In [ ]:
```
!pip install ta
!pip install requests pandas numpy matplotlib ta
!pip install pandas numpy matplotlib scikit-learn requests scipy SQLAlchemy
!pip install statsmodels
!pip install dash
!pip freeze > requirements.txt
!conda install -c conda-forge ta-lib
!pip install sqlalchemy pyodbc
!pip install yfinance
!pip install streamlit
!pip install dash plotly
!pip install nbconvert[webpdf]
!pip install playwright
!pip install ccxt
```

In [5]:
```
import ccxt
import pandas as pd
import numpy as np
import talib as ta
import matplotlib.pyplot as plt
from datetime import datetime
from sqlalchemy import create_engine
import urllib
import os  # Import the os module for checking file existence

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

# Create a connection URI for SQLAlchemy
params = urllib.parse.quote_plus(f"DRIVER={DRIVER};SERVER={SERVER};DATABASE={DATABA
DATABASE_URI = f"{DATABASE_TYPE}+{DBAPI}:///?odbc_connect={params}"

# Create SQLAlchemy engine
engine = create_engine(DATABASE_URI, echo=False)

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

# Download historical data from Kraken
def get_crypto_data(symbol, timeframe='1m', since=None):
    ohlcv = kraken.fetch_ohlcv(symbol, timeframe=timeframe, since=since)
    df = pd.DataFrame(ohlcv, columns=['timestamp', 'Open', 'High', 'Low', 'Close',
    df['timestamp'] = pd.to_datetime(df['timestamp'], unit='ms')
    df.set_index('timestamp', inplace=True)
    return df

# Calculate 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

# Support and resistance levels
def find_support_resistance(df):
    df['support'] = df['Low'].rolling(window=60).min()
    df['resistance'] = df['High'].rolling(window=60).max()
    return df

# Moving Averages (SMA, EMA)
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

# 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

# Relative Strength Index (RSI)
def calculate_rsi(df, period=14):
    df['RSI'] = ta.RSI(df['Close'], timeperiod=period)
    return df

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

# Fibonacci retracements (simplified)
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

# MACD (Moving Average Convergence Divergence)
def calculate_macd(df):
    df['macd'], df['macdsignal'], df['macdhist'] = ta.MACD(df['Close'], fastperiod=
    return df

# Average True Range (ATR)
def calculate_atr(df, window=14):
    df['ATR'] = ta.ATR(df['High'], df['Low'], df['Close'], timeperiod=window)
    return df

# 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
```

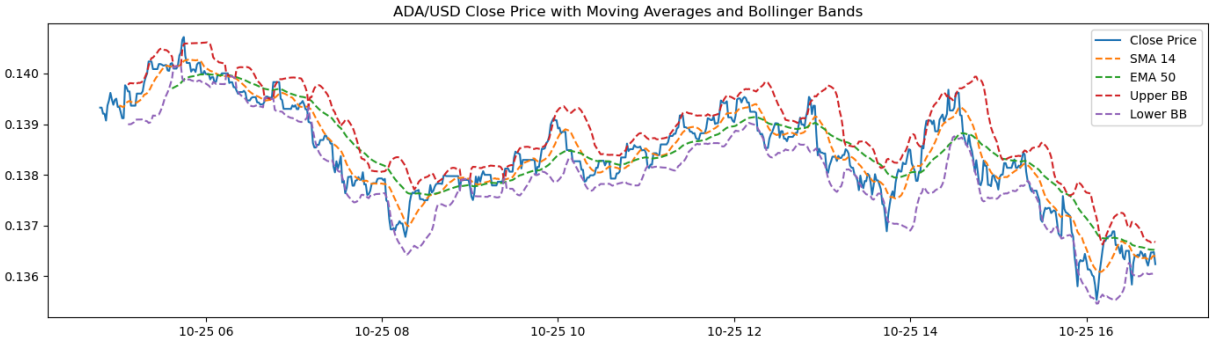
```python
# Ichimoku Cloud
def calculate_ichimoku(df):
    df['ichimoku_a'], df['ichimoku_b'], df['ichimoku_c'], df['ichimoku_d'], df['ich
    return df

# 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

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

# Chaikin Money Flow (CMF)
def calculate_cmf(df, window=20):
    df['CMF'] = ta.ADOSC(df['High'], df['Low'], df['Close'], df['Volume'], fastperi
    return df

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

# Verify and clean data
def clean_data(df):
    df.dropna(how='all', inplace=True)
    df.ffill(inplace=True)  # Forward fill missing data
    df.bfill(inplace=True)  # Backward fill missing data
    df.replace([np.inf, -np.inf], np.nan, inplace=True)
    df.dropna(inplace=True)
    return df

# Save data to SQL Server
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='timesta
        print(f"Data successfully saved to {table_name} in SQL Server.")
    except Exception as e:
        print(f"Error saving to SQL Server: {e}")
    finally:
        engine.dispose()
        print("SQL connection closed.")

# Save data to CSV
def save_to_csv(df, file_name):
    try:
        if df.empty:
            print("Data is empty after cleaning. Nothing to save.")
            return
        df.to_csv(file_name)
        print(f"Data successfully saved to {file_name}.")
```

