

In [295...]

```
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
```

In [296...]

```
# 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={DATABASE}")
DATABASE_URI = f"{DATABASE_TYPE}+{DBAPI}://?odbc_connect={params}"

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

In [297...]

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

In [298...]

```
# 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',
                                      'volume'])
    df['timestamp'] = pd.to_datetime(df['timestamp'], unit='ms')
    df.set_index('timestamp', inplace=True)
    return df
```

In [299...]

```
# 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 [300...]

```
# 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 [301...]

```
# 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 [302...]: # Calculate 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'], timeperiod=window, nbdev=2)
    return df

In [303...]: # Calculate the RSI
def calculate_rsi(df, period=14):
    df['RSI'] = ta.RSI(df['Close'], timeperiod=period)
    return df

In [304...]: # Calculate the VWAP
def calculate_vwap(df):
    df['vwap'] = (df['Volume'] * (df['High'] + df['Low'] + df['Close']) / 3).cumsum()
    return df

In [305...]: # Calculate 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 [306...]: # Calculate the MACD
def calculate_macd(df):
    df['macd'], df['macdsignal'], df['macdhist'] = ta.MACD(df['Close'], fastperiod=12, slowperiod=26, signalperiod=9)
    return df

In [307...]: # Calculate the ATR
def calculate_atr(df, window=14):
    df['ATR'] = ta.ATR(df['High'], df['Low'], df['Close'], timeperiod=window)
    return df

In [308...]: # 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=5, slowk_p=k_window, slowd_p=d_window)
    return df

In [309...]: # Ichimoku Cloud
def calculate_ichimoku(df):
    df['ichimoku_a'], df['ichimoku_b'], df['ichimoku_c'], df['ichimoku_d'], df['ichimoku_l'] = ta.ICHIMOKU(df['High'], df['Low'], df['Close'], tenkan_sen=9, kijun_sen=26, chikou=52)
    return df

In [310...]: # 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 [311... # 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 [312... # Chaikin Money Flow (CMF)
def calculate_cmf(df, window=20):
    df['CMF'] = ta.ADOOSC(df['High'], df['Low'], df['Close'], df['Volume'], fastperiod=13, slowperiod=34, window=window)
    return df
```

```
In [313... # On-Balance Volume (OBV)
def calculate_obv(df):
    df['OBV'] = ta.OBV(df['Close'], df['Volume'])
    return df
```

```
In [314... # Sweep and Clean the data
def clean_data(df):
    df.dropna(how='all', inplace=True)
    df.fillna(method='ffill') # 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
```

```
In [315... # Create the table from the return data in SQL and save
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 {table_name} in SQL Server.")
    except Exception as e:
        print(f"Error saving to SQL Server: {e}")
    finally:
        engine.dispose()
        print("SQL connection closed.")
```

```
In [316... # 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}")
```

```
In [317... # 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
return df
```

In [318...]

```
# 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()
```

In [319...]

```
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/USD', '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 retrieval

    # Initialize dictionaries for buy/sell counts and risk classification
```

```

sell_counts = {}
buy_counts = {}
risk_labels = {}

# Define a threshold for volatility (or ATR) to classify "High Risk" or "Low Risk"
risk_threshold = 0.05 # threshold; adjust based on volatility range for the data

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)

    # Calculate risk classification based on Volatility or ATR
    if 'Volatility' in df.columns:
        avg_volatility = df['Volatility'].mean()
    elif 'ATR' in df.columns:
        avg_volatility = df['ATR'].mean()
    else:
        print(f"Warning: Neither 'Volatility' nor 'ATR' columns found for {symbol}")
        avg_volatility = None

    # Determine the risk label if avg_volatility is available
    if avg_volatility is not None:
        risk_labels[symbol] = "High Risk" if avg_volatility > risk_threshold else "Low Risk"
    else:
        risk_labels[symbol] = "Unknown Risk"

