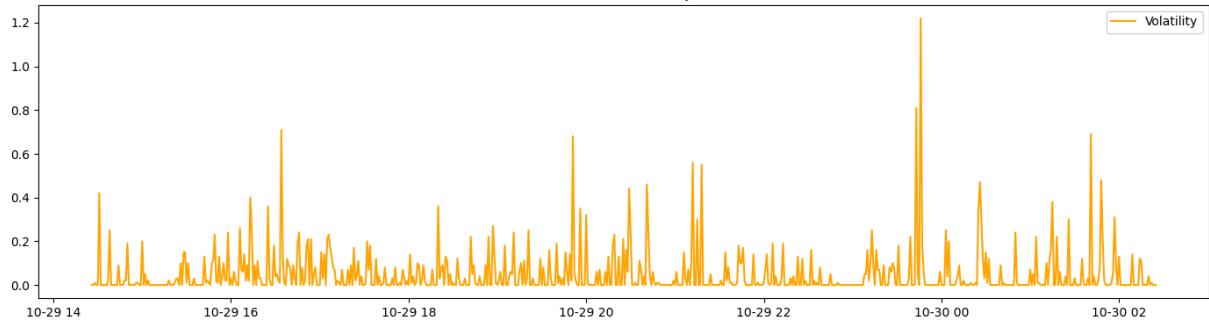
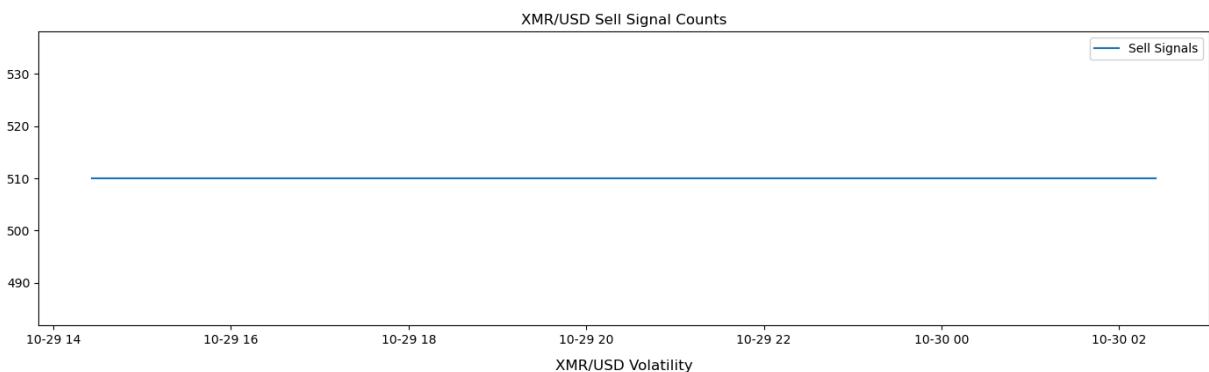




Data successfully saved to SQL table: XMR_USD_data
SQL connection closed.