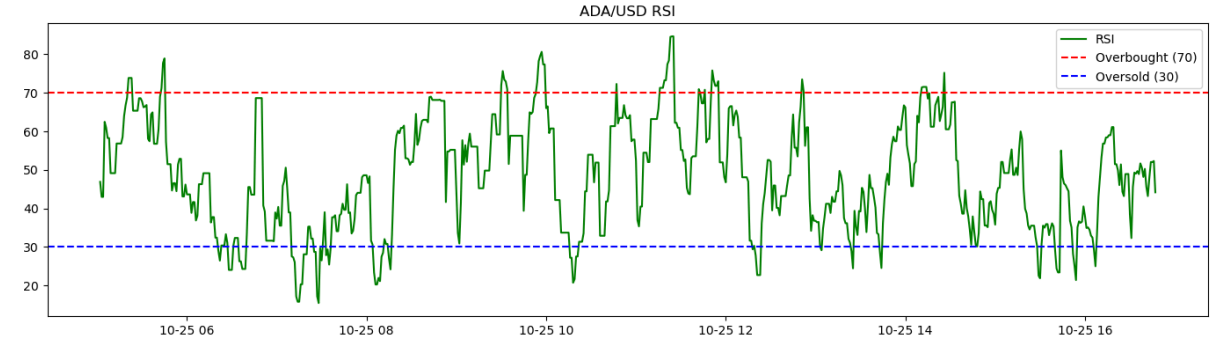
```
    except Exception as e:
        print(f"Error saving to CSV: {e}")

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

    # Plot Close Price, Moving Averages, and Bollinger Bands
    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()

    # Plot RSI
    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 Returns and Volatility
    plt.figure(figsize=(14, 8))
    plt.subplot(2, 1, 1)
    plt.plot(df.index, df['returns'], label='Returns')
    plt.title(f'{symbol} Returns')
    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()

# 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) >= thresho
    return df

# Integrate into main function after calculating other indicators
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/U
```