    # 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")

    # Display risk classification for each symbol

```

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

# Save data to SQL Server and CSV, and plot for each symbol
for symbol in symbols:
    table_name = symbol.replace('/', '_').lower()
    save_to_sql(df, table_name)

    csv_file_name = f"{symbol.replace('/', '_').lower()}.csv"
    save_to_csv(df, csv_file_name)

    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
APE/USD: 720 'BUY' signals
AUCTION/USD: 720 'BUY' signals
BODEN/USD: 720 'BUY' signals
BTC/USD: 720 'BUY' signals

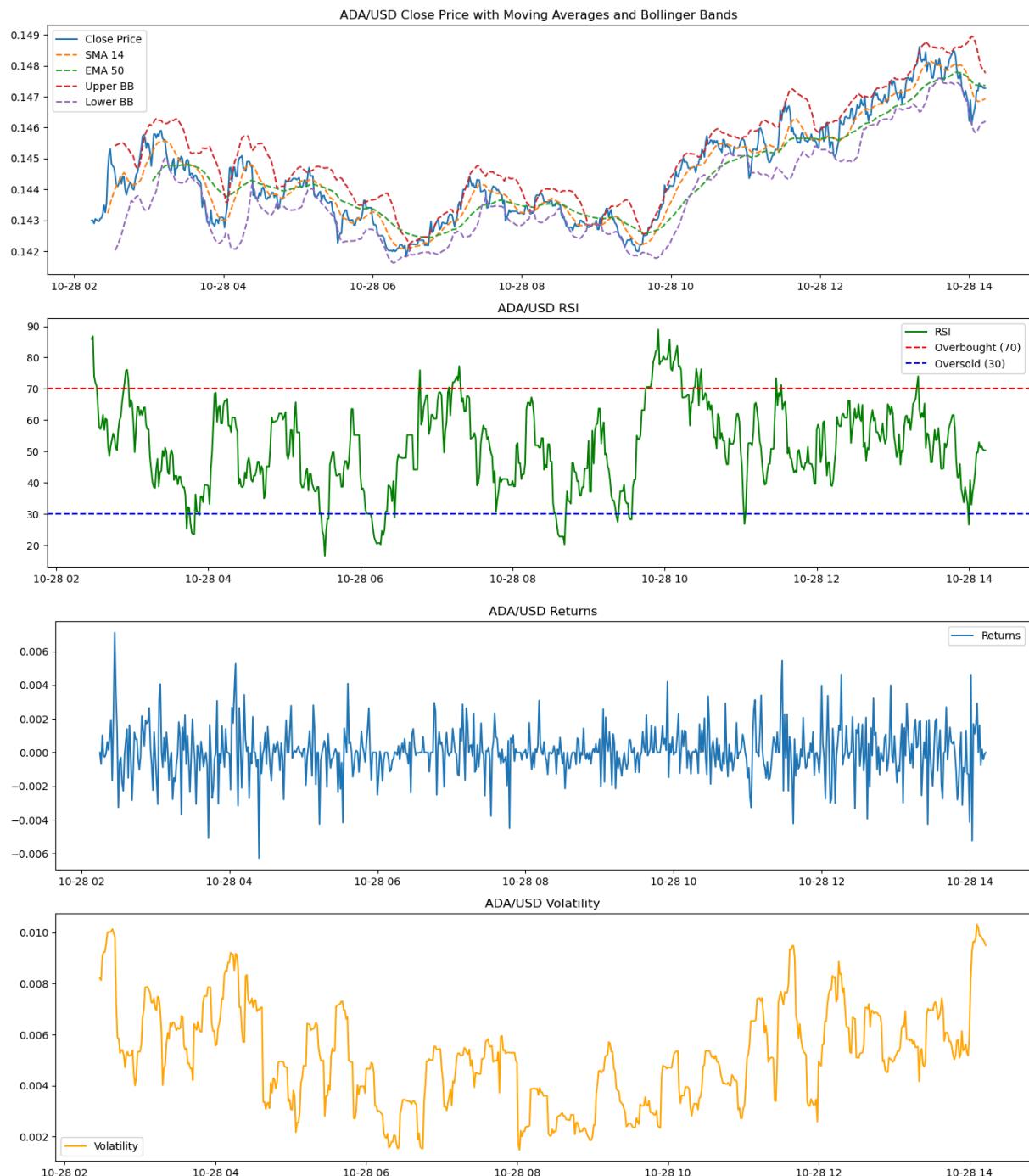
Risk Classification:

ADA/USD: Low Risk
APE/USD: Low Risk
AUCTION/USD: Low 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: High Risk