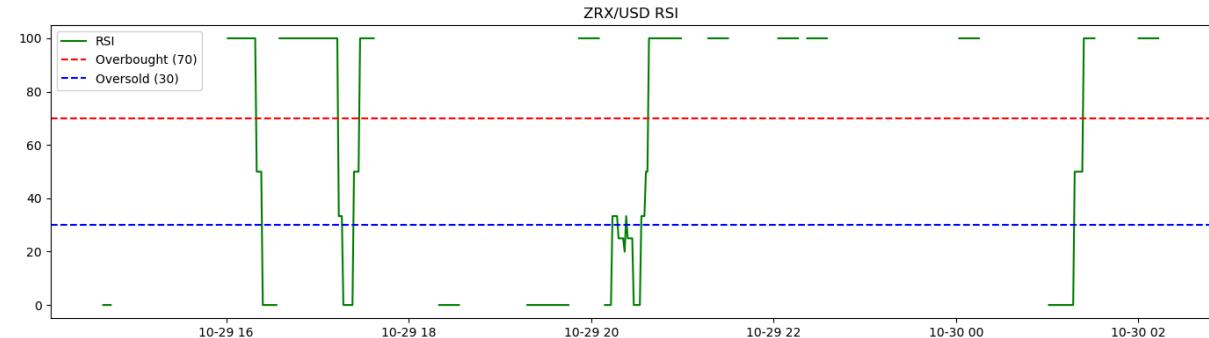
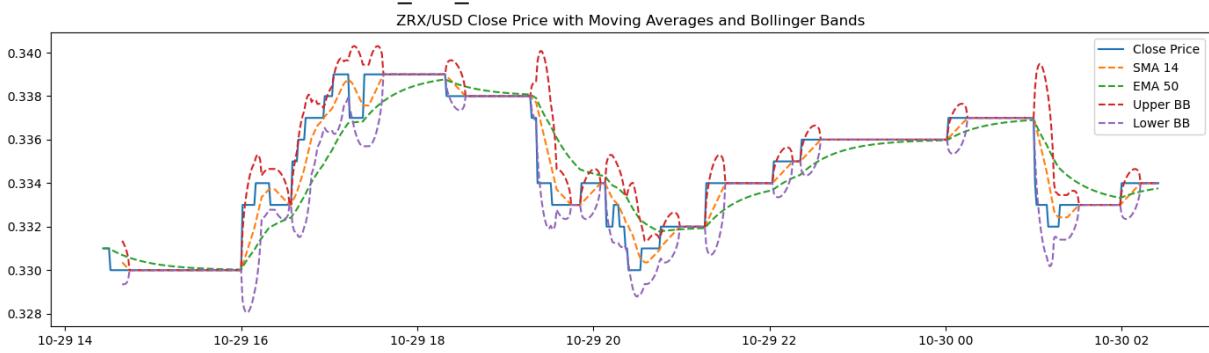
Data saved to CSV file: XMR_USD_data.csv

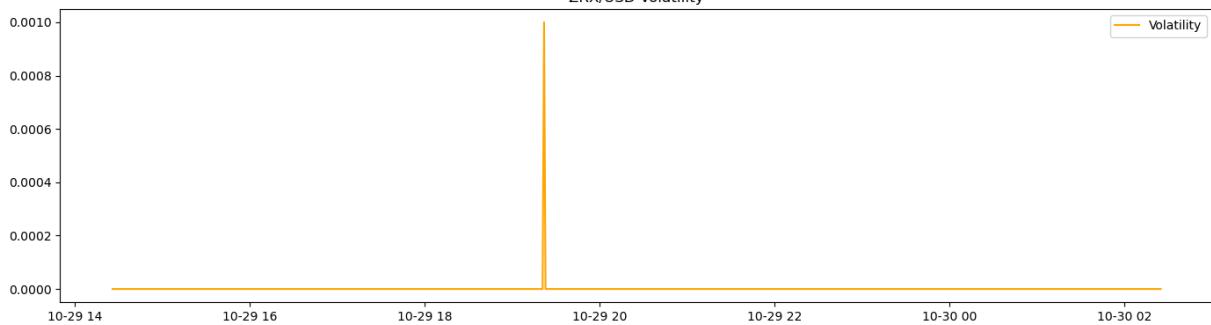
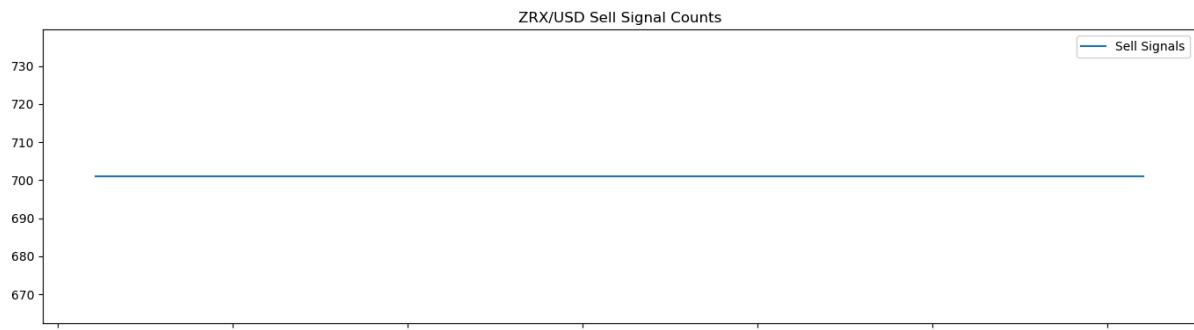




Data successfully saved to SQL table: ZRX_USD_data
SQL connection closed.