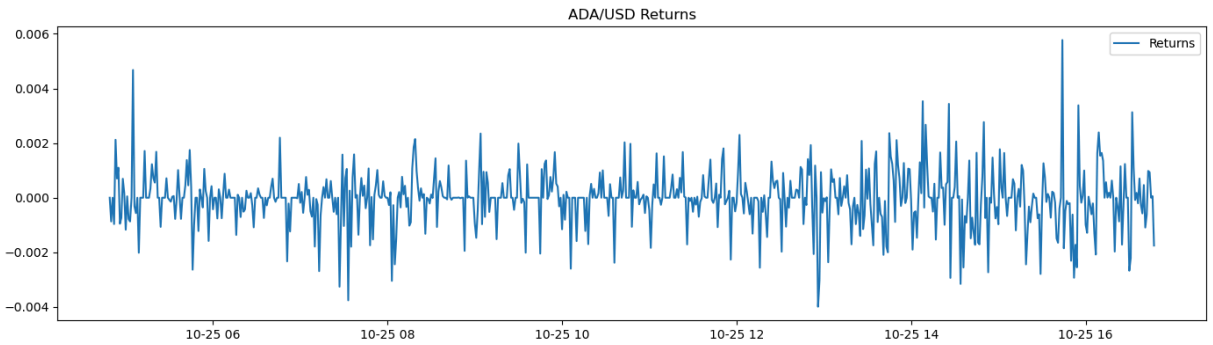
```
        'NTRN/USD', 'PYTH/USD', 'RENDER/USD', 'SAFE/USD', 'SUPER/USD', 'TNSR/USD',
        'XMR/USD', 'ZRX/USD', 'LTC/USD', 'DOGE/USD'
    ]

    timeframe = '1m'
    since = kraken.parse8601('2024-01-01T00:00:00Z')  # Starting point for data ret

    sell_counts = {}  # Store count of "SELL" signals for each symbol
    buy_counts = {}   # Store count of "BUY" signals for each symbol

    for symbol in symbols:
        print(f"\nFetching data for {symbol}...")
        df = get_crypto_data(symbol, timeframe, since)
        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)

        # Apply buy/sell signal calculation
        df = calculate_buy_sell_signal(df)

        # Count the number of "SELL" and "BUY" signals
        sell_counts[symbol] = df['Signal'].value_counts().get('SELL', 0)
        buy_counts[symbol] = df['Signal'].value_counts().get('BUY', 0)

    # Display the top 5 symbols with the most "SELL" signals
    top_sell_symbols = sorted(sell_counts.items(), key=lambda x: x[1], reverse=True
    print("\nTop 5 Symbols with Most 'SELL' Signals:")
    for symbol, count in top_sell_symbols:
        print(f"{symbol}: {count} 'SELL' signals")

    # Display the top 5 symbols with the most "BUY" signals
    top_buy_symbols = sorted(buy_counts.items(), key=lambda x: x[1], reverse=True)[
    print("\nTop 5 Symbols with Most 'BUY' Signals:")
    for symbol, count in top_buy_symbols:
        print(f"{symbol}: {count} 'BUY' signals")

        # Save to SQL and CSV
        table_name = symbol.replace('/', '_').lower()
        save_to_sql(df, table_name)

        # Save data as CSV file
        csv_file_name = f"{symbol.replace('/', '_').lower()}.csv"
        save_to_csv(df, csv_file_name)

        # Plot data
        plot_data(df, symbol)

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

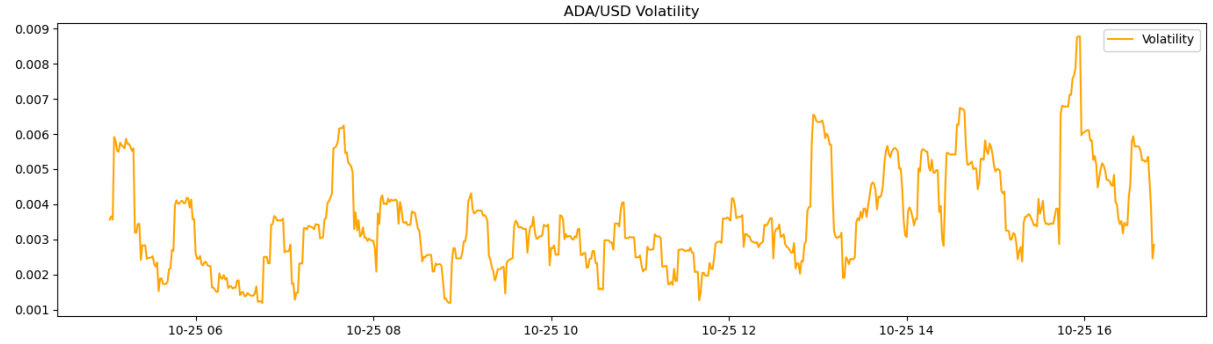
```
Fetching data for ADA/USD...

Fetching data for APE/USD...

Fetching data for AUCTION/USD...

Fetching data for BODEN/USD...

Fetching data for BTC/USD...

Fetching data for CPOOL/USD...

Fetching data for ETH/USD...

Fetching data for EUL/USD...

Fetching data for GMT/USD...

Fetching data for LINK/USD...

Fetching data for USDT/USD...

Fetching data for MEME/USD...

Fetching data for MNT/USD...

Fetching data for MOG/USD...

Fetching data for NOS/USD...

Fetching data for NTRN/USD...

Fetching data for PYTH/USD...

Fetching data for RENDER/USD...

Fetching data for SAFE/USD...

Fetching data for SUPER/USD...

Fetching data for TNSR/USD...

Fetching data for TREMP/USD...

Fetching data for USDT/USD...

Fetching data for XMR/USD...

Fetching data for LINK/USD...

Fetching data for XMR/USD...

Fetching data for ZRX/USD...

Fetching data for LTC/USD...
```

```
Fetching data for DOGE/USD...

Top 5 Symbols with Most 'SELL' Signals:
ADA/USD: 0 'SELL' signals
APE/USD: 0 'SELL' signals
AUCTION/USD: 0 'SELL' signals
BODEN/USD: 0 'SELL' signals
BTC/USD: 0 'SELL' signals