ZRX/USD: Low Risk
LTC/USD: Low Risk
DOGE/USD: Low Risk

Data successfully saved to ada_usd in SQL Server.

SQL connection closed.

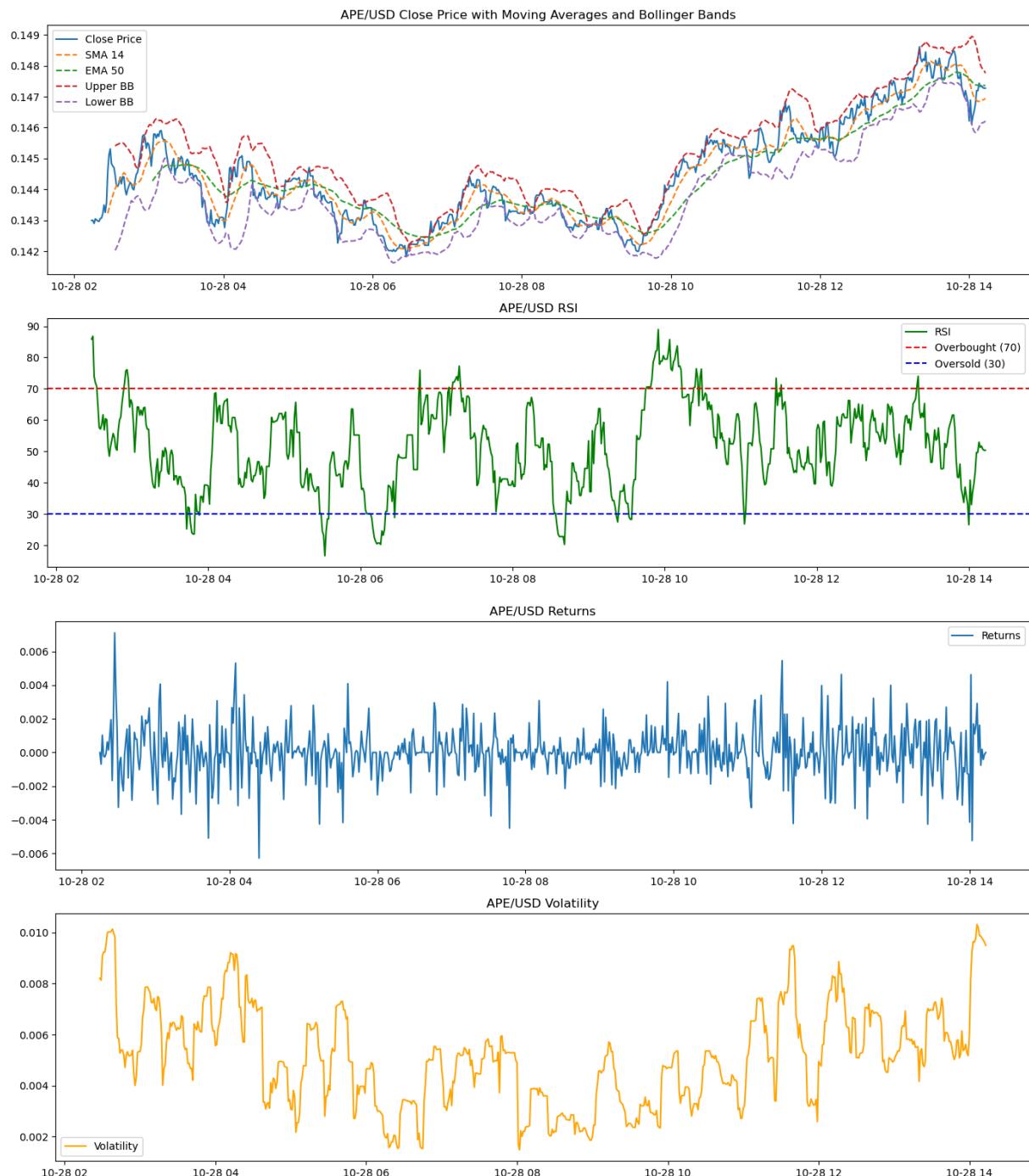
Data successfully saved to ada_usd.csv.



Data successfully saved to `ape_usd` in SQL Server.

SQL connection closed.

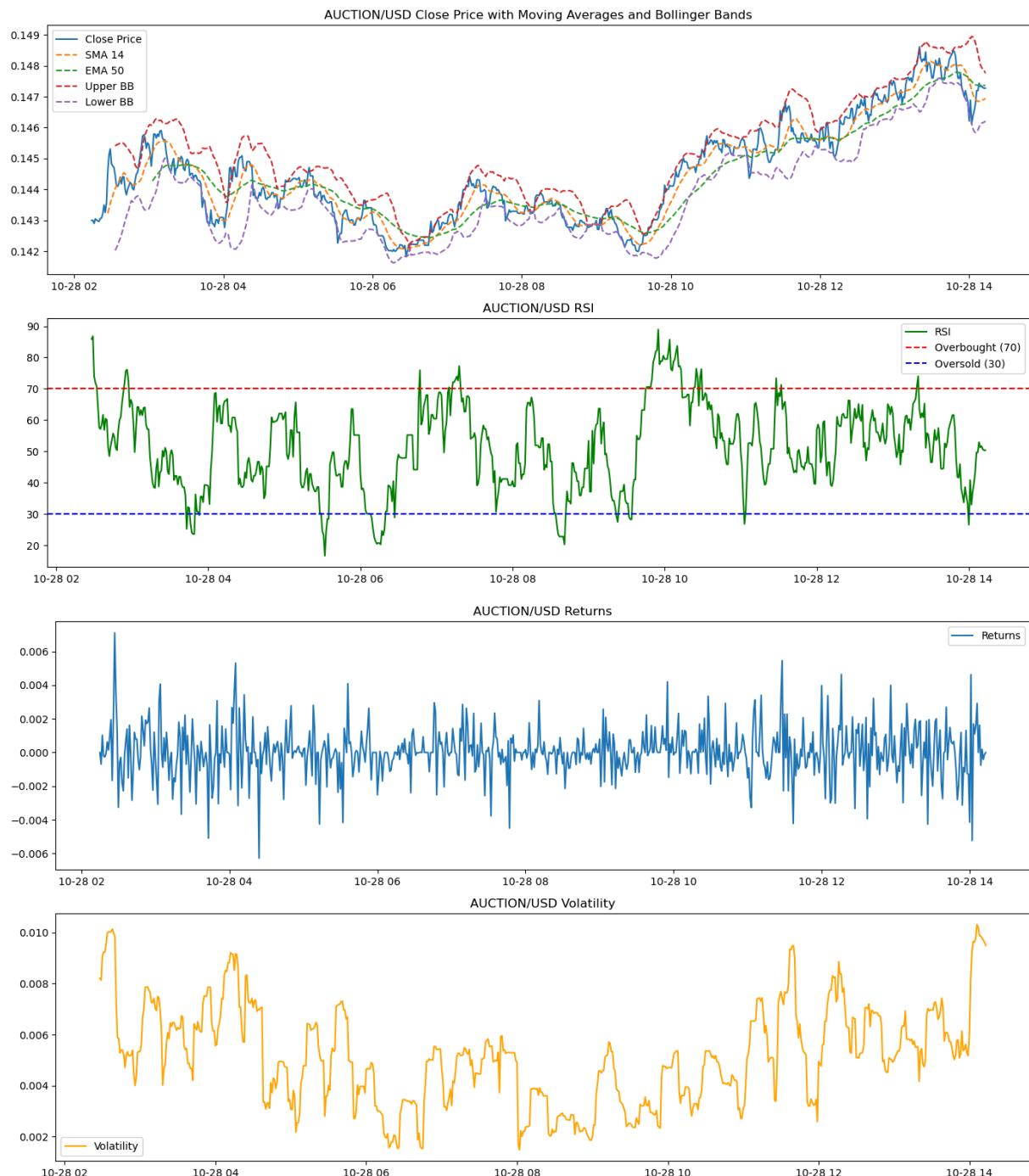
Data successfully saved to `ape_usd.csv`.



Data successfully saved to auction_usd in SQL Server.

SQL connection closed.

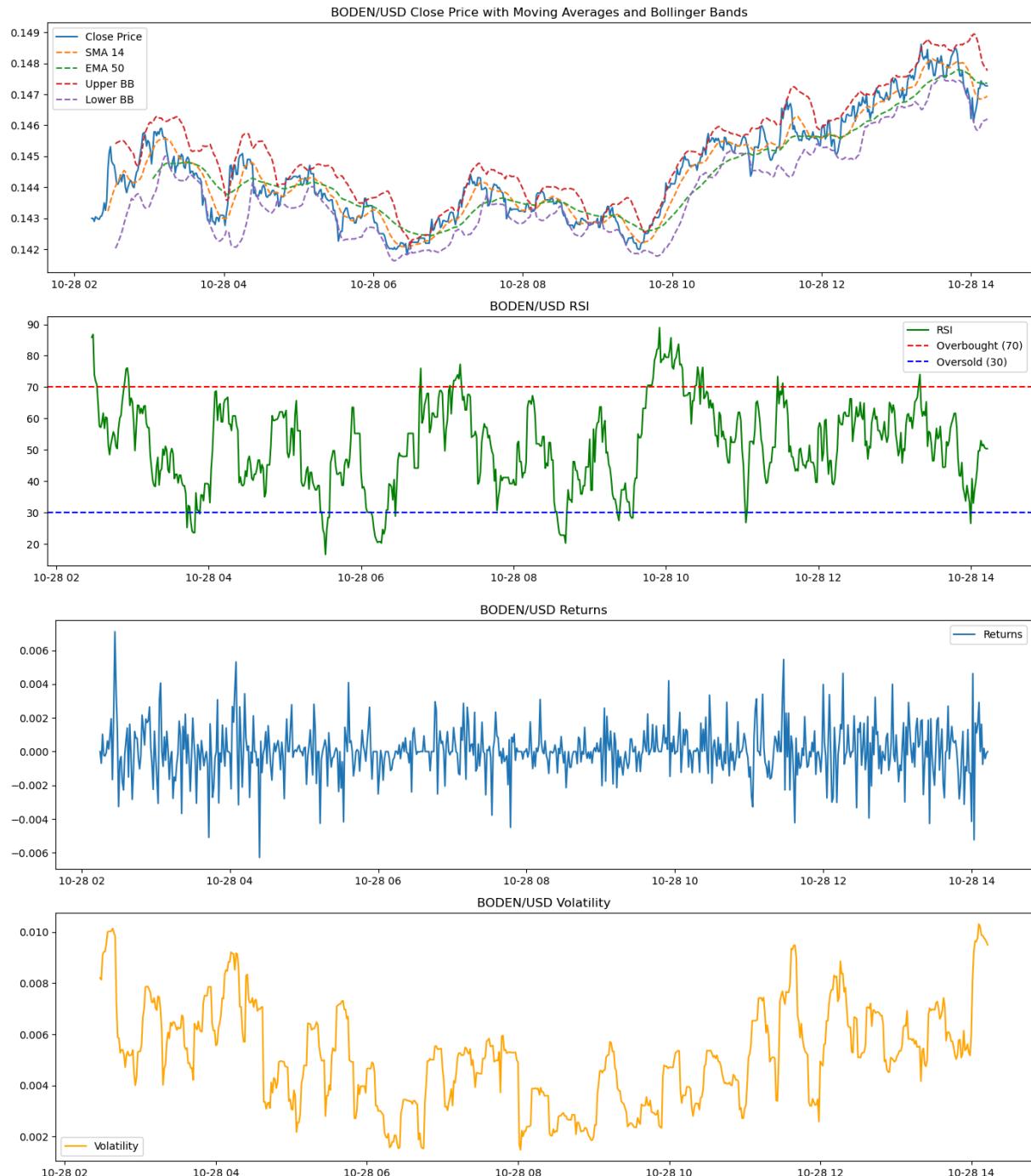
Data successfully saved to auction_usd.csv.



Data successfully saved to boden_usd in SQL Server.

SQL connection closed.

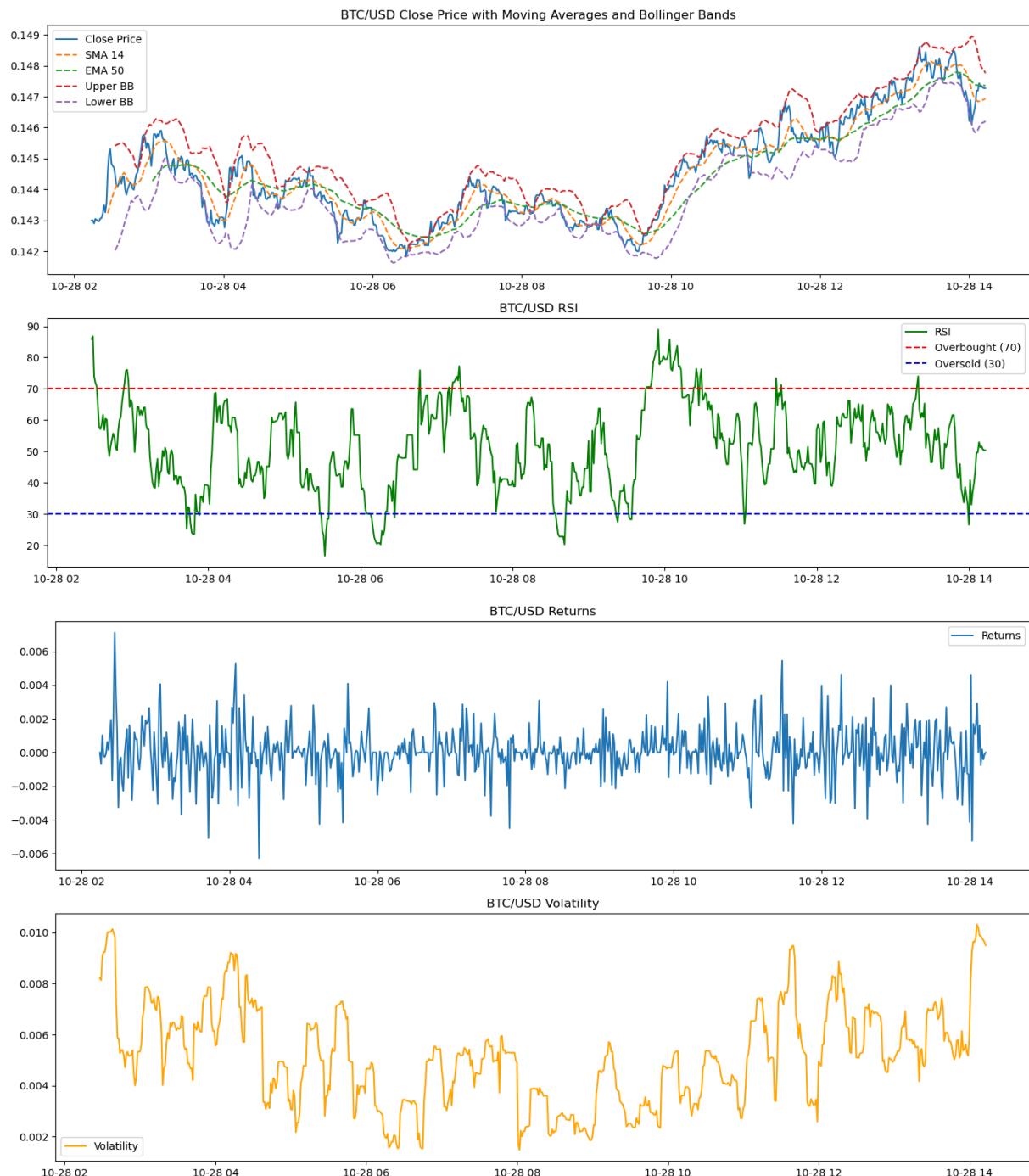
Data successfully saved to boden_usd.csv.



Data successfully saved to btc_usd in SQL Server.

SQL connection closed.

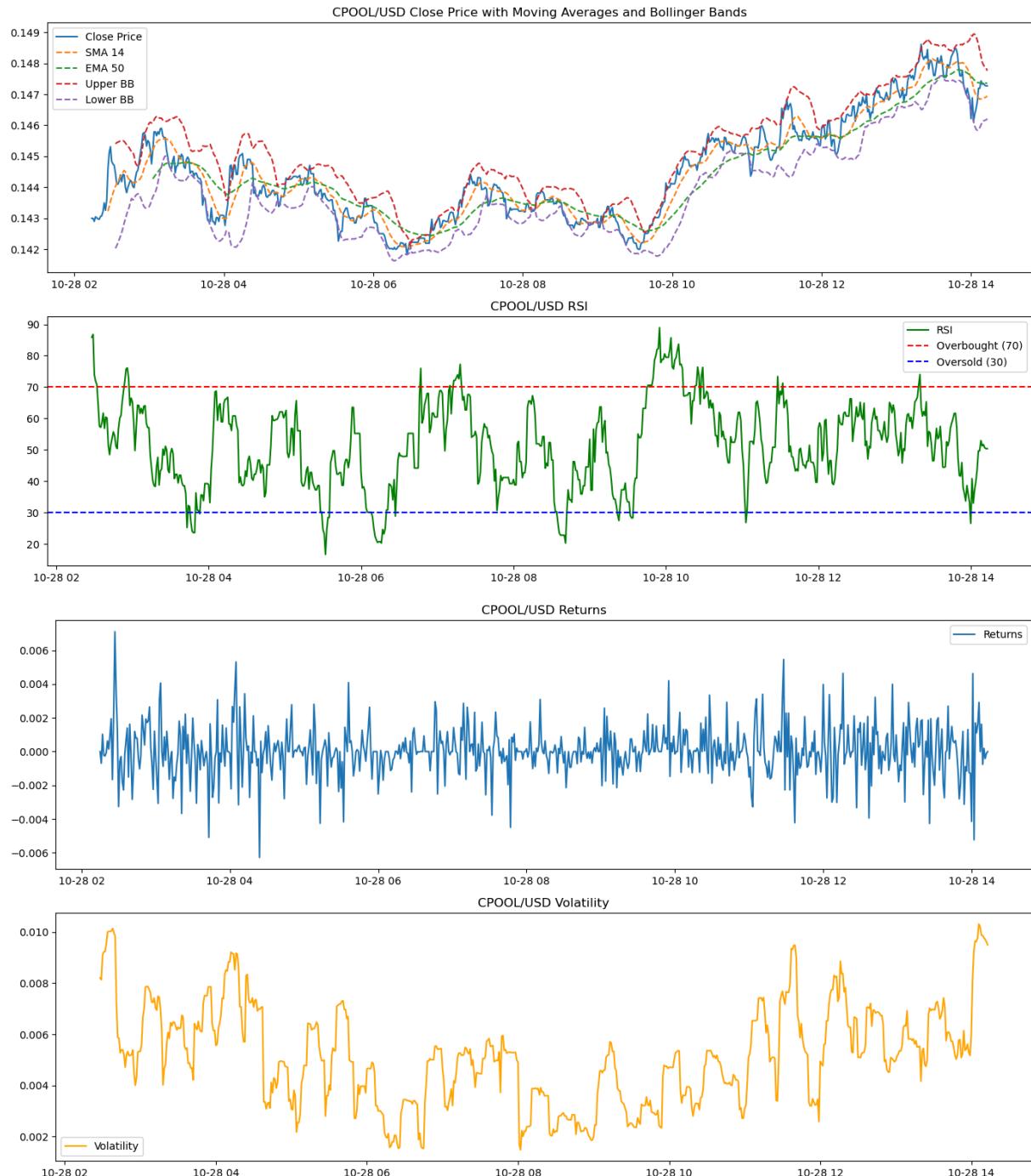
Data successfully saved to btc_usd.csv.



Data successfully saved to cpool_usd in SQL Server.

SQL connection closed.

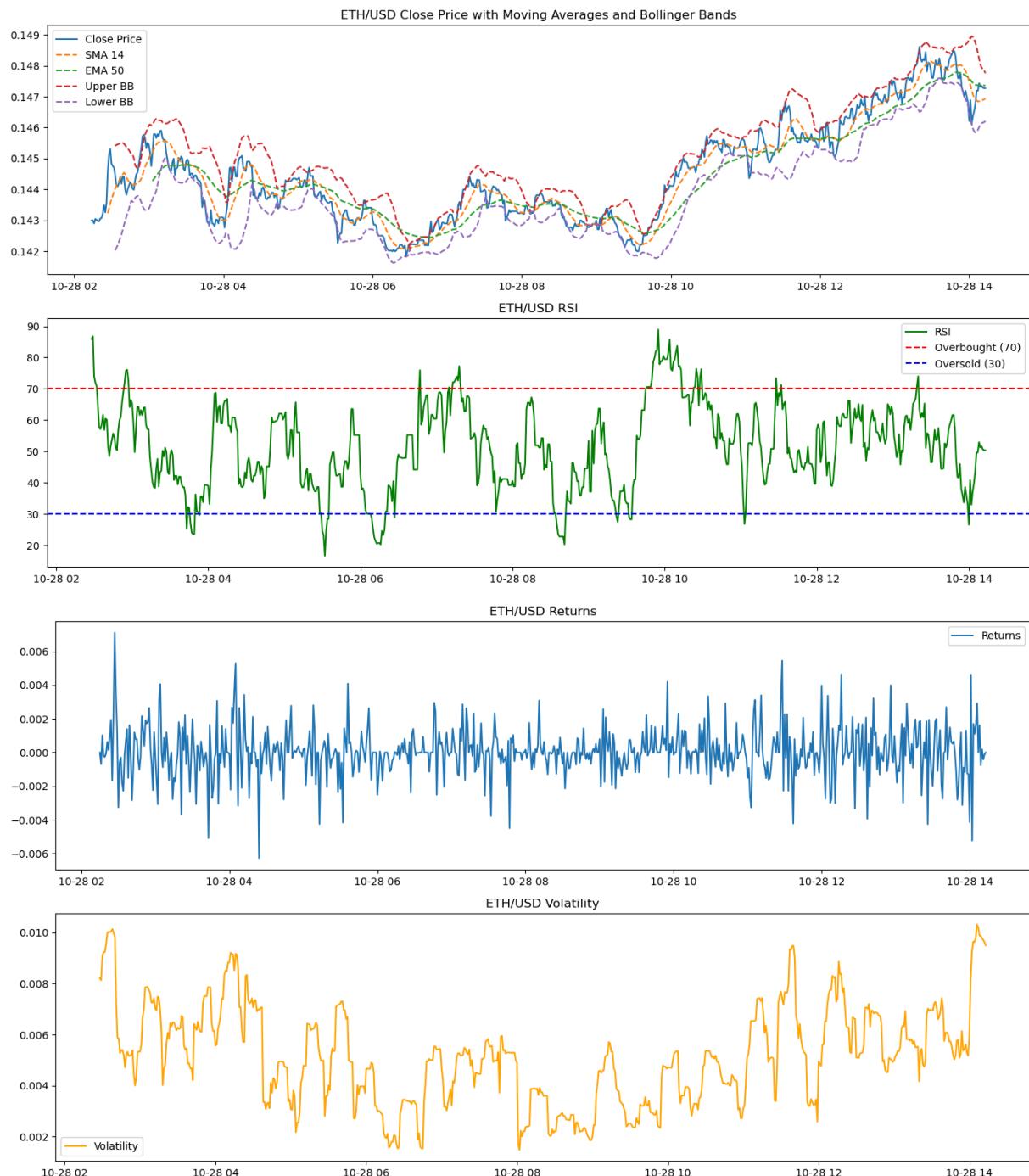
Data successfully saved to cpool_usd.csv.



Data successfully saved to eth_usd in SQL Server.

SQL connection closed.

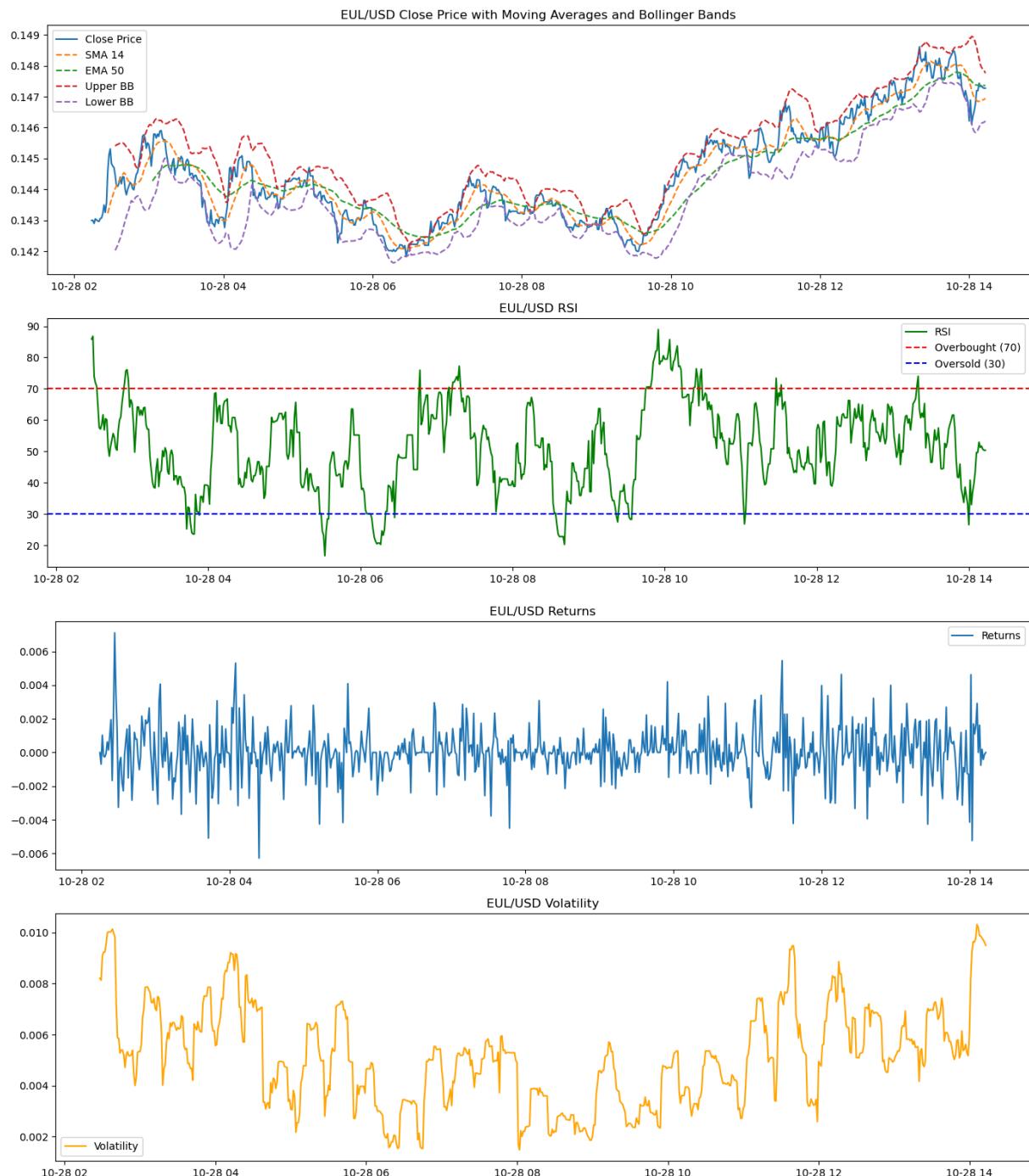
Data successfully saved to eth_usd.csv.



Data successfully saved to eul_usd in SQL Server.

SQL connection closed.

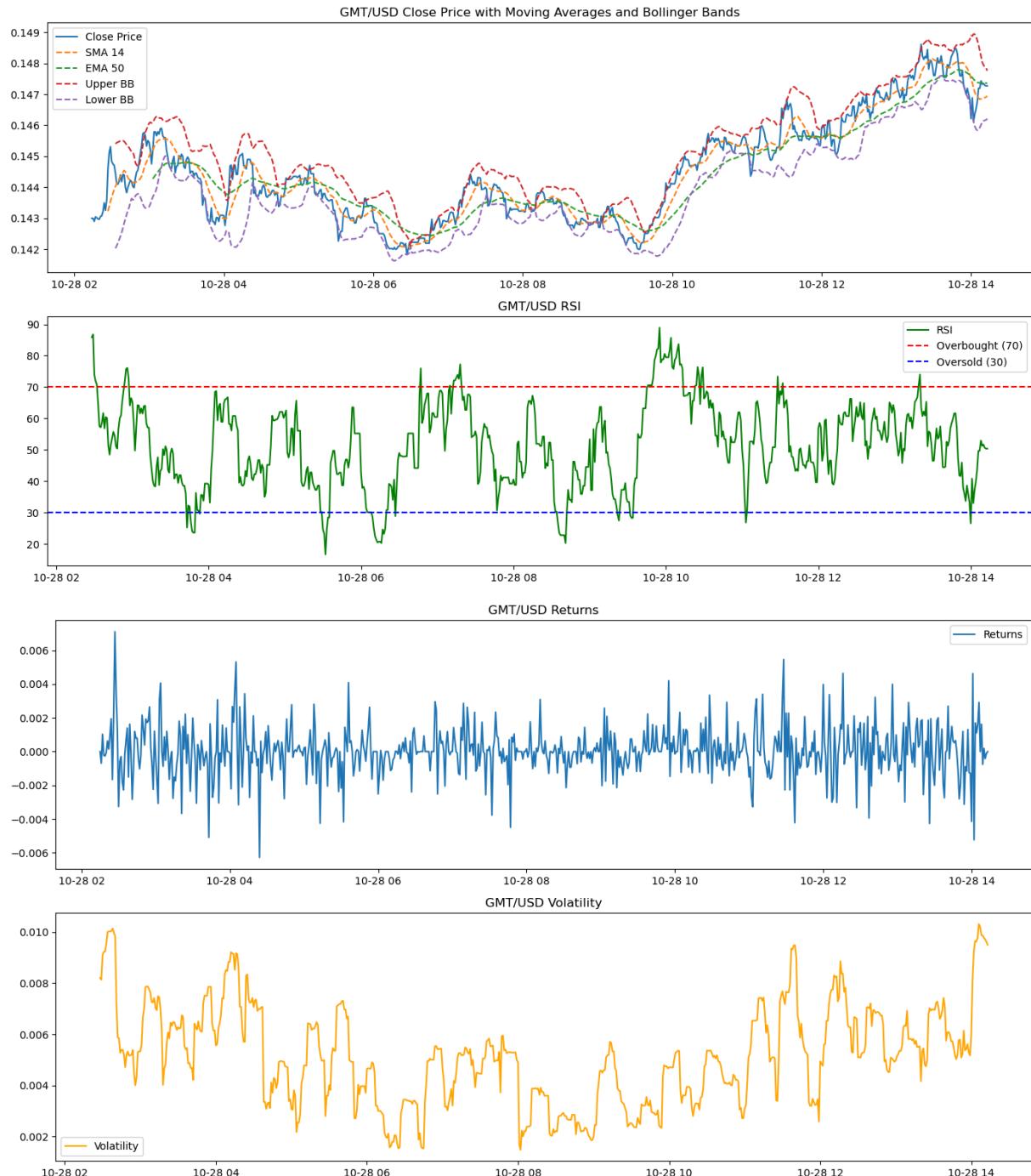
Data successfully saved to eul_usd.csv.



Data successfully saved to gmt_usd in SQL Server.

SQL connection closed.

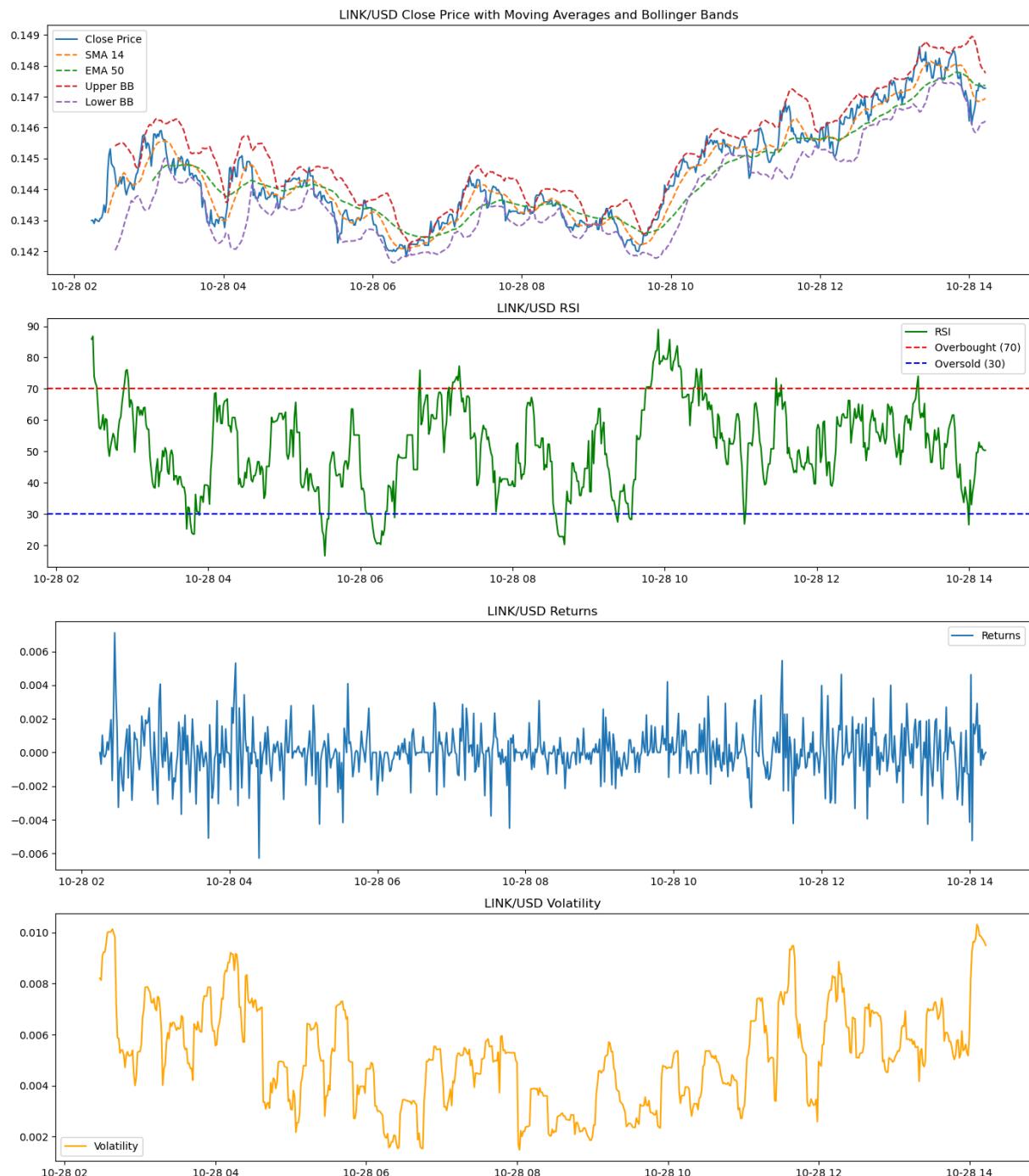
Data successfully saved to gmt_usd.csv.



Data successfully saved to [link_usd](#) in SQL Server.

SQL connection closed.

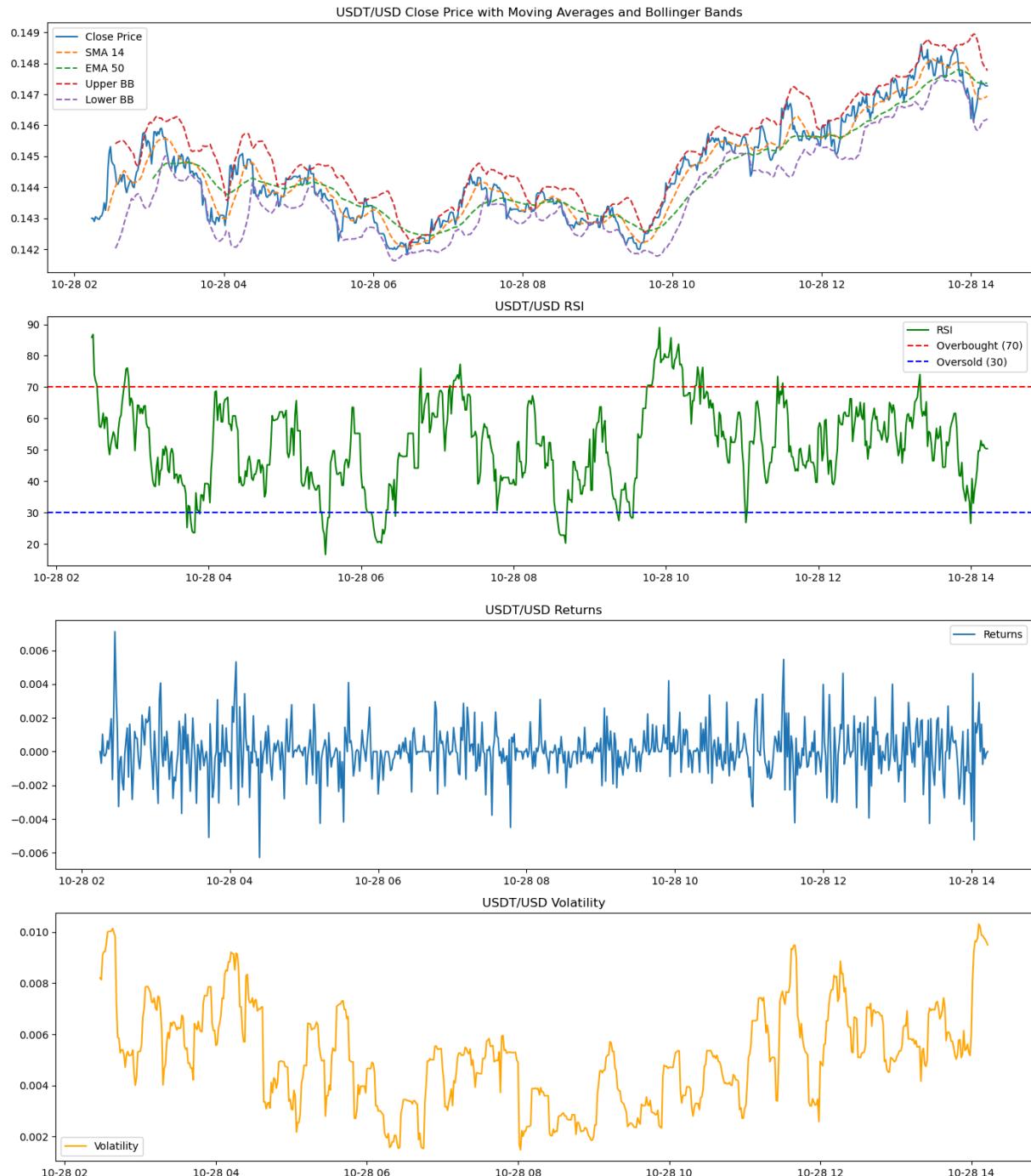
Data successfully saved to [link_usd.csv](#).



Data successfully saved to usdt_usd in SQL Server.

SQL connection closed.

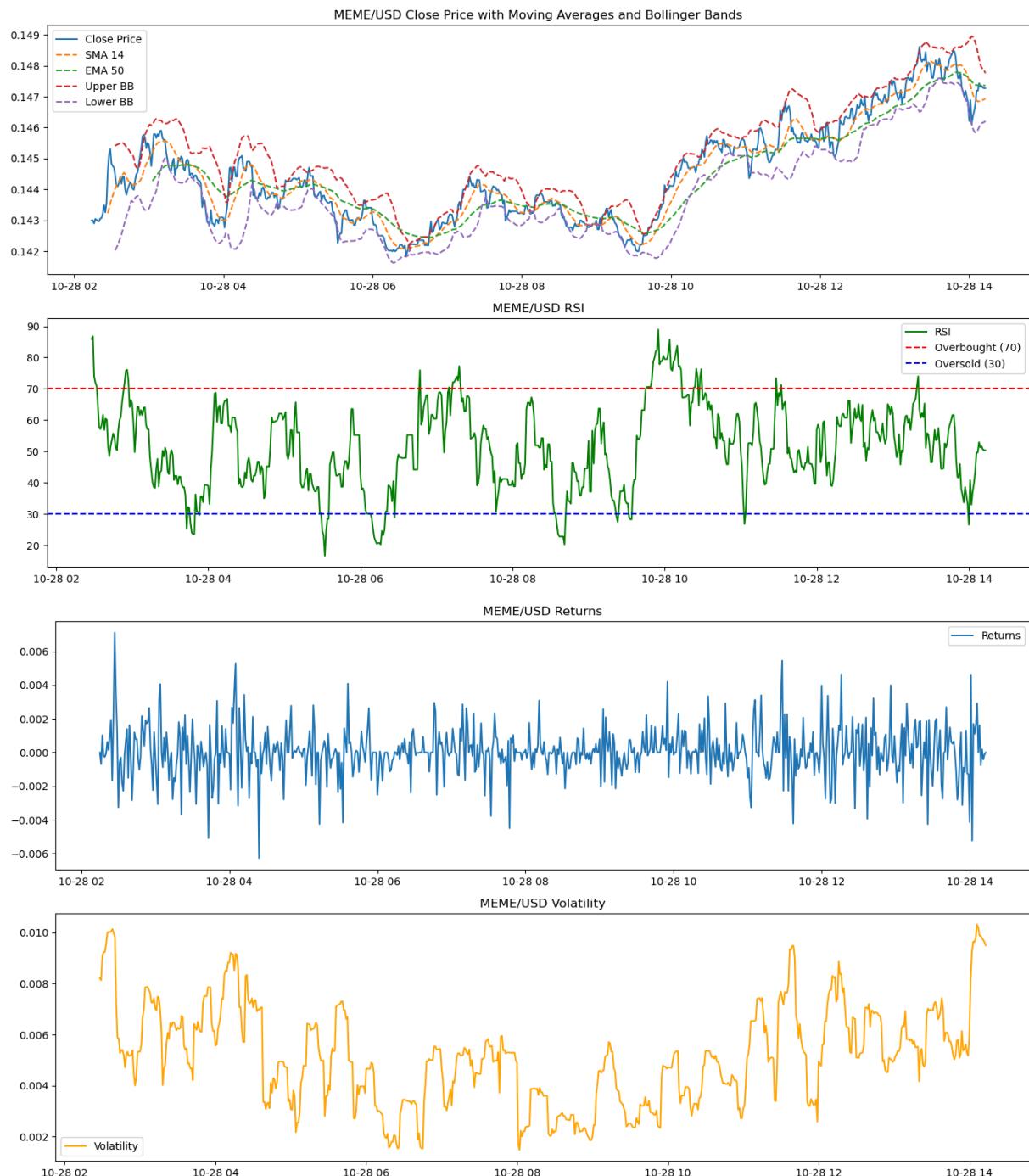
Data successfully saved to usdt_usd.csv.



Data successfully saved to `meme_usd` in SQL Server.

SQL connection closed.

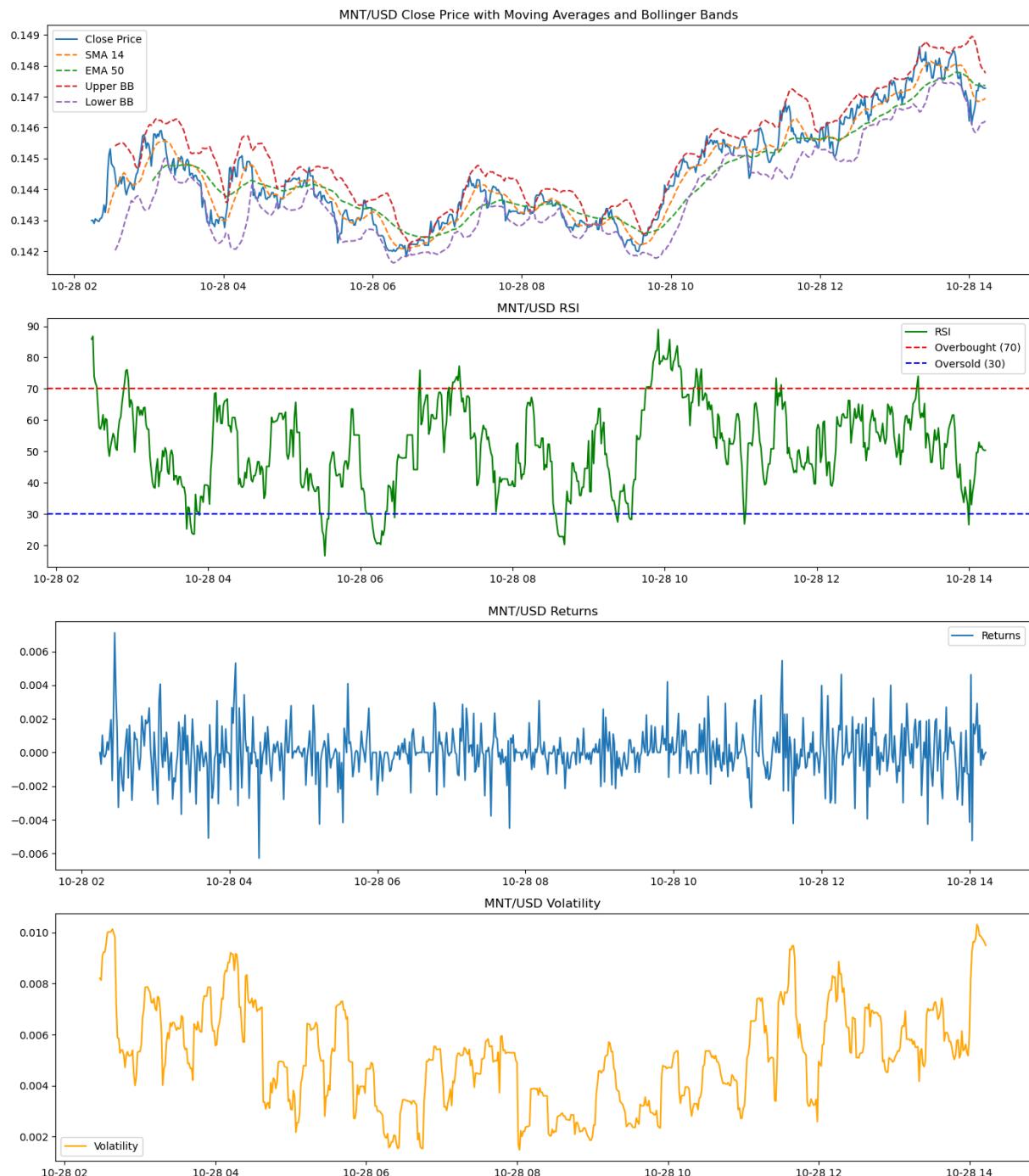
Data successfully saved to `meme_usd.csv`.



Data successfully saved to mnt_usd in SQL Server.

SQL connection closed.

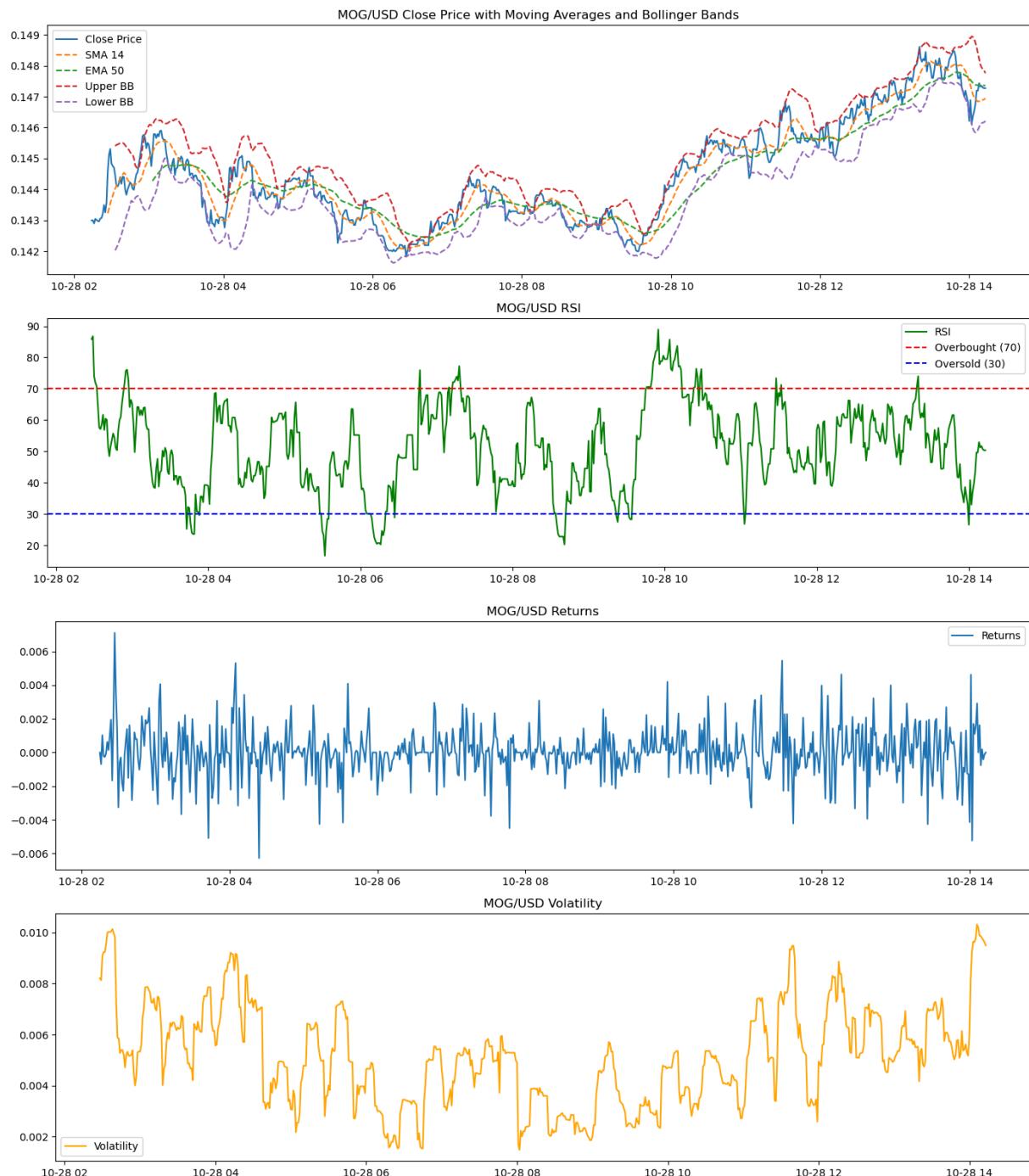
Data successfully saved to mnt_usd.csv.



Data successfully saved to mog_usd in SQL Server.

SQL connection closed.

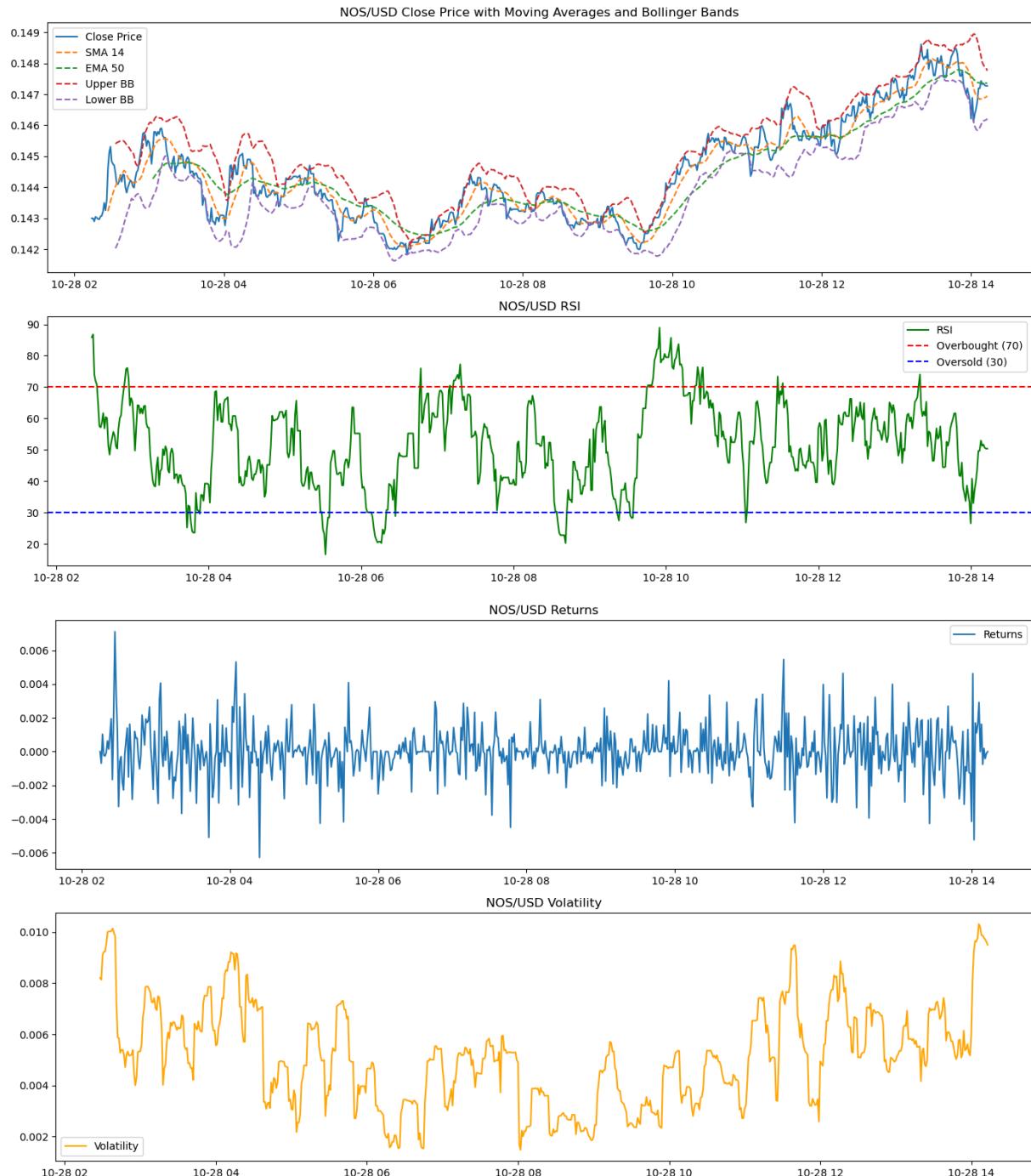
Data successfully saved to mog_usd.csv.



Data successfully saved to nos_usd in SQL Server.

SQL connection closed.

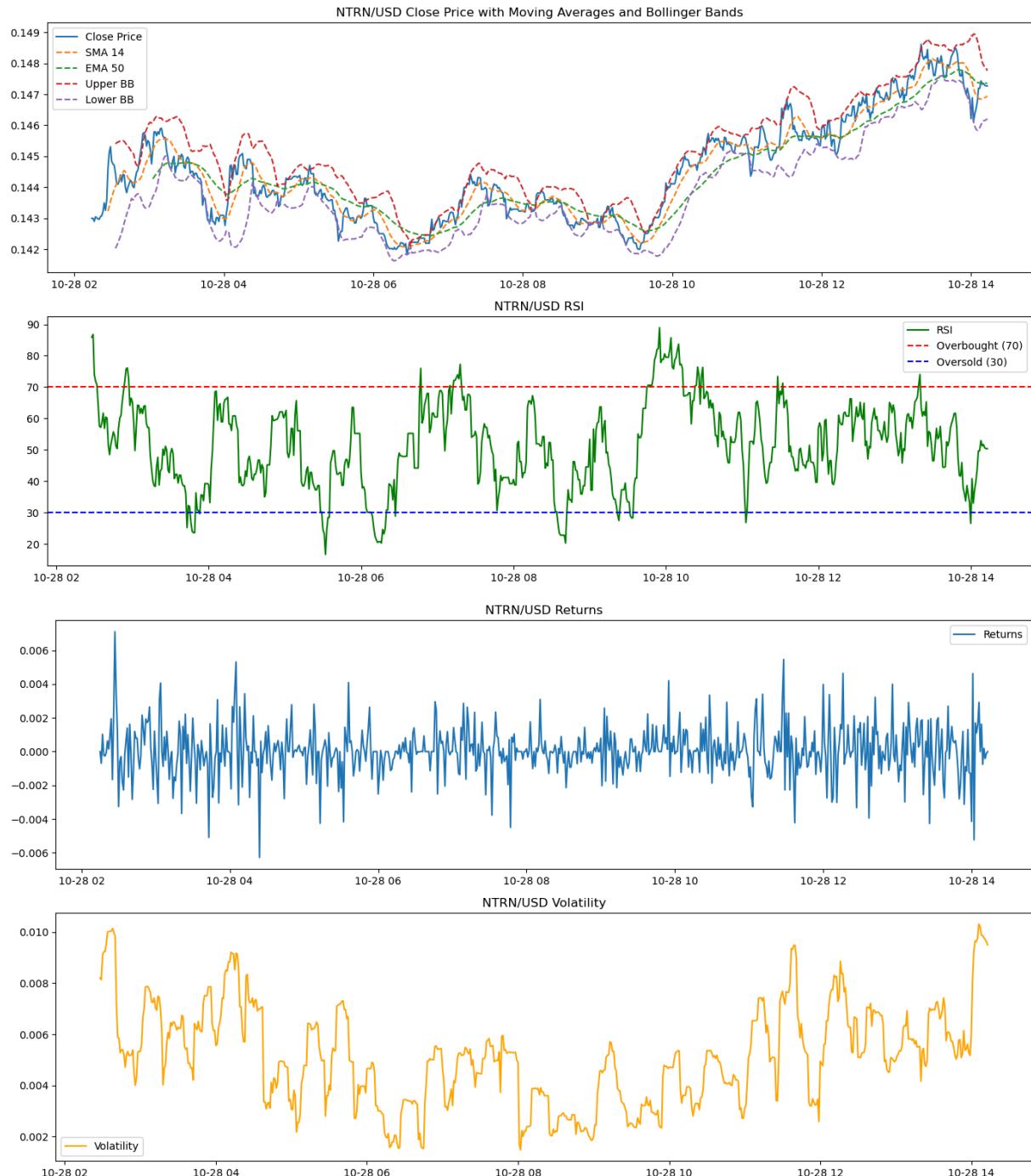
Data successfully saved to nos_usd.csv.



Data successfully saved to ntrn_usd in SQL Server.

SQL connection closed.

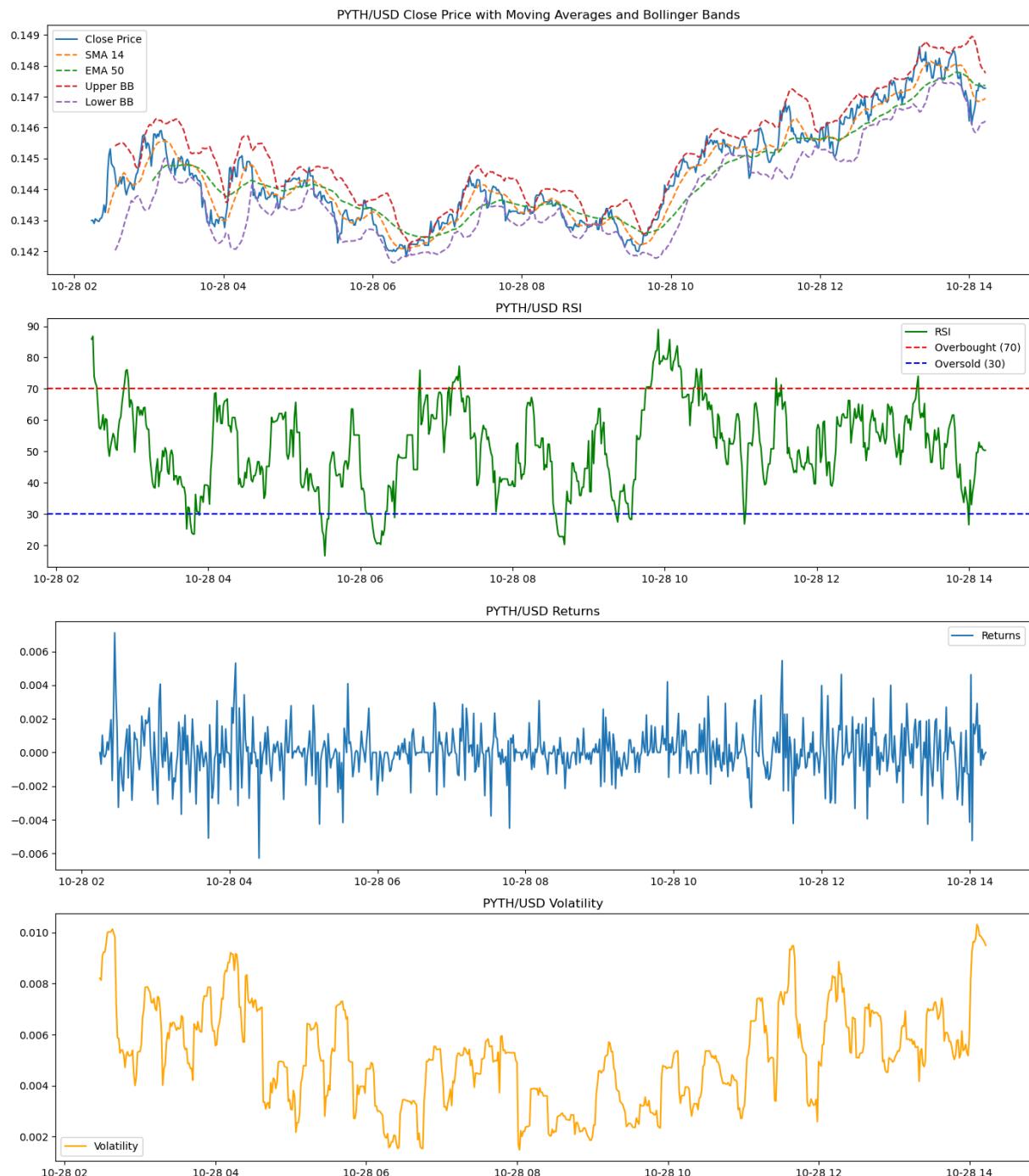
Data successfully saved to ntrn_usd.csv.



Data successfully saved to pyth_usd in SQL Server.

SQL connection closed.

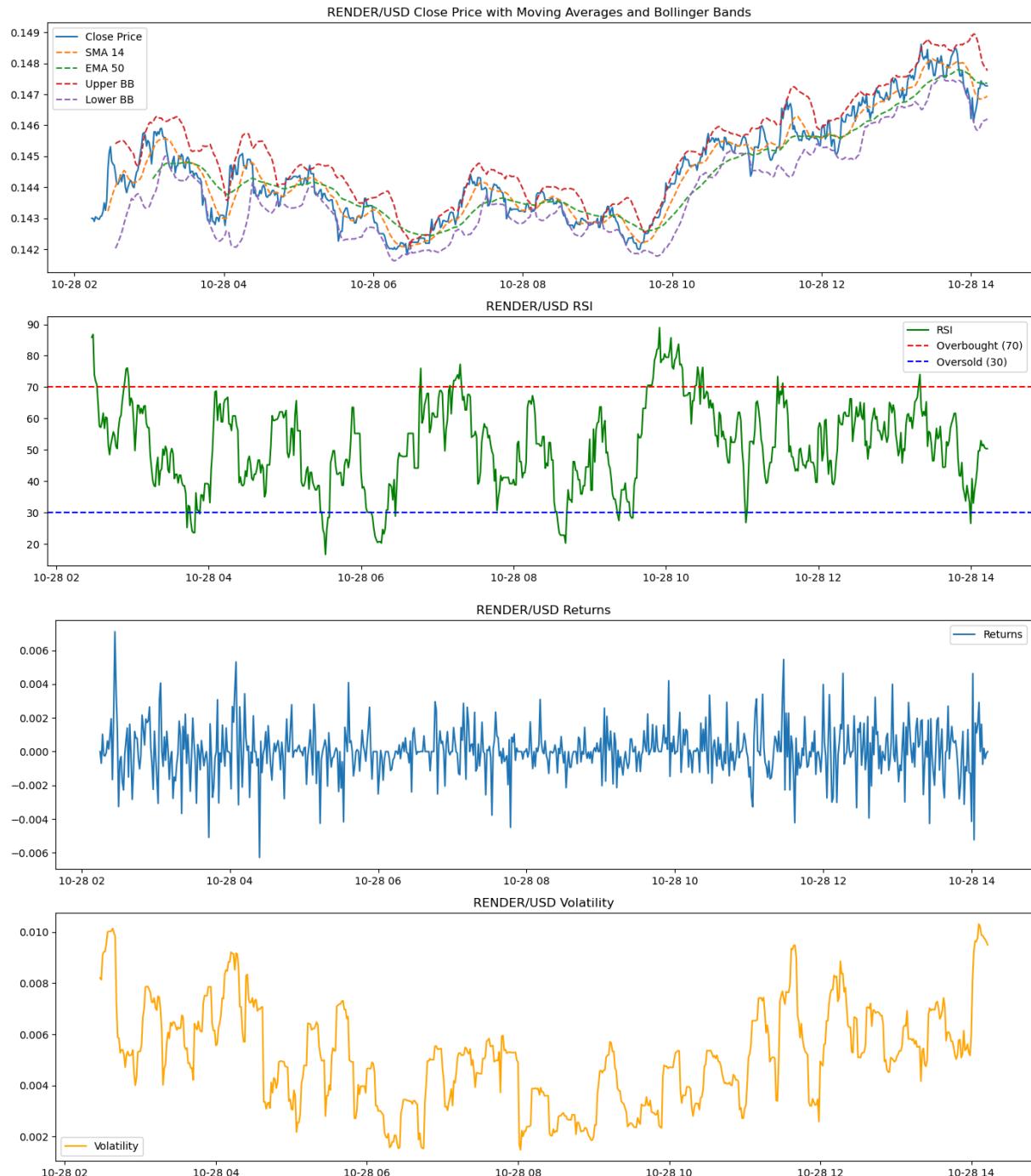
Data successfully saved to pyth_usd.csv.



Data successfully saved to render_usd in SQL Server.

SQL connection closed.

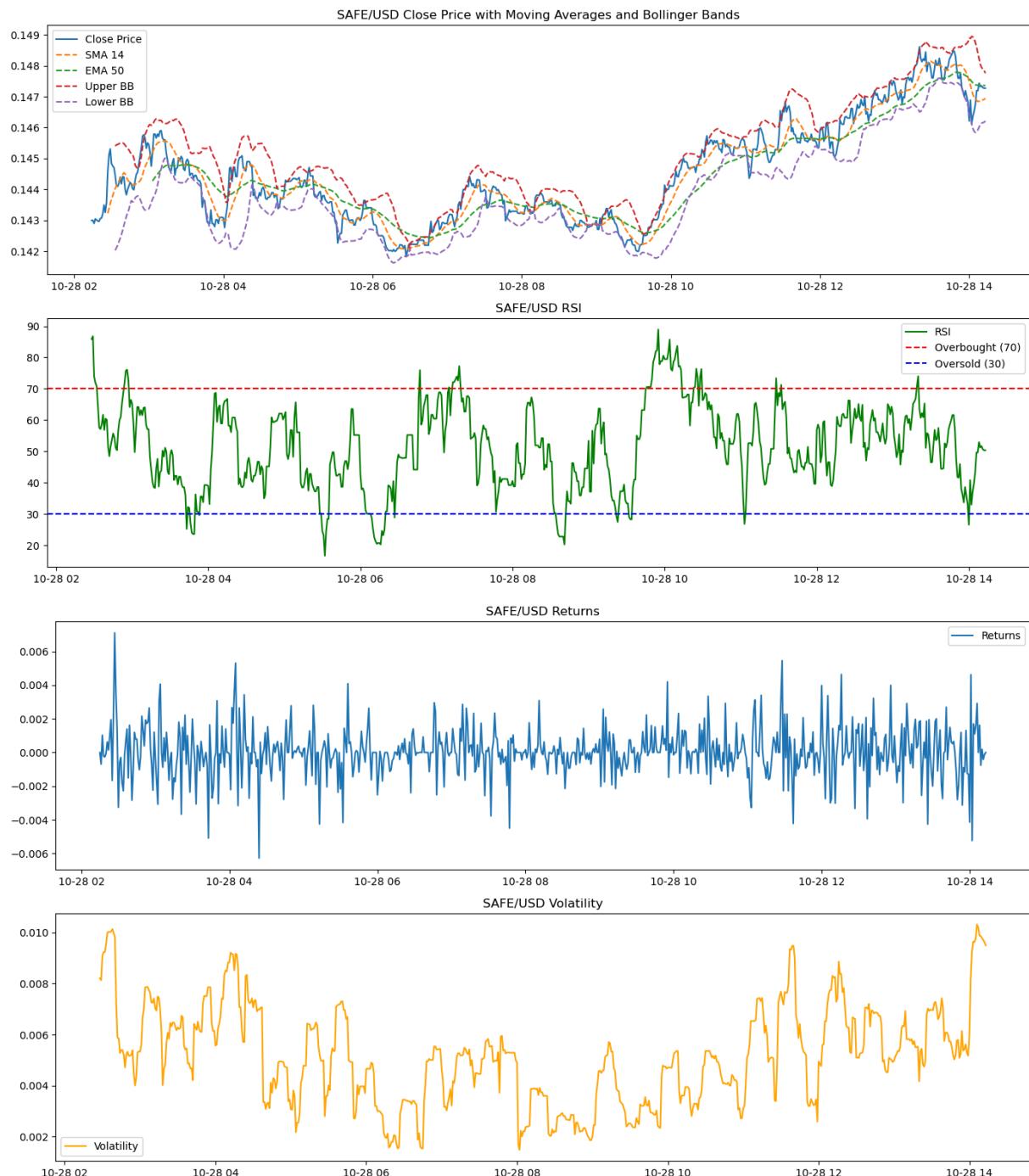
Data successfully saved to render_usd.csv.



Data successfully saved to `safe_usd` in SQL Server.

SQL connection closed.

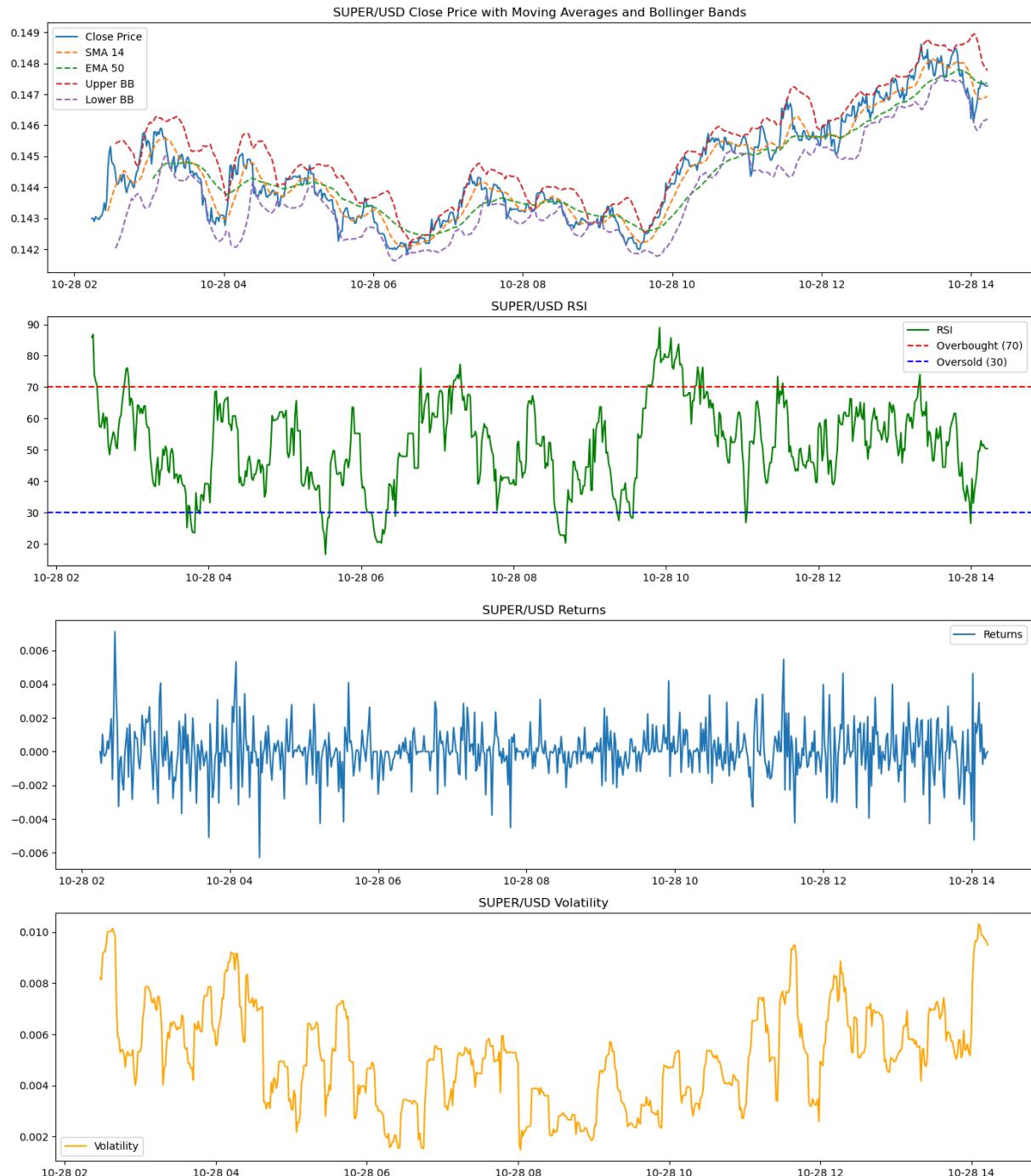
Data successfully saved to `safe_usd.csv`.



Data successfully saved to super_usd in SQL Server.

SQL connection closed.

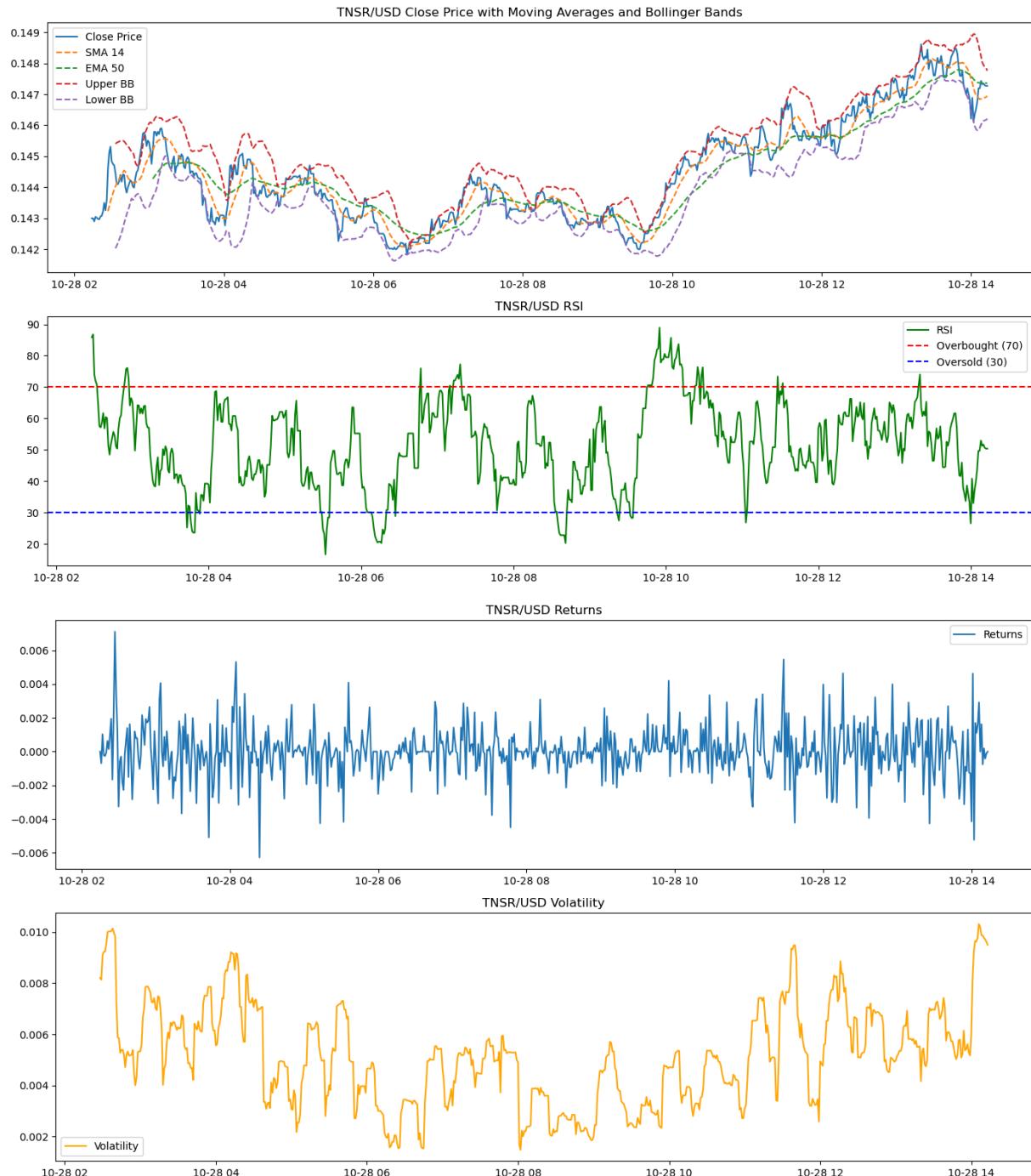
Data successfully saved to super_usd.csv.



Data successfully saved to tnsr_usd in SQL Server.

SQL connection closed.

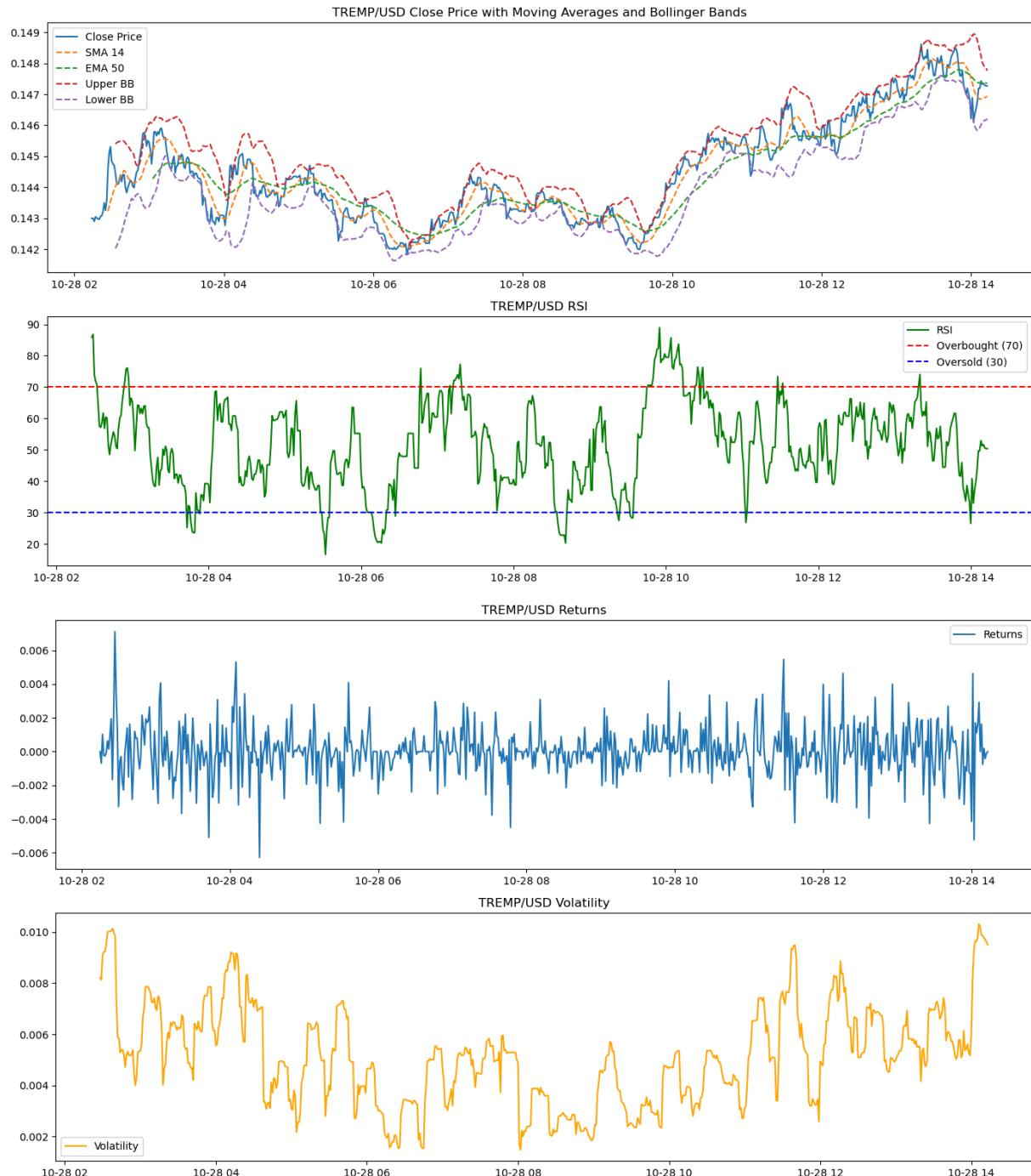
Data successfully saved to tnsr_usd.csv.



Data successfully saved to tremp_usd in SQL Server.

SQL connection closed.

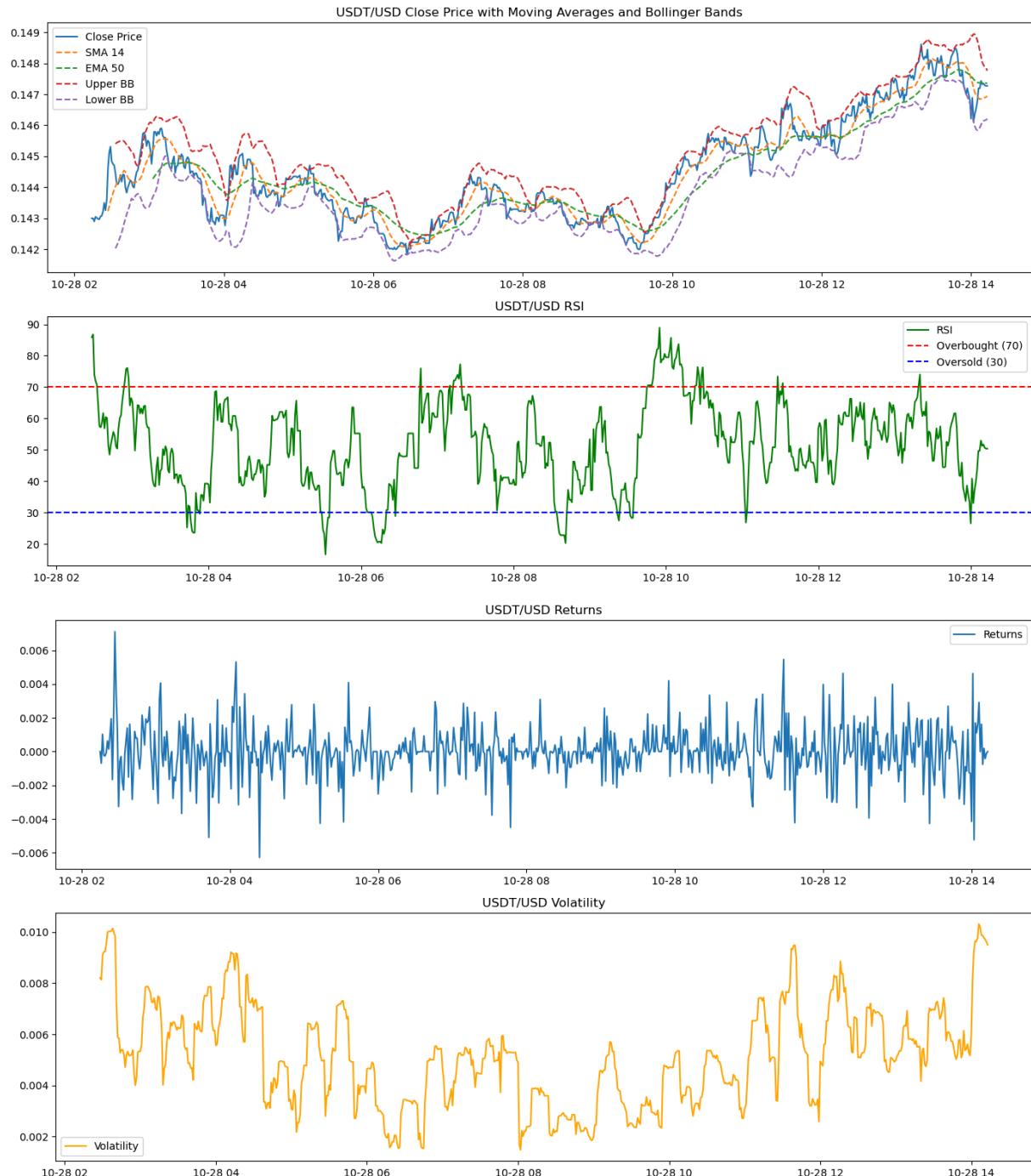
Data successfully saved to tremp_usd.csv.



Data successfully saved to usdt_usd in SQL Server.

SQL connection closed.

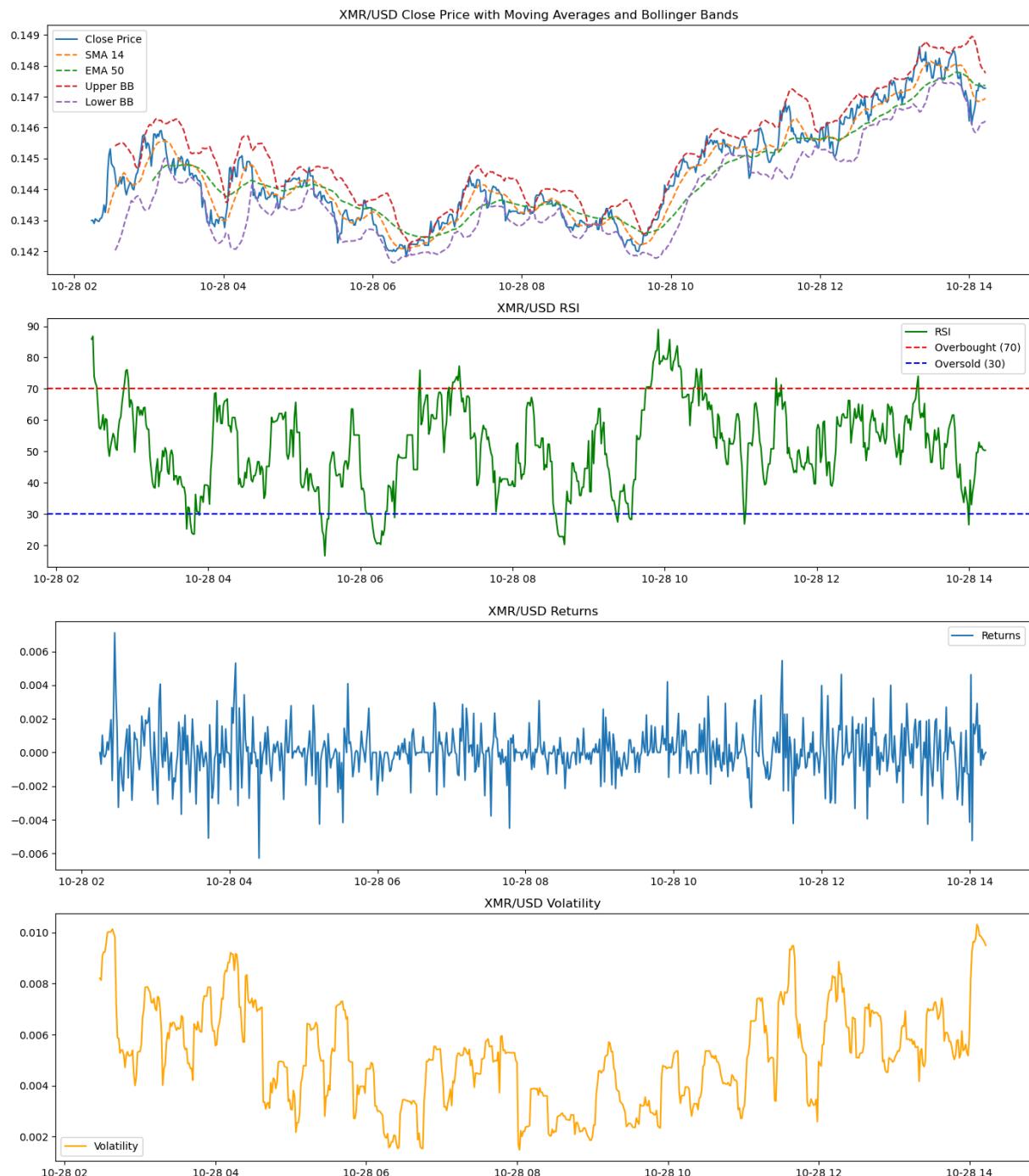
Data successfully saved to usdt_usd.csv.



Data successfully saved to xmr_usd in SQL Server.

SQL connection closed.

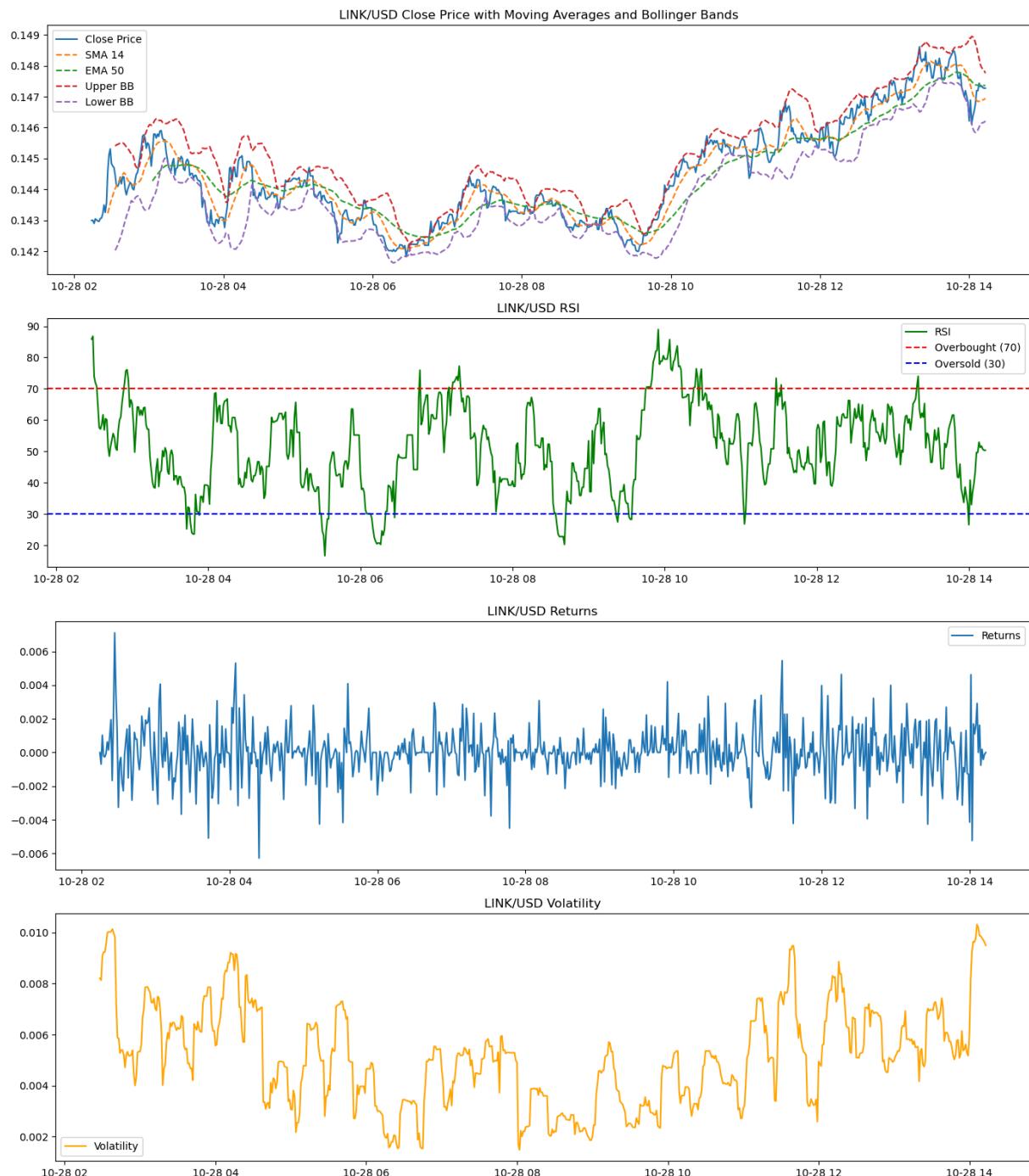
Data successfully saved to xmr_usd.csv.



Data successfully saved to [link_usd](#) in SQL Server.

SQL connection closed.

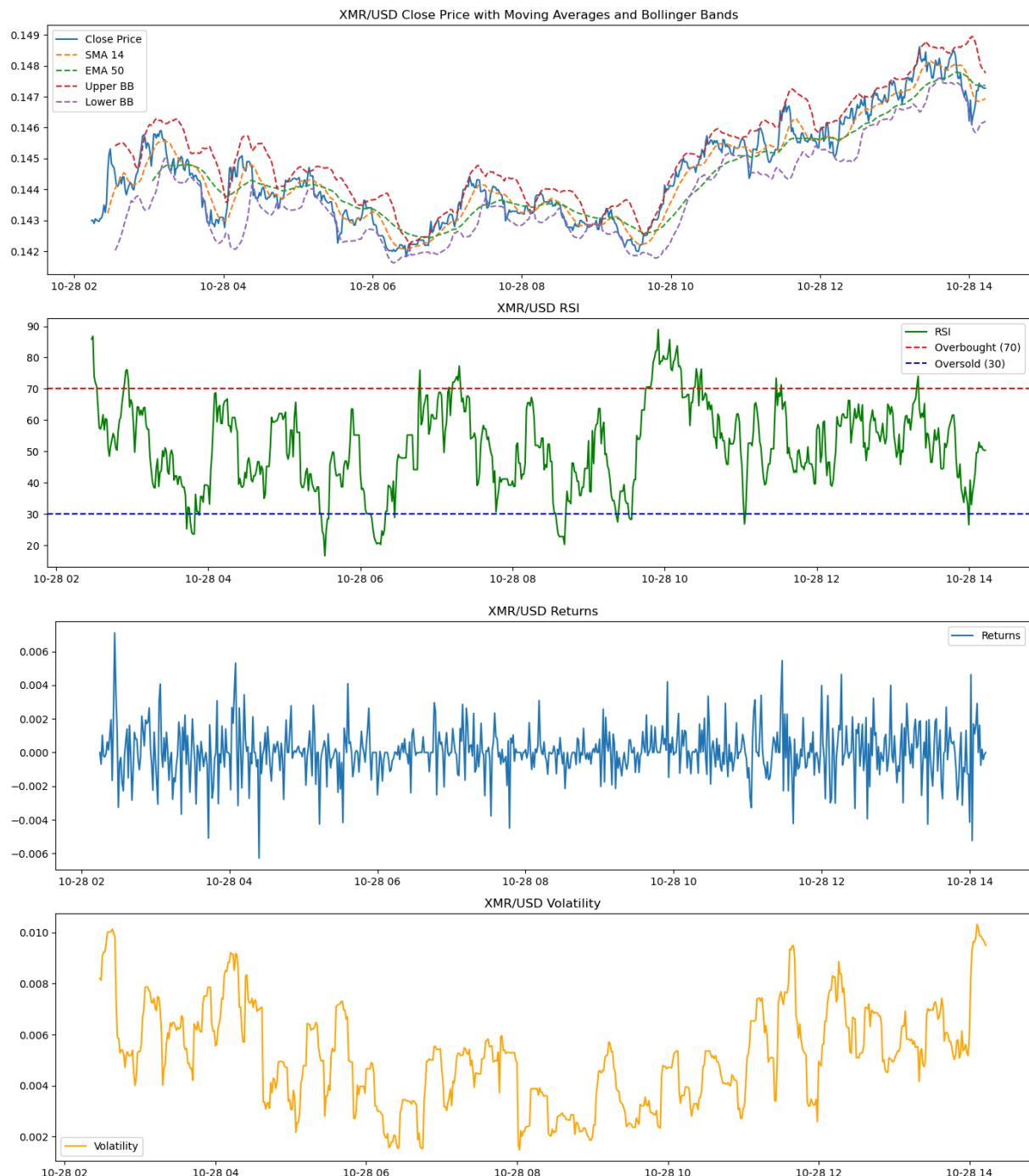
Data successfully saved to [link_usd.csv](#).



Data successfully saved to xmr_usd in SQL Server.

SQL connection closed.

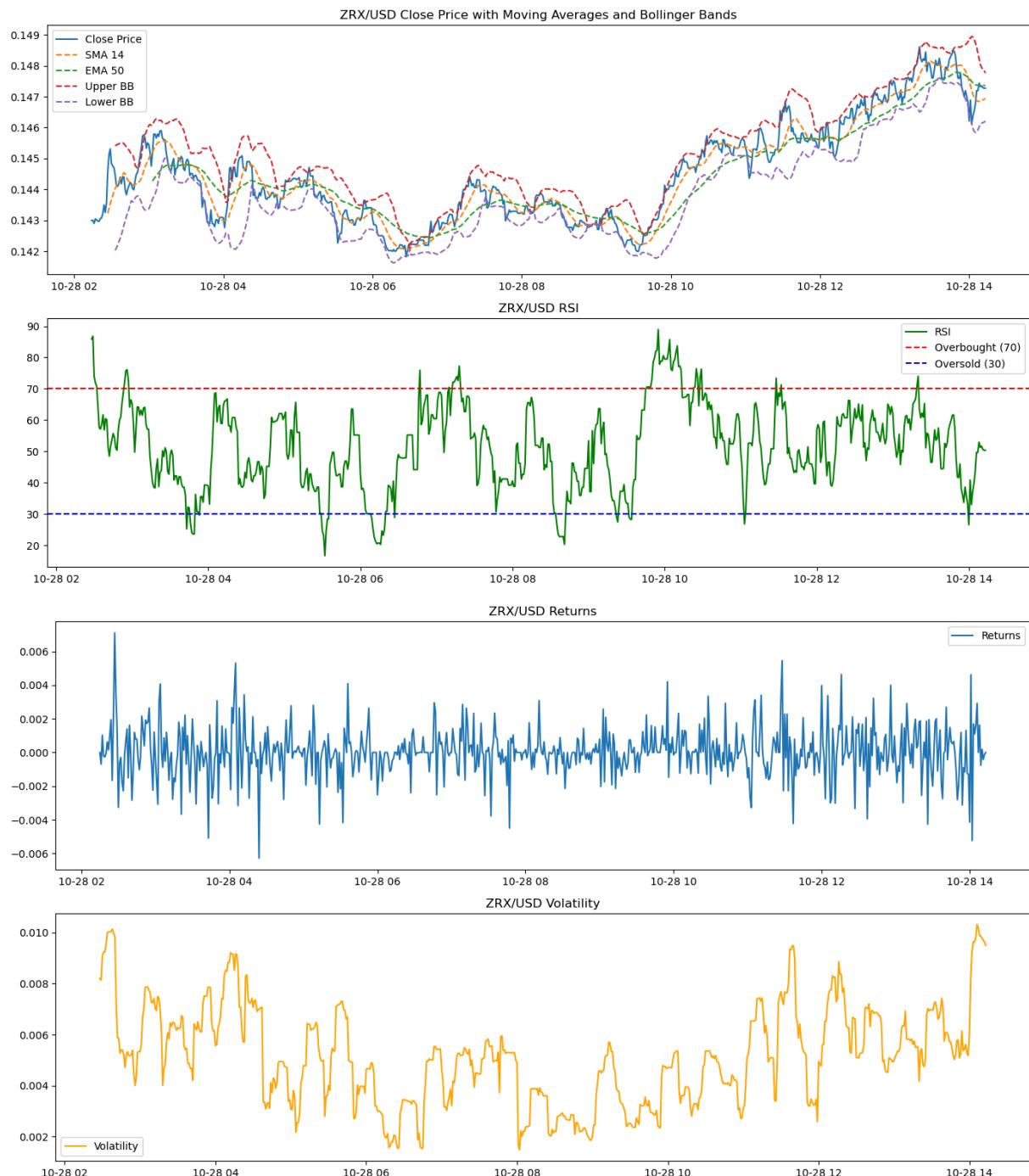
Data successfully saved to xmr_usd.csv.



Data successfully saved to zrx_usd in SQL Server.

SQL connection closed.

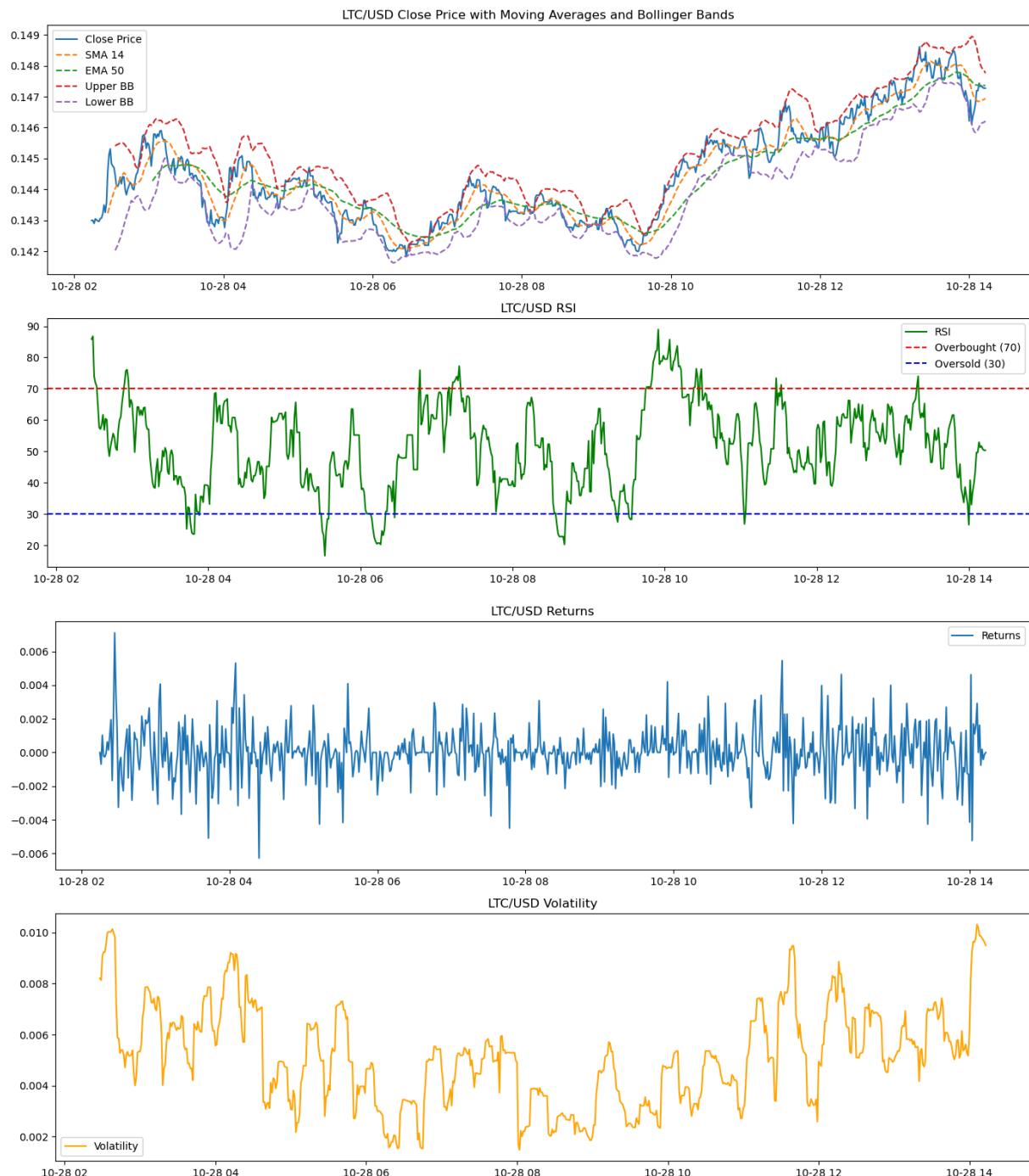
Data successfully saved to zrx_usd.csv.



Data successfully saved to ltc_usd in SQL Server.

SQL connection closed.

Data successfully saved to ltc_usd.csv.



Data successfully saved to doge_usd in SQL Server.

SQL connection closed.

Data successfully saved to doge_usd.csv.