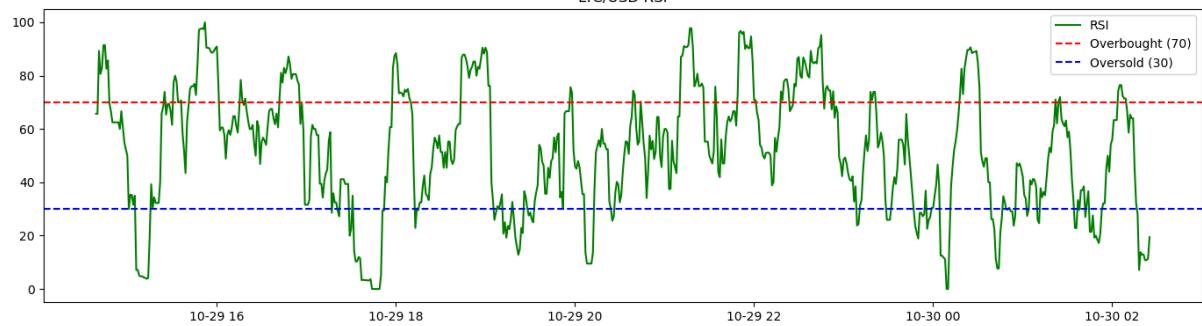
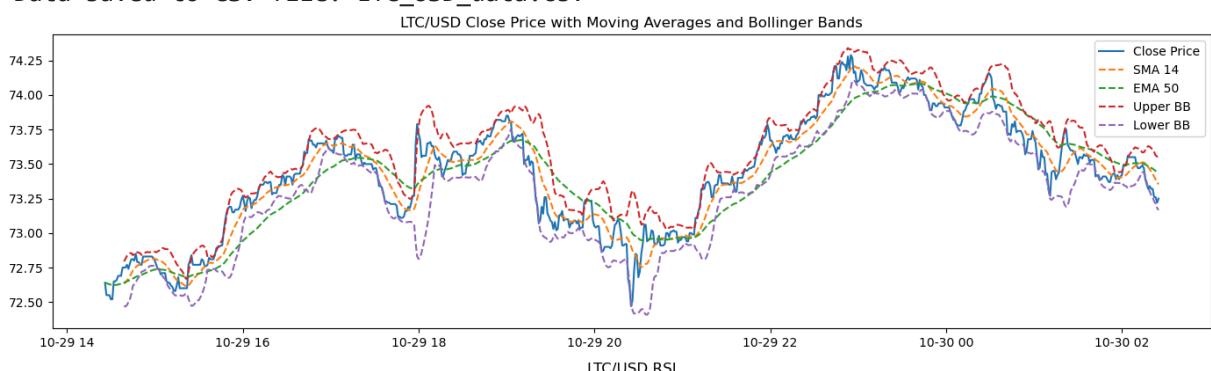
Data saved to CSV file: ZRX_USD_data.csv