Top 5 Symbols with Most 'BUY' Signals:
ADA/USD: 720 'BUY' signals
Data successfully saved to ada_usd in SQL Server.
SQL connection closed.
Data successfully saved to ada_usd.csv.
```

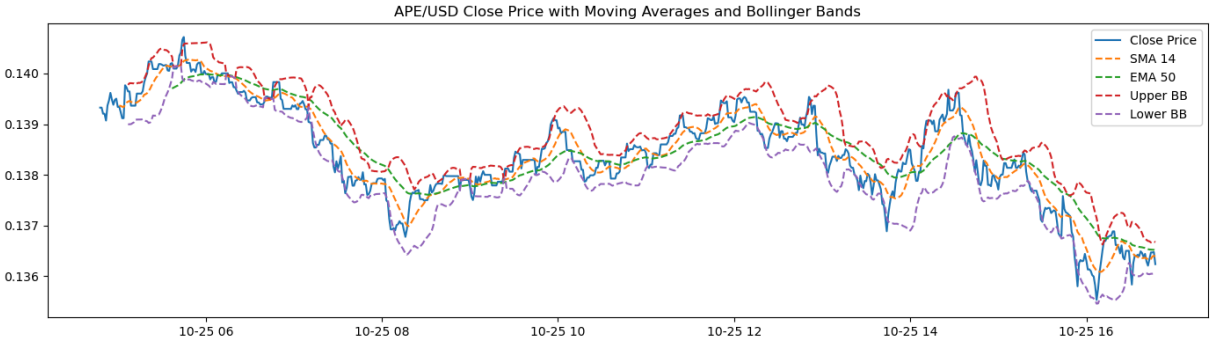


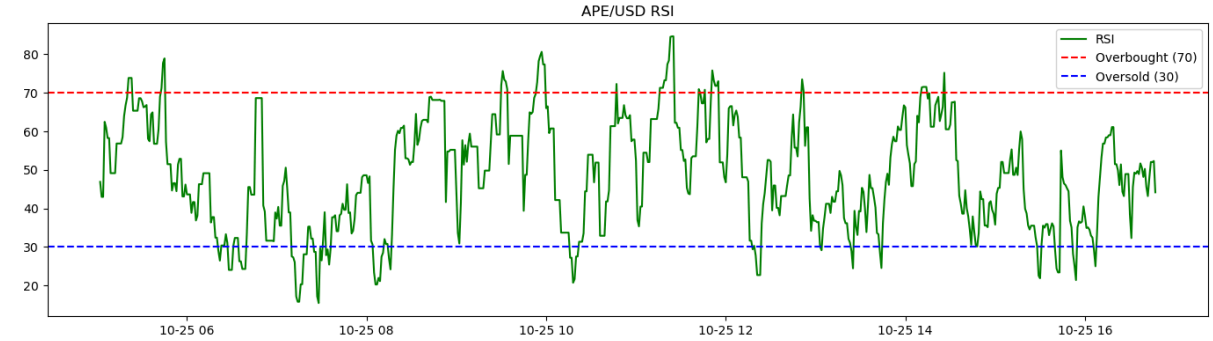


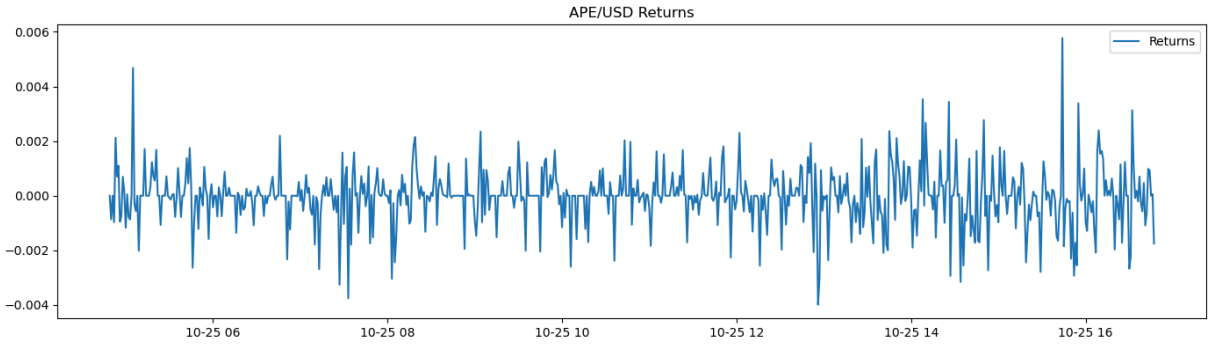


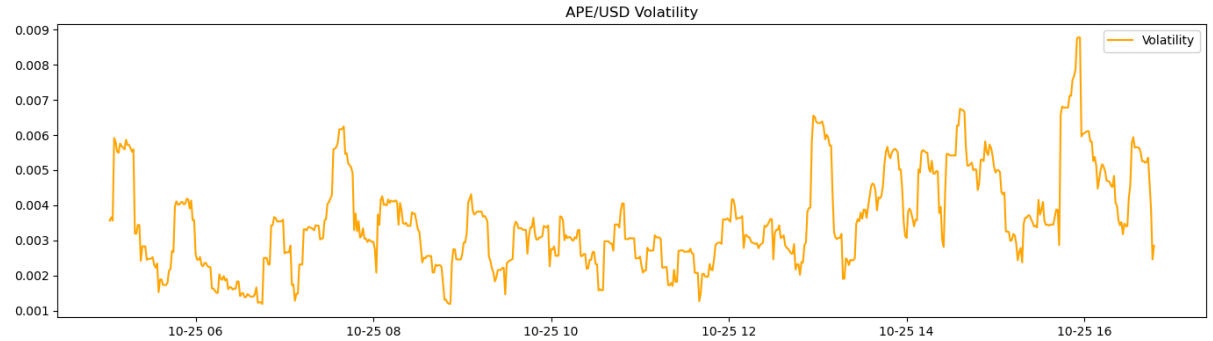


APE/USD: 720 'BUY' signals
Data successfully saved to ape_usd in SQL Server.
SQL connection closed.
Data successfully saved to ape_usd.csv.

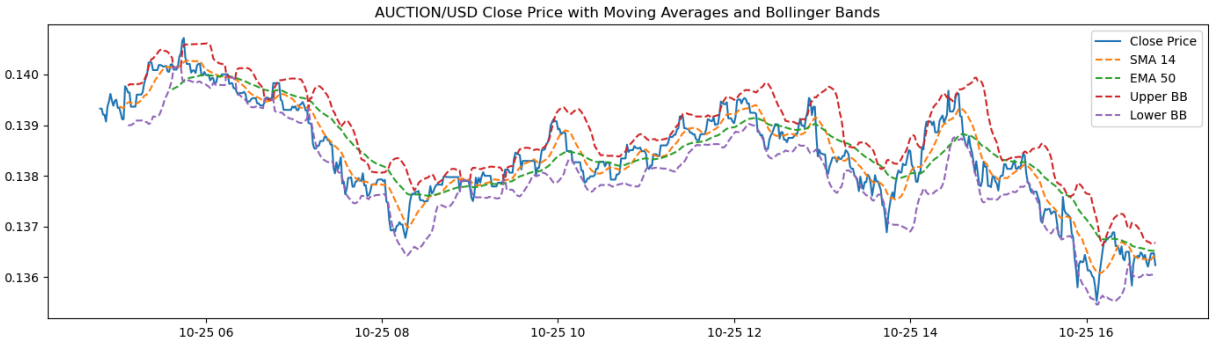


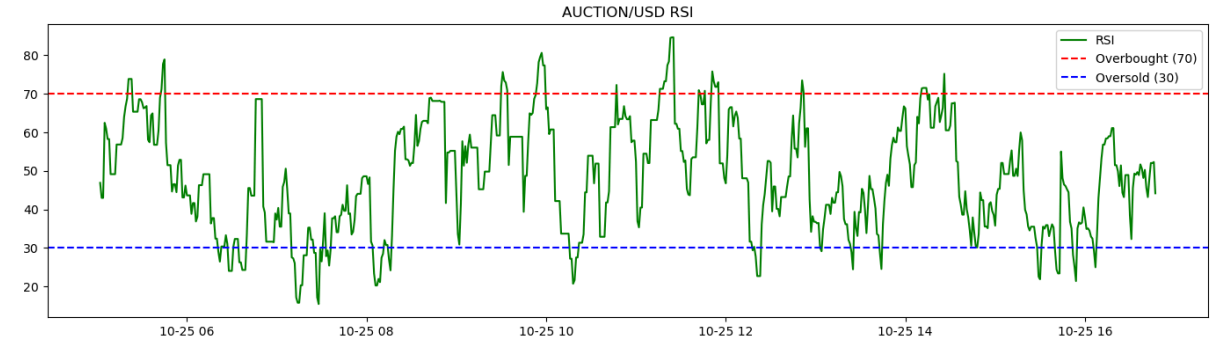


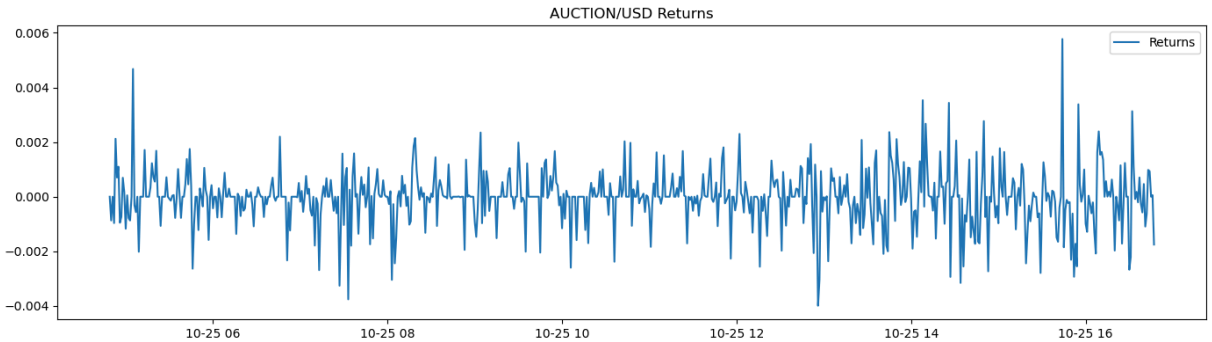


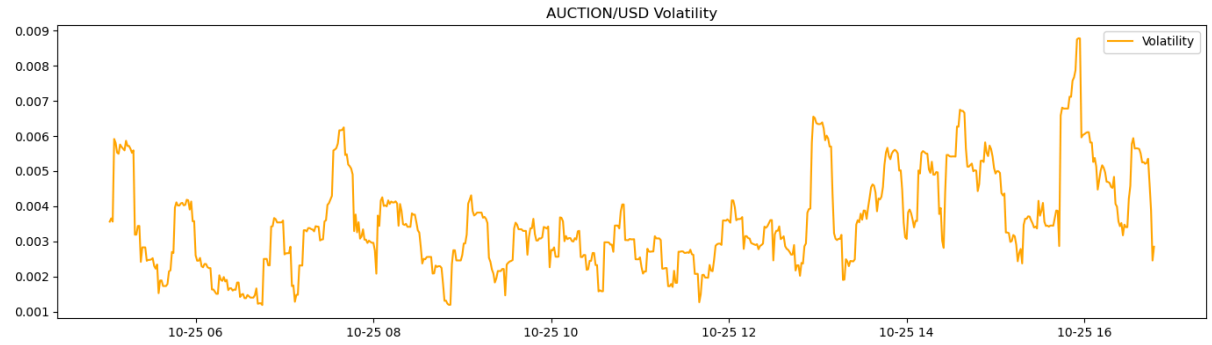


AUCTION/USD: 720 'BUY' signals
Data successfully saved to auction_usd in SQL Server.
SQL connection closed.
Data successfully saved to auction_usd.csv.

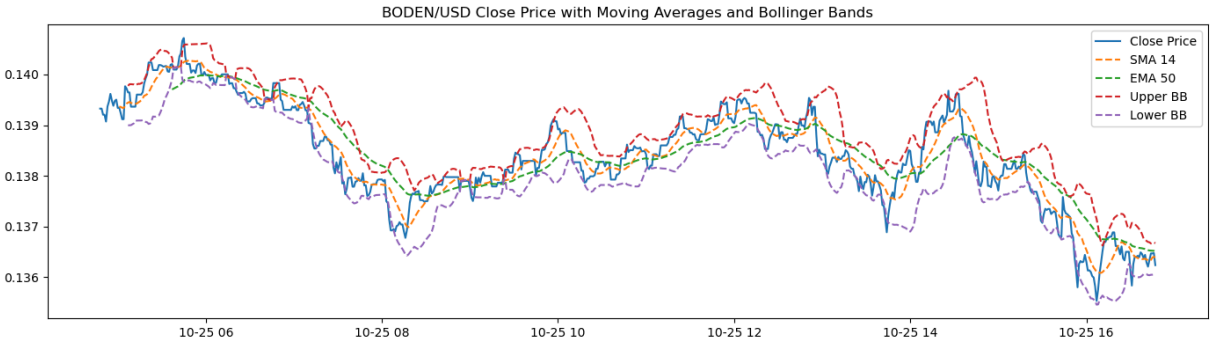


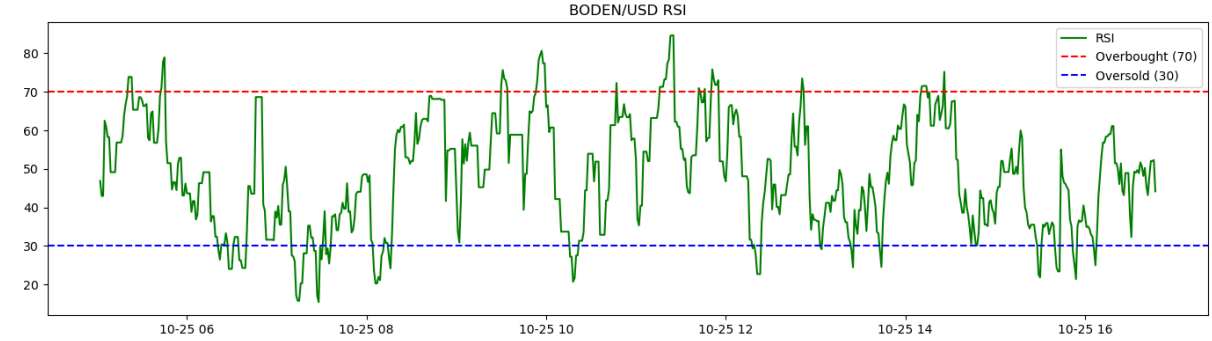


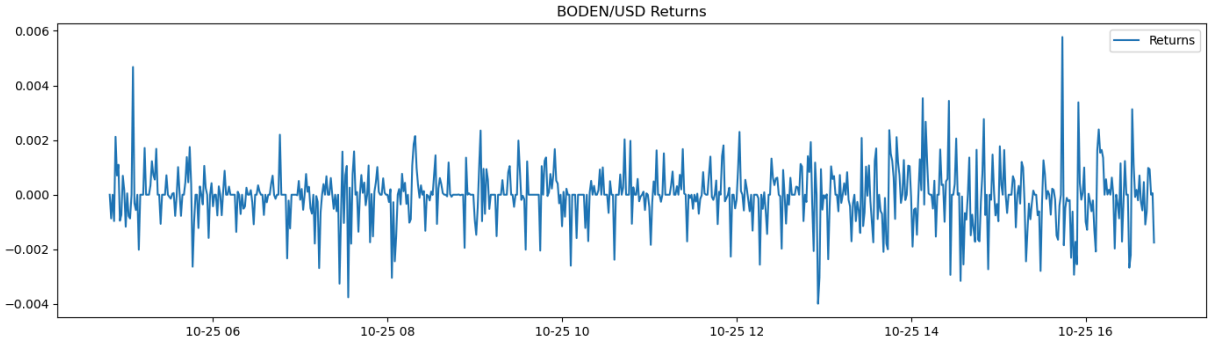


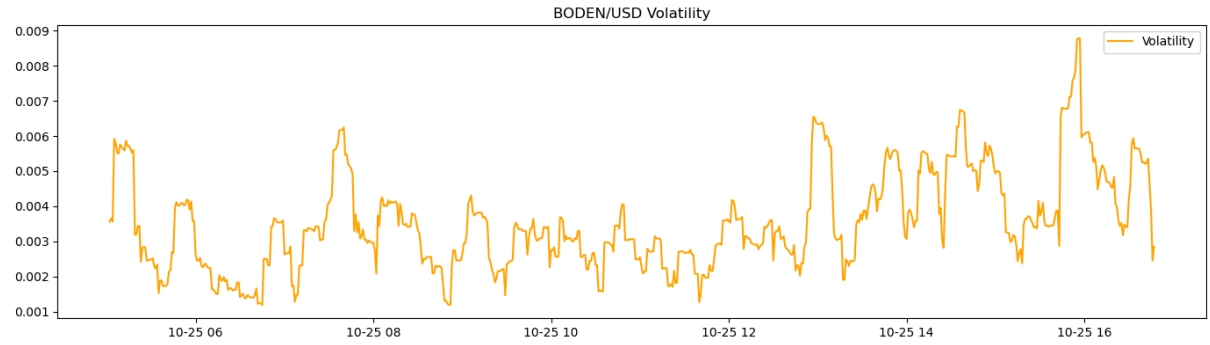


BODEN/USD: 720 'BUY' signals
Data successfully saved to boden_usd in SQL Server.
SQL connection closed.
Data successfully saved to boden_usd.csv.

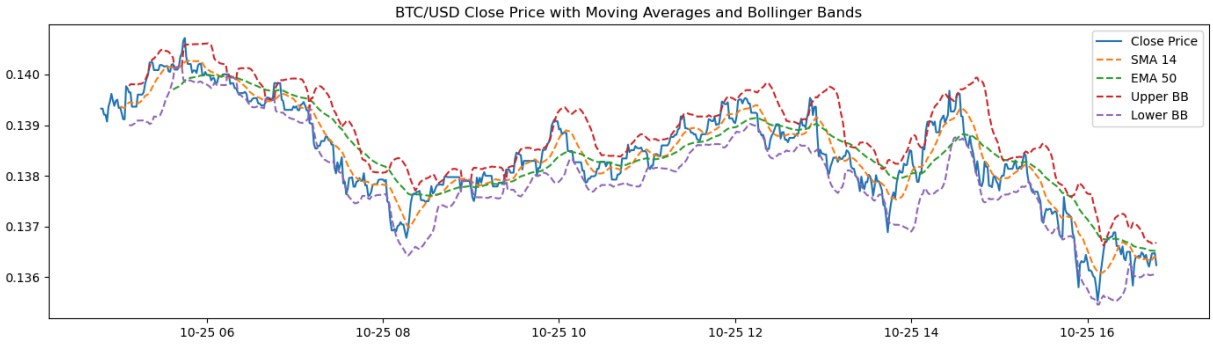


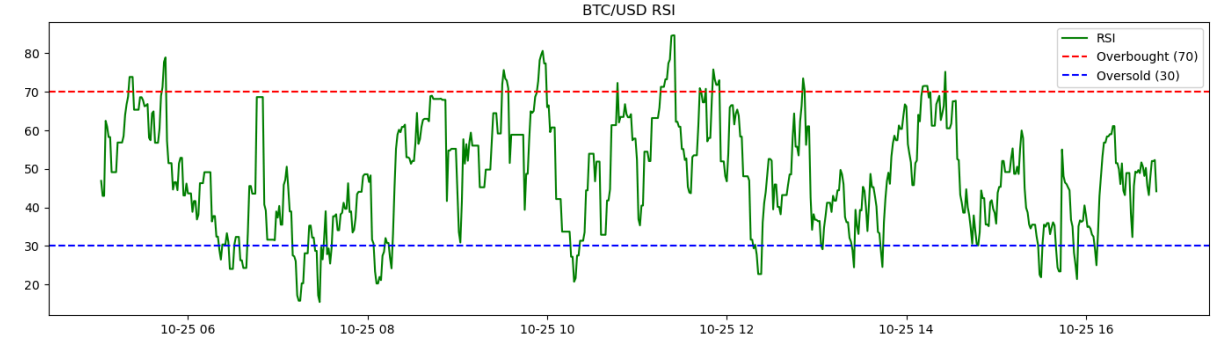


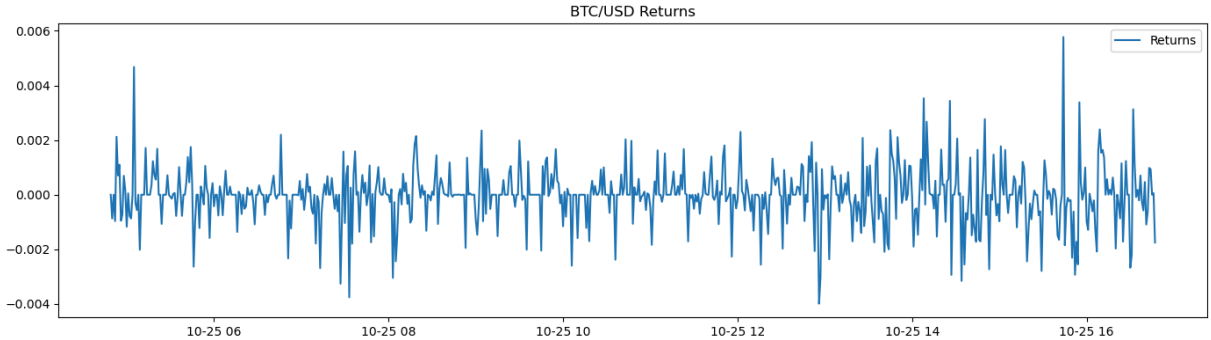


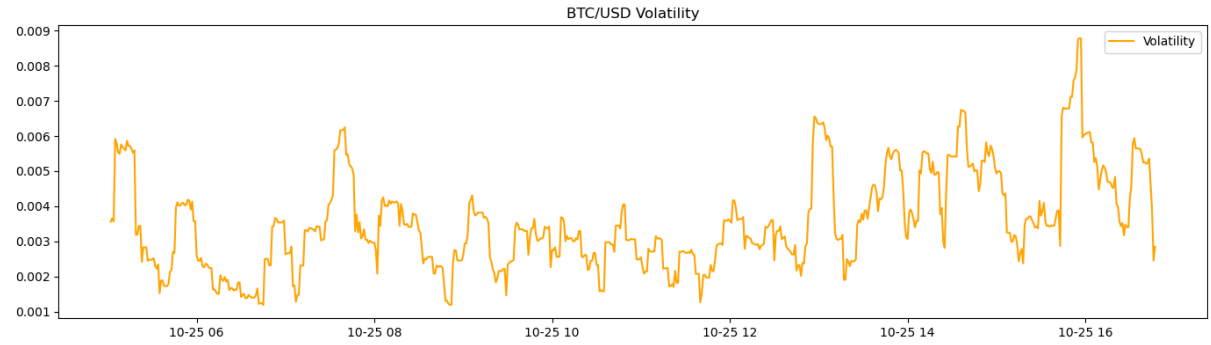


BTC/USD: 720 'BUY' signals
Data successfully saved to btc_usd in SQL Server.
SQL connection closed.
Data successfully saved to btc_usd.csv.


BTC/USD Close Price with Moving Averages and Bollinger Bands
Close Price
SMA 14
EMA 50
Upper BB
Lower BB
0.140
0.139
0.138
0.137
0.136
10-25 06
10-25 08
10-25 10
10-25 12
10-25 14
10-25 16


BTC/USD RSI
RSI
Overbought (70)
Oversold (30)
80
70
60
50
40
30
20
10-25 06
10-25 08
10-25 10
10-25 12
10-25 14
10-25 16


BTC/USD Returns
Returns
0.006
0.004
0.002
0.000
−0.002
−0.004
10-25 06
10-25 08
10-25 10
10-25 12
10-25 14
10-25 16


BTC/USD Volatility
Volatility
0.009
0.008
0.007
0.006
0.005
0.004
0.003
0.002
0.001
10-25 06
10-25 08
10-25 10
10-25 12
10-25 14
10-25 16