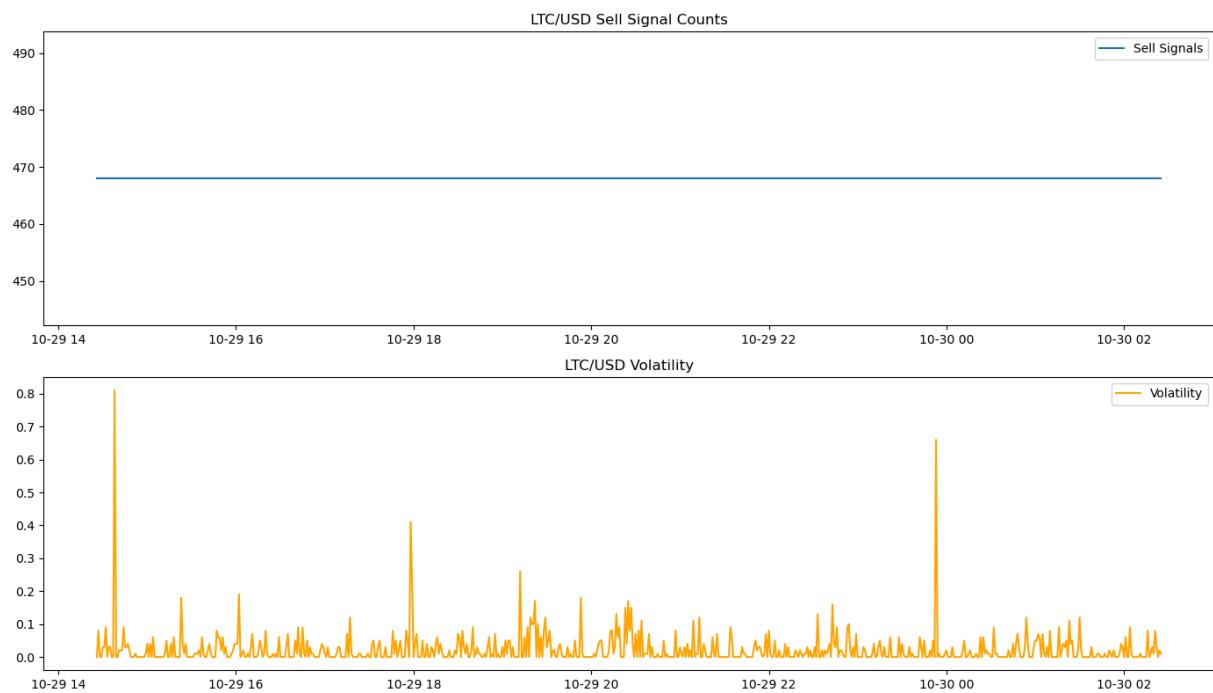




Data successfully saved to SQL table: LTC_USD_data
SQL connection closed.

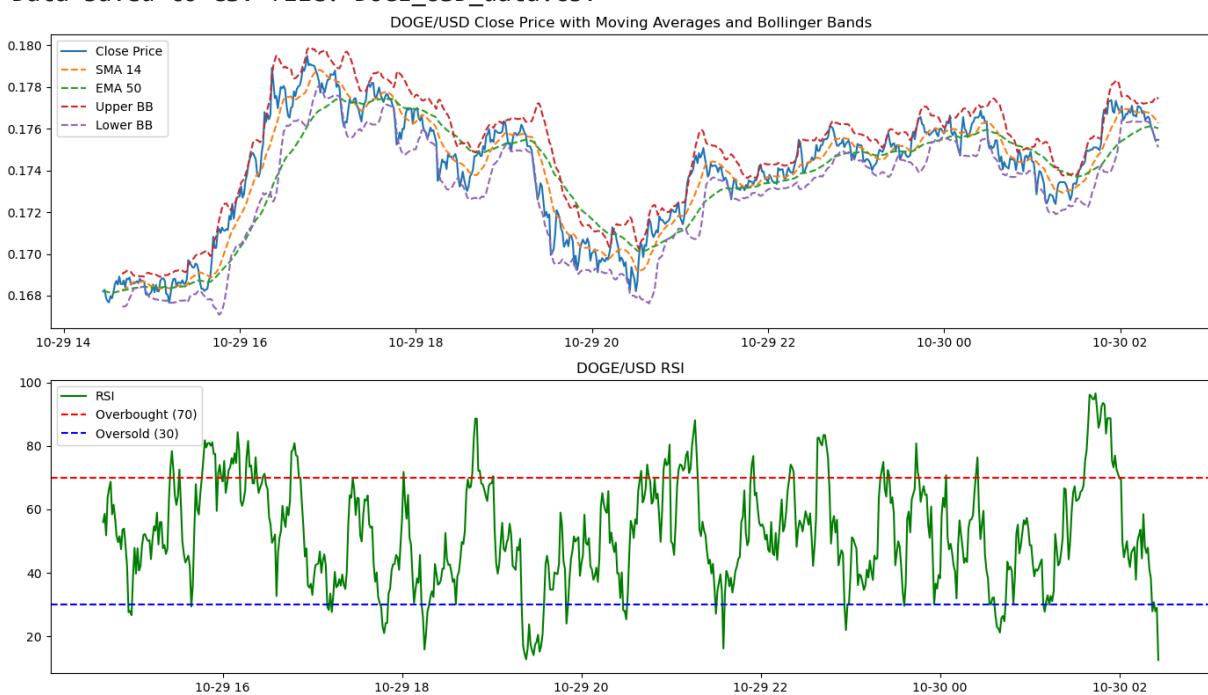
Data saved to CSV file: LTC_USD_data.csv

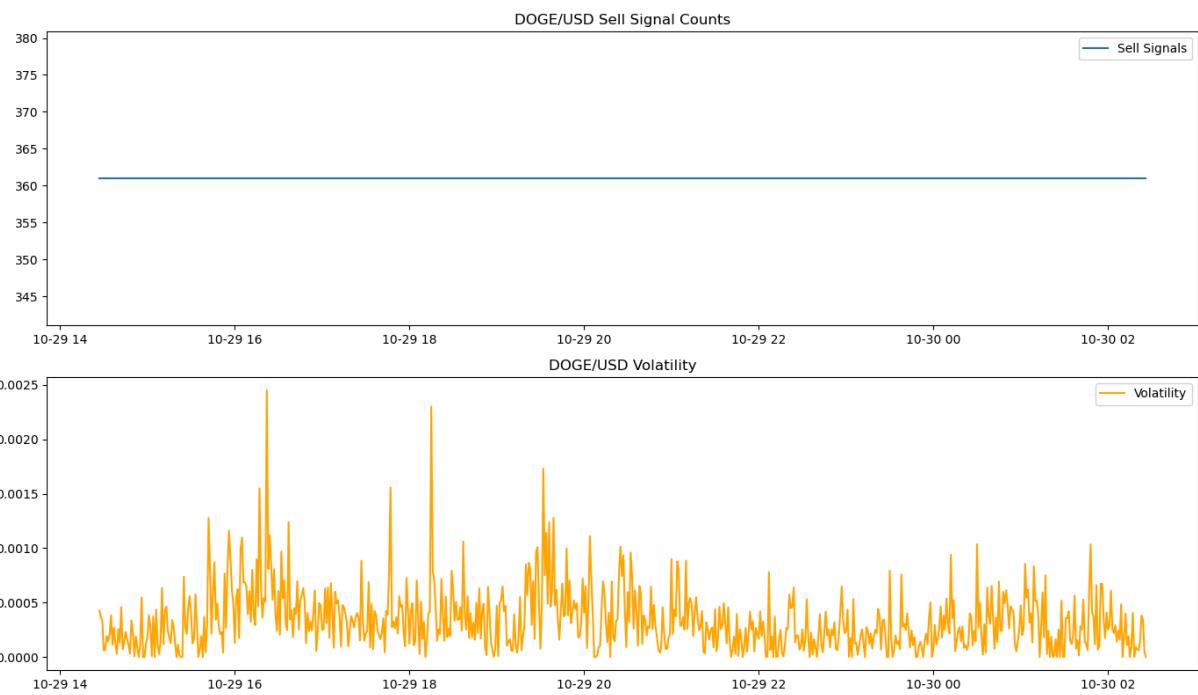




Data successfully saved to SQL table: DOGE_USD_data
SQL connection closed.

Data saved to CSV file: DOGE_USD_data.csv





Top 10 Symbols with Most 'SELL' Signals:

AUCTION/USD: 713 'SELL' signals
NTRN/USD: 710 'SELL' signals
MNT/USD: 709 'SELL' signals
GMT/USD: 703 'SELL' signals
ZRX/USD: 701 'SELL' signals
BODEN/USD: 695 'SELL' signals
MEME/USD: 694 'SELL' signals
SAFE/USD: 686 'SELL' signals
EUL/USD: 680 'SELL' signals
TNSR/USD: 663 'SELL' signals

Risk Classification:

ADA/USD: Low Risk

APE/USD: Low Risk

AUCTION/USD: Unknown Risk

BODEN/USD: Low Risk

BTC/USD: High Risk

CPOOL/USD: Low Risk

ETH/USD: High Risk

EUL/USD: Low Risk

GMT/USD: Low Risk

LINK/USD: Low Risk

USDT/USD: Low Risk

MEME/USD: Low Risk

MNT/USD: Low Risk

MOG/USD: Low Risk

NOS/USD: Low Risk

NTRN/USD: Low Risk

PYTH/USD: Low Risk

RENDER/USD: Low Risk

SAFE/USD: Low Risk

SUPER/USD: Low Risk

TNSR/USD: Low Risk

TREMP/USD: Low Risk

XMR/USD: Low Risk

ZRX/USD: Low Risk

LTC/USD: Low Risk

DOGE/USD: Low Risk

Cryptocurrency Volatility Dashboard

▼▼

Ada/usd Price Over Time



In []: