In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import requests
import datetime
from google.colab import data_table
from IPython.core.display import display, HTML
import yfinance as yf
```

In [2]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [3]:
```
# link to the monte carlo data folder https://drive.google.com/drive/folders/10GU2YP8ijheI8hR6IYnzU2F7A8b1I

#julia filepath
#file_path = '/content/drive/My Drive/MSDS 460/Tennessee Redistricting/data/'

# paul filepath
file_path = '/content/drive/My Drive/shared_folders/MSDS 460/Monte Carlo/monte_carlo_data/each_ticker_as_a_

#graham filepath
#file_path = "/content/drive/My Drive/"

# sue filepath
#file_path = '/content/drive/My Drive/MSDS 460/MSDS 460/Monte Carlo/monte_carlo_data/each_ticker_as_a_sepa
```

In [4]:
```
##########################################################################################################
##########################################################################################################
##########################################################################################################
##########################################################################################################
##########################################################################################################

#------------------------------------          Part 1: Calculate the buy and hold returns for the 10 stocks

##########################################################################################################
##########################################################################################################
##########################################################################################################
```

```
##############################################################################################
##############################################################################################
```

In [4]:

```

```

In [5]:

```
# this code takes in the seperate price history for all ten tickers since the example from https://github.c
# I assume we have $100,000 to start and invest 10,000 in each ticker
```

In [6]:

```
# import data. each ticker's df seperately

tickers = ["XOM", "CVX", "COP", "EOG", "EPD", "WMB", "OKE", "LNG", "OXY", "HES"]

stock_dfs = {}
for ticker in tickers:
    csv_path = f"{file_path}{ticker}_data.csv"
    print(csv_path)
    stock_dfs[ticker] = pd.read_csv(csv_path)
```

```
/content/drive/My Drive/shared_folders/MSDS 460/Monte Carlo/monte_carlo_data/each_ticker_as_a_separate_csv/
XOM_data.csv
/content/drive/My Drive/shared_folders/MSDS 460/Monte Carlo/monte_carlo_data/each_ticker_as_a_separate_csv/
CVX_data.csv
/content/drive/My Drive/shared_folders/MSDS 460/Monte Carlo/monte_carlo_data/each_ticker_as_a_separate_csv/
COP_data.csv
/content/drive/My Drive/shared_folders/MSDS 460/Monte Carlo/monte_carlo_data/each_ticker_as_a_separate_csv/
EOG_data.csv
/content/drive/My Drive/shared_folders/MSDS 460/Monte Carlo/monte_carlo_data/each_ticker_as_a_separate_csv/
EPD_data.csv
/content/drive/My Drive/shared_folders/MSDS 460/Monte Carlo/monte_carlo_data/each_ticker_as_a_separate_csv/
WMB_data.csv
/content/drive/My Drive/shared_folders/MSDS 460/Monte Carlo/monte_carlo_data/each_ticker_as_a_separate_csv/
OKE_data.csv
/content/drive/My Drive/shared_folders/MSDS 460/Monte Carlo/monte_carlo_data/each_ticker_as_a_separate_csv/
LNG_data.csv
/content/drive/My Drive/shared_folders/MSDS 460/Monte Carlo/monte_carlo_data/each_ticker_as_a_separate_csv/
OXY_data.csv
/content/drive/My Drive/shared_folders/MSDS 460/Monte Carlo/monte_carlo_data/each_ticker_as_a_separate_csv/
HES_data.csv
```

In [7]:

```
print(stock_dfs['XOM'].head())
```

```
         date      close       high        low       open   volume
0  1999-01-04  16.089716  16.422034  16.006637  16.117409  8853600
1  1999-01-05  15.951257  16.158955  15.882024  16.062029  6652800
2  1999-01-06  16.588202  16.795900  16.103571  16.200497  9965600
3  1999-01-07  16.560503  16.602043  16.338958  16.491270  7417200
4  1999-01-08  16.463579  16.546659  16.158955  16.449733  6343400
```

In [8]:
```
print(stock_dfs['CVX'].tail())
```

```
            date       close        high         low        open   volume
6284  2023-12-22  143.222809  144.493368  142.938352  143.877056  6394600
6285  2023-12-26  144.512314  145.081214  144.028732  144.189936  5165600
6286  2023-12-27  144.038223  145.043293  143.497753  144.379569  5337200
6287  2023-12-28  142.009109  144.142517  141.658273  143.346034  8148000
6288  2023-12-29  141.430710  142.445256  140.966096  142.255623  7653800
```

In [8]:
```

```

In [9]:
```
# calculating buy and hold return - buy on the first date (open price) and sell on the last row close price

stock_dfs["XOM"]['open'].iloc[0]
```

Out[9]: 16.11740910042584

In [10]:
```
stock_dfs["XOM"]['close'].iloc[-1]
```

Out[10]: 95.82491302490234

In [11]:
```
# https://www.investopedia.com/articles/basics/10/guide-to-calculating-roi.asp#:~:text=Return%20on%20inves

# return = (sell price - buy price) / buy price
# e.g. buy at $100, sell at $1000

# (1000 - 100) / 100 = 9

XOM_return = (95.82491302490234 - 16.11740910042584) / 16.11740910042584

print(XOM_return)
```

```
4.94542909644023
```

In [12]:
```
# 10,000 invested in XOM on 1/1/1999 would be worth 10,000 * 4.94542909644023 = $ 49,454
```

In [13]:
```
buy_and_hold_return_dict = {}

for ticker in tickers:
    df = stock_dfs[ticker]

    first_open = df['open'].iloc[0]
    last_close = df['close'].iloc[-1]

    buy_and_hold_return = ((last_close - first_open )/ first_open)
    buy_and_hold_return_dict[ticker] = buy_and_hold_return

print(buy_and_hold_return_dict)
```

```
{'XOM': 4.94542909644023, 'CVX': 7.908235098670692, 'COP': 14.887227633417485, 'EOG': 38.53016571767444, 'E
PD': 36.283392191978834, 'WMB': 2.7239286861671164, 'OKE': 27.19543392860096, 'LNG': 98.26574050609922, 'OX
Y': 14.034967078450647, 'HES': 11.154099747948093}
```

In [14]:
```
total_buy_and_hold_returns_energy_stocks = 0

for ticker, buy_hold_return in buy_and_hold_return_dict.items():
  final_value = 10000 * buy_hold_return
  total_buy_and_hold_returns_energy_stocks += final_value

print(f'{total_buy_and_hold_returns_energy_stocks:.2f}')
```

```
2559286.20
```

In [15]:
```
# return on initial 100k spread across all ten stocks

overall_energy_return = total_buy_and_hold_returns_energy_stocks / 100000
print(f'{overall_energy_return:.2f}')
```

```
25.59
```

In [15]:

In [16]:
```
spy_df = yf.download("SPY", start="1999-01-01", end = "2024-01-01", multi_level_index = False)

spy_df.head()
```

```
YF.download() has changed argument auto_adjust default to True
[*********************100%***********************]  1 of 1 completed
```

Out[16]:

Date	Close	High	Low	Open	Volume
1999-01-04	77.577934	78.957273	76.750331	77.794687	9450400
1999-01-05	78.464684	78.740551	77.518851	77.518851	8031000
1999-01-06	80.356354	80.553402	79.292292	79.331702	7737700
1999-01-07	79.962242	80.218405	79.311982	79.686374	5504900
1999-01-08	80.553360	81.026276	79.430184	80.829228	6224400

In [17]:

```
spy_df.tail()
```

Out[17]:

Date	Close	High	Low	Open	Volume
2023-12-22	467.651306	469.359407	465.726021	467.858638	67126600
2023-12-26	469.625946	470.544160	467.986966	468.065970	55387000
2023-12-27	470.475067	470.623161	468.875589	469.418611	68000300
2023-12-28	470.652771	471.501865	470.228224	470.840367	77158100
2023-12-29	469.290283	470.988501	467.305730	470.455331	122234100

In [18]:

```
spy_df['Open'][0]
```

```
<ipython-input-18-58fec3316e36>:1: FutureWarning: Series.__getitem__ treating keys as positions is deprecat
ed. In a future version, integer keys will always be treated as labels (consistent with DataFrame behavio
r). To access a value by position, use `ser.iloc[pos]`
  spy_df['Open'][0]
```

Out[18]: 77.79468744697988

In [19]:

```
spy_df['Close'][-1]
```

```
<ipython-input-19-ee0949ca8fe6>:1: FutureWarning: Series.__getitem__ treating keys as positions is deprecat
ed. In a future version, integer keys will always be treated as labels (consistent with DataFrame behavio
r). To access a value by position, use `ser.iloc[pos]`
  spy_df['Close'][-1]
```

Out[19]: 469.290283203125

In [20]:
```
# calculating market retuns using SPY index for the S&P 500

spy_open = spy_df['Open'].iloc[0]
spy_close = spy_df['Close'].iloc[-1]
```

In [21]:
```
sp500_return = (spy_close - spy_open) / spy_open
print(sp500_return)
```

```
5.03242070382974
```

In [22]:
```
# buying and holding the basket of 10 energy stocks returned ~2500% while S&P 500 returned ~500%
```

In [22]:

In [22]:

In [23]:
```
#####################################################################################################
#####################################################################################################
#####################################################################################################
#####################################################################################################
#####################################################################################################

#------------------------------------         Part 2: Applies a moving average strategy using the moving av

#------------------------------------         uses a 200 day moving average, calculates the bollinger band

#------------------------------------         and the relative strength index over a 6 day period.    ----

#------------------------------------         Sells when rsi is > 70 and price is > than the upper thresho

# starting with the first buy when the stock price and rsi move below the thresholds, we invest all of the


# rsi explanation https://www.investopedia.com/terms/r/rsi.asp
```

```
# bollinger explanation https://www.investopedia.com/terms/b/bollingerbands.asp


#------------------------------------   example from https://github.com/bryancwh/algo-trading-mean-reversi

##########################################################################################################
##########################################################################################################
##########################################################################################################
##########################################################################################################
##########################################################################################################
```

In [24]:
```
# applying the transformation from https://github.com/bryancwh/algo-trading-mean-reversion/blob/main/Mean%2

# changed the function to take in periods and thresholds as arguments so we can experiment with differnt st

# https://medium.com/@redsword_23261/bollinger-bands-and-rsi-crossover-trading-strategy-85246fc52379

# 20 day ma is standard for bollinger bands, 2 std dev is also standard

# rsi period of 6-14 is standard, 30 and 70 are common threshholds

def gain(value):
      if value < 0:
          return 0
      else:
          return value

def loss(value):
      if value > 0:
          return 0
      else:
          return abs(value)

def apply_mean_reversion_strategy(stock_df_dict, ma_period = 200, bollinger_period = 20, rsi_period = 6, bo

  updated_stock_df_dict = {}


  for ticker, df in stock_df_dict.items():

      df = df.copy()
```

```
    df['date'] = pd.to_datetime(df['date'])

    # moving average
    df['ma_200'] = df['close'].rolling(ma_period).mean()

    #Bollinger
    bollinger_period = bollinger_period
    ma_period_column = f'ma_{bollinger_period}'
    df[ma_period_column] = df['close'].rolling(bollinger_period).mean()
    df['std'] = df['close'].rolling(bollinger_period).std()
    df['upper_bollinger'] = df[ma_period_column] + (bollinger_std * df['std'])
    df['lower_bollinger'] = df[ma_period_column] - (bollinger_std * df['std'])

    # rsi
    rsi_period = rsi_period
    df['delta'] = df['close'].diff()
    df['gain'] = df['delta'].apply(lambda x: gain(x))
    df['loss'] = df['delta'].apply(lambda x: loss(x))
    df['ema_gain'] = df['gain'].ewm(span=rsi_period, adjust=False).mean()
    df['ema_loss'] = df['loss'].ewm(span=rsi_period, adjust=False).mean()
    df['rs'] = df['ema_gain'] / df['ema_loss']
    df['rsi'] = df['rs'].apply(lambda x: 100 - (100/(x+1)))

    # buy
    df['signal'] = np.where(
        (df['rsi'] < rsi_low_threshold) & (df['close'] < df['lower_bollinger']),
        1, np.nan
    )

    # sell
    df['signal'] = np.where(
        (df['rsi'] > rsi_high_threshold) & (df['close'] > df['upper_bollinger']),
        -1, df['signal']
    )

    #buy/sell next trading day
    df['signal'] = df['signal'].shift()
    df['signal'] = df['signal'].fillna(0)

    updated_stock_df_dict[ticker] = df
```

```
    return updated_stock_df_dict
```

In [25]:
```
# making dfs with various valus for testing

stock_dfs_original = apply_mean_reversion_strategy(stock_dfs, ma_period = 200, bollinger_period = 20, rsi_p

stock_dfs_bollinger_period_20_rsi_period_14_rsilower_30_rsi_upper_70 = apply_mean_reversion_strategy(stock_

stock_dfs_bollinger_period_20_rsi_period_14_rsilower_20_rsi_upper_80 = apply_mean_reversion_strategy(stock_

stock_dfs_bollinger_period_30_rsi_period_6_rsilower_30_rsi_upper_70 = apply_mean_reversion_strategy(stock_d

stock_dfs_bollinger_period_50_rsi_period_6_rsilower_30_rsi_upper_70 = apply_mean_reversion_strategy(stock_d

stock_dfs_bollinger_period_20_not_using_rsi = apply_mean_reversion_strategy(stock_dfs, ma_period = 200, bol

stock_dfs_bollinger_period_50_not_using_rsi = apply_mean_reversion_strategy(stock_dfs, ma_period = 200, bol
```

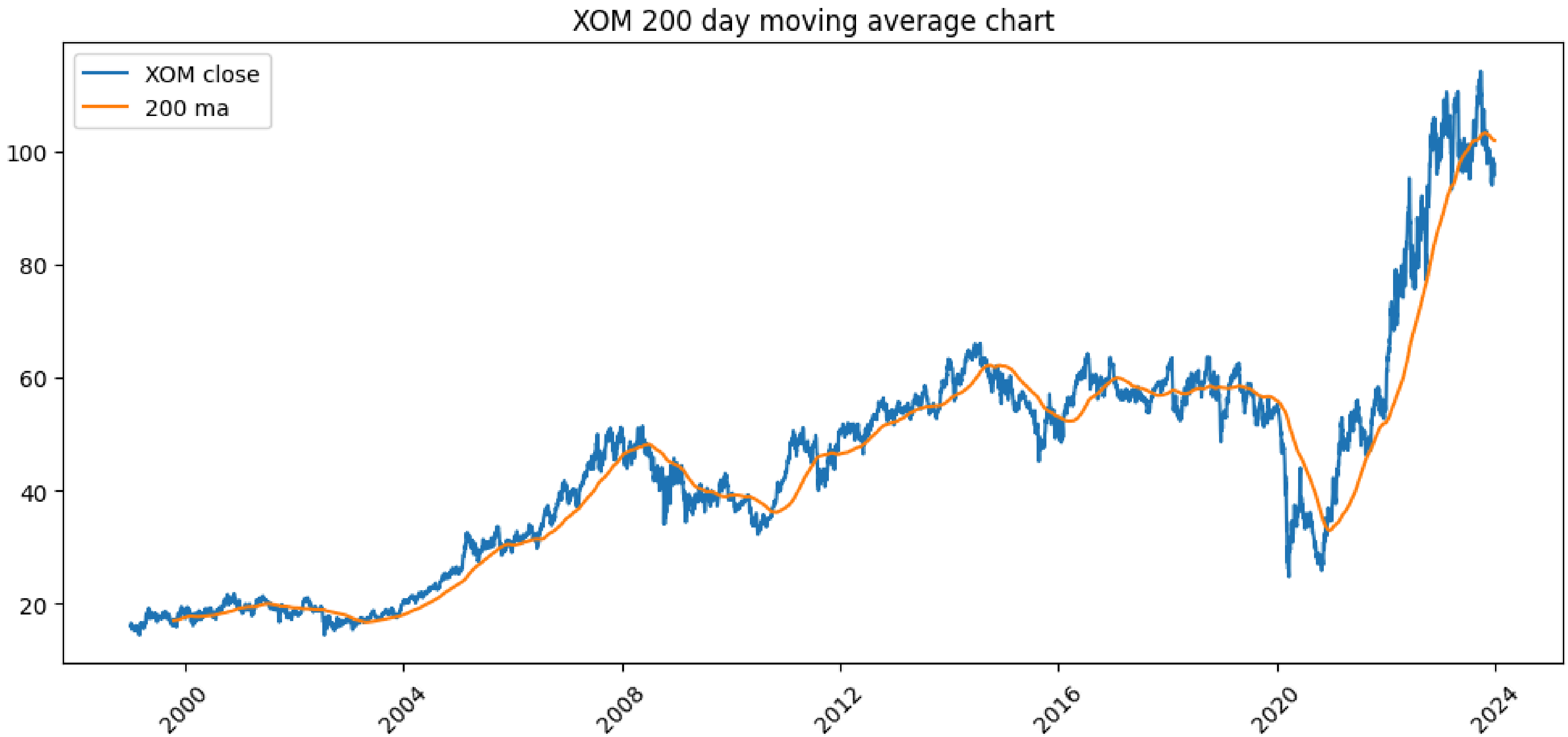
In [26]:
```
display(stock_dfs_bollinger_period_20_not_using_rsi["XOM"][500:502])
```

	date	close	high	low	open	volume	ma_200	ma_20	std	upper_bollinger	low
500	2000-12-26	20.482729	20.540549	20.077989	20.077989	5580600	19.174942	20.226871	0.486072	21.199015	
501	2000-12-27	20.294815	20.656190	20.164720	20.540550	10437800	19.188477	20.165437	0.379542	20.924522	

In [27]:
```
display(stock_dfs_bollinger_period_50_not_using_rsi["XOM"][500:502])
```

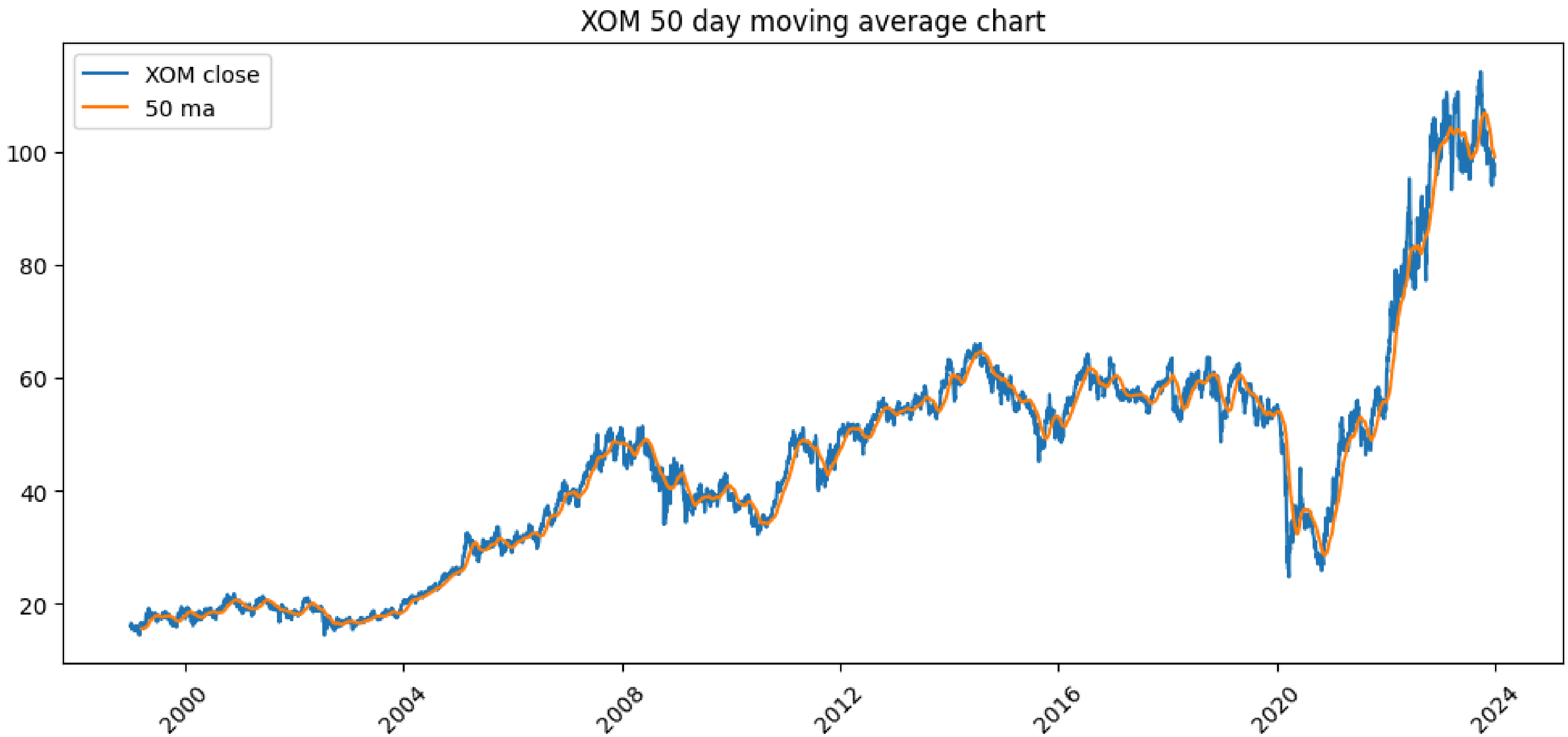
	date	close	high	low	open	volume	ma_200	ma_50	std	upper_bollinger	low
500	2000-12-26	20.482729	20.540549	20.077989	20.077989	5580600	19.174942	20.484620	0.557019	21.598658	
501	2000-12-27	20.294815	20.656190	20.164720	20.540550	10437800	19.188477	20.486301	0.556302	21.598905	

In [28]:
```
#  plot showing XOM 200 day moving avg

plt.figure(figsize=(12,5))
plt.xticks(rotation=45)

plt.plot(stock_dfs_bollinger_period_50_not_using_rsi["XOM"]['date'], stock_dfs_bollinger_period_50_not_usi
plt.plot(stock_dfs_bollinger_period_50_not_using_rsi["XOM"]['date'], stock_dfs_bollinger_period_50_not_usi

plt.title('XOM 200 day moving average chart')
plt.legend()
plt.show()
```



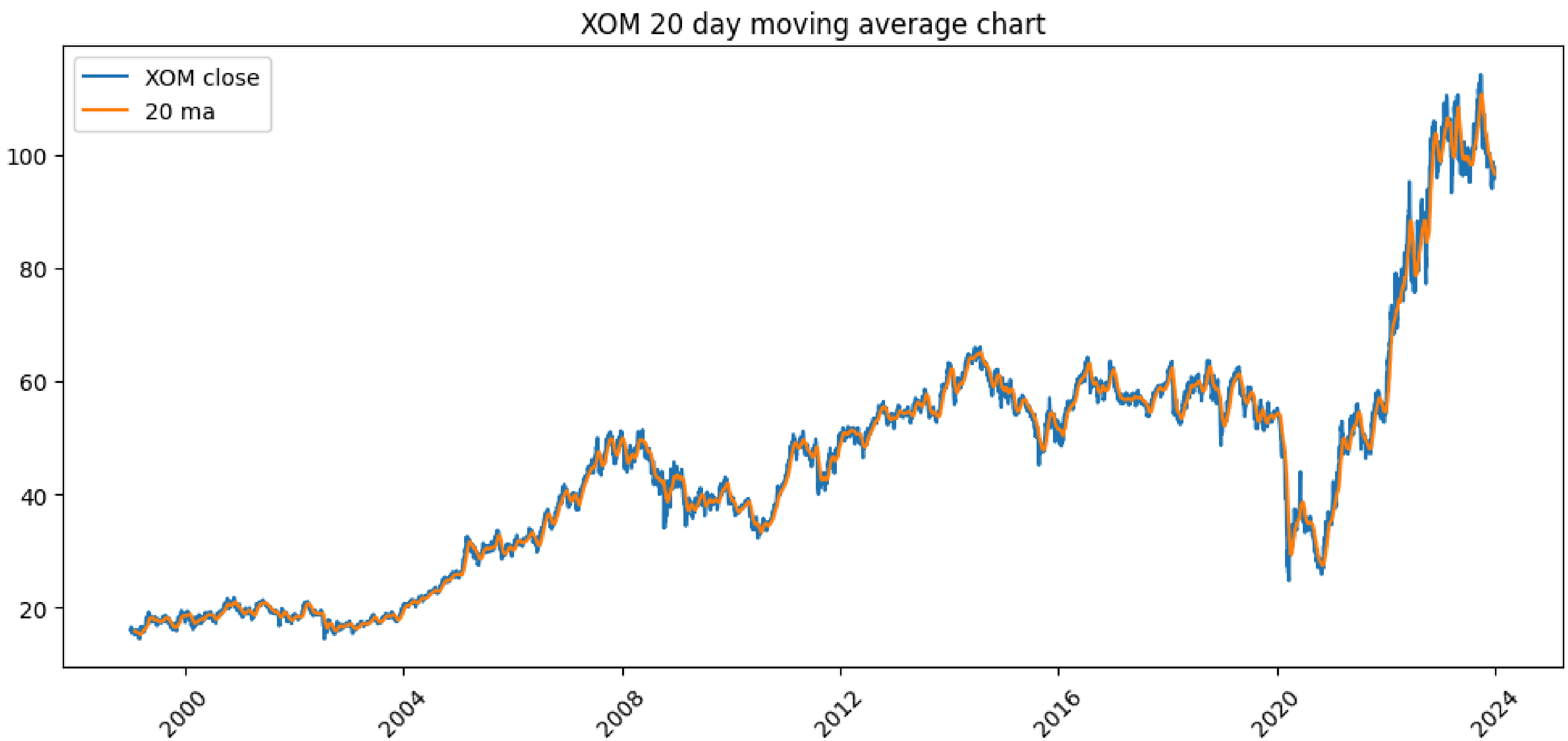
In [29]:
```
#  plot showing XOM 50 day moving avg


plt.figure(figsize=(12,5))
plt.xticks(rotation=45)
```

```
plt.plot(stock_dfs_bollinger_period_50_not_using_rsi["XOM"]['date'], stock_dfs_bollinger_period_50_not_usi
plt.plot(stock_dfs_bollinger_period_50_not_using_rsi["XOM"]['date'], stock_dfs_bollinger_period_50_not_usi

plt.title('XOM 50 day moving average chart')
plt.legend()
plt.show()
```



In [30]:
```
#  plot showing XOM 50 day moving avg


plt.figure(figsize=(12,5))
plt.xticks(rotation=45)

plt.plot(stock_dfs_bollinger_period_20_not_using_rsi["XOM"]['date'], stock_dfs_bollinger_period_20_not_usi
plt.plot(stock_dfs_bollinger_period_20_not_using_rsi["XOM"]['date'], stock_dfs_bollinger_period_20_not_usi
```

```
plt.title('XOM 20 day moving average chart')
plt.legend()
plt.show()
```



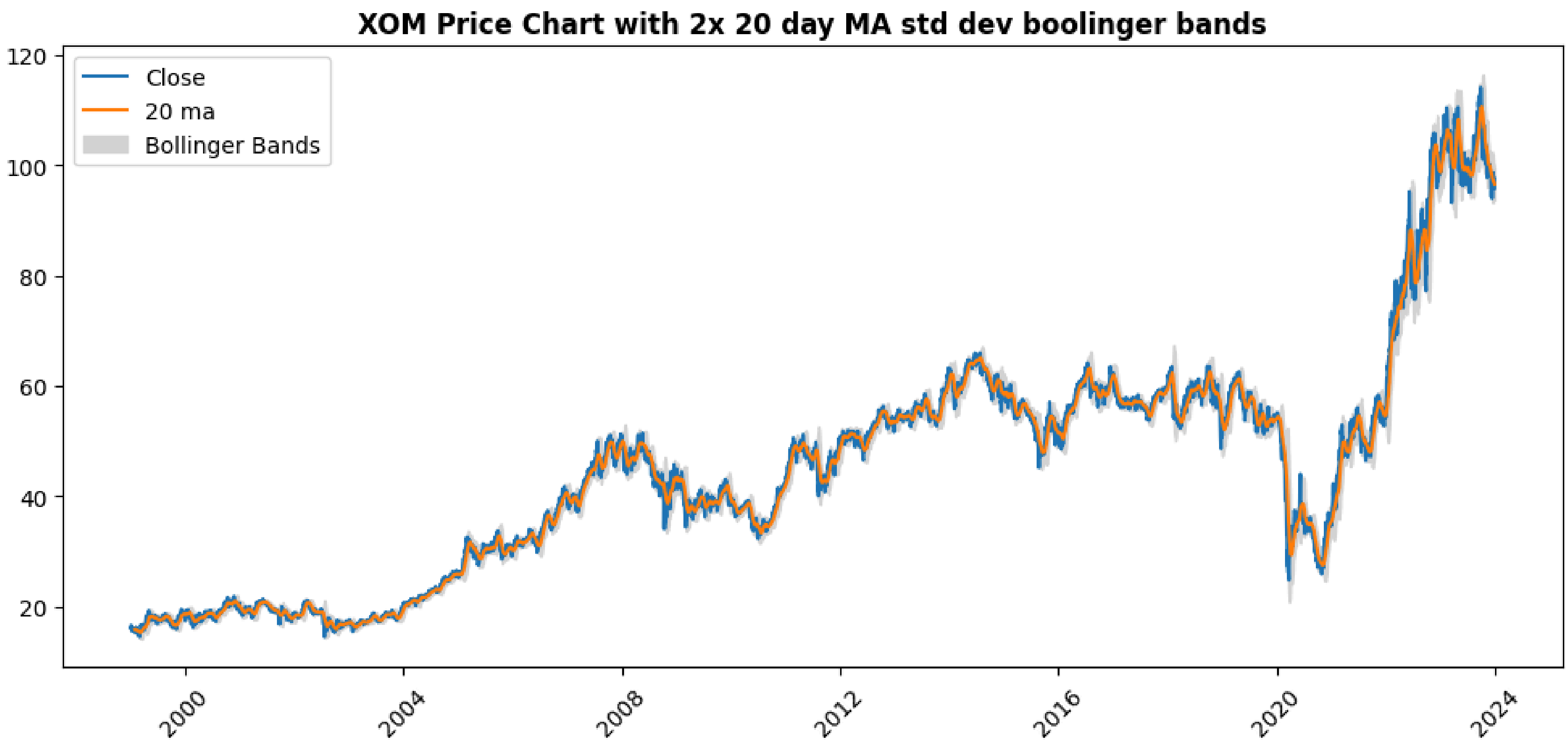
In [30]:

```

```

In [31]:

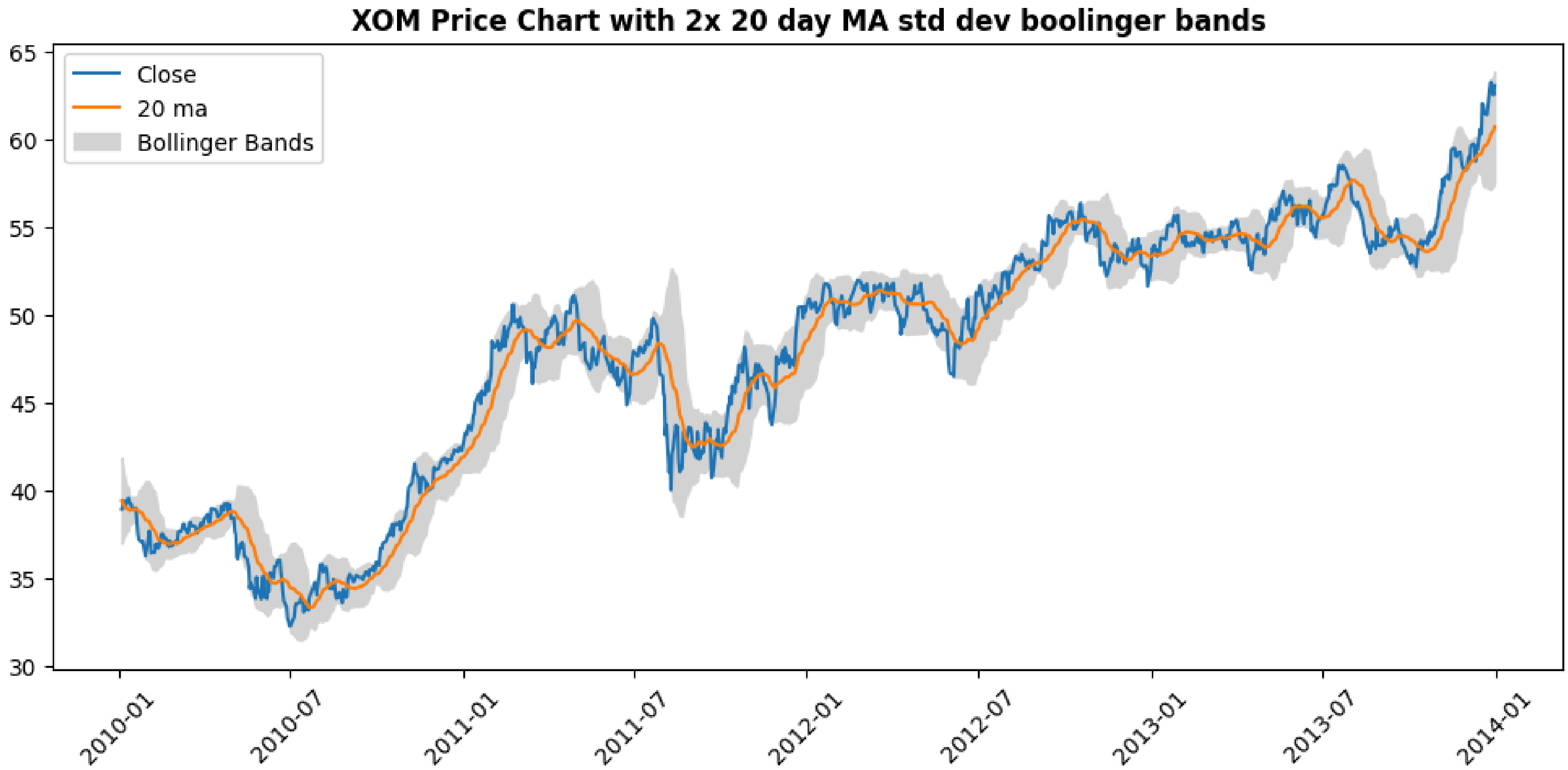
```
# XOM bollinger plot using 2x std dev of 20 day ma

plt.figure(figsize=(12,5))
plt.xticks(rotation=45)

x_axis = stock_dfs_bollinger_period_20_not_using_rsi['XOM']['date']

plt.plot(x_axis, stock_dfs_bollinger_period_20_not_using_rsi['XOM']['close'], label = 'Close')
plt.plot(stock_dfs_bollinger_period_20_not_using_rsi['XOM']['date'], stock_dfs_bollinger_period_20_not_usi
plt.fill_between(x_axis, stock_dfs_bollinger_period_20_not_using_rsi['XOM']['upper_bollinger'], stock_dfs_
```

```
plt.title('XOM Price Chart with 2x 20 day MA std dev boolinger bands', fontweight="bold")
plt.legend()
plt.show()
```



In [32]:
```
# # XOM bollinger plot using 2x std dev of 20 day ma. using just 2010 to 2014 to show more detail


XOM_df = stock_dfs_bollinger_period_20_not_using_rsi['XOM']
XOM_truncated = XOM_df[(XOM_df['date'] > '2010-01-01') & (XOM_df['date'] < '2014-01-01')]

plt.figure(figsize=(12,5))
plt.xticks(rotation=45)

x_axis = XOM_truncated['date']

plt.plot(x_axis, XOM_truncated['close'], label = 'Close')
plt.plot(XOM_truncated['date'], XOM_truncated['ma_20'], label = '20 ma')
```

```
plt.fill_between(x_axis, XOM_truncated['upper_bollinger'], XOM_truncated['lower_bollinger'], label = 'Boll

plt.title('XOM Price Chart with 2x 20 day MA std dev boolinger bands', fontweight="bold")
plt.legend()
plt.show()
```



In [33]:
```
# define backtesting function from https://github.com/bryancwh/algo-trading-mean-reversion/blob/main/back_

def backtest_dataframe(df):
    position = 0
    net_profit = 0
    percentage_change = []
    df['buy_date'] = ''
    df['sell_date'] = ''

    for i in df.index:
```

```
        close = df["close"][i]
        date = df['date'][i]

        # Buy action
        if df["signal"][i] == 1:
            if(position == 0):
                buy_price = close
                position = 1
                df.at[i, 'buy_date'] = date
                print(f"Buying at {str(buy_price)} on {str(date)}")

        # Sell action
        elif (df["signal"][i] == -1):
            if(position == 1):
                sell_price = close
                bought = 0
                position = 0
                df.at[i, 'sell_date'] = date
                print(f"Selling at {str(sell_price)} on {str(date)}")

                # Get percentage change of trade
                pc = (sell_price/buy_price-1)*100
                percentage_change.append(pc)
                net_profit += (sell_price - buy_price)

    # Calculate trade statistics
    gains = 0
    ng = 0
    losses = 0
    nl = 0
    totalR = 1

    for i in percentage_change:
        if(i > 0):
            gains += i
            ng += 1
        else:
            losses += i
            nl += 1
        totalR = totalR * ((i/100)+1)

    totalR = round((totalR-1)*100, 2)
```

```
    if(ng > 0):
        avgGain = round(gains/ng, 2)
        maxR = round(max(percentage_change), 2)
    else:
        avgGain = 0
        maxR = "undefined"

    if(nl > 0):
        avgLoss = round(losses/nl, 2)
        maxL = round(min(percentage_change), 2)
    else:
        avgLoss = 0
        maxL = "undefined"

    if(ng > 0 or nl > 0):
        win_rate = round((ng/(ng+nl))*100, 2)
    else:
        win_rate = 0

    print()
    print('Evaluation Metrics:')
    print('-----------------------------------')
    print(f"Number of Trades: {ng+nl}")
    print(f"Number of Gains: {ng}")
    print(f"Number of Losses: {nl}")
    print(f"Total Returns: {totalR}%")
    print(f"Win Rate: {win_rate}%")
    print(f"Average Gain: {avgGain}%")
    print(f"Average Loss: {avgLoss}%")
    print(f"Max Return: {maxR}%")
    print(f"Max Loss: {maxL}%")
    print()
```

In [34]:
```
backtest_dataframe(stock_dfs_original["XOM"])
```

```
Buying at 14.441488265991213 on 1999-03-02 00:00:00
Selling at 16.655757904052734 on 1999-03-11 00:00:00
Buying at 17.494253158569336 on 1999-06-23 00:00:00
Selling at 17.998096466064453 on 1999-08-06 00:00:00
Buying at 17.751846313476562 on 1999-08-31 00:00:00
Selling at 17.118858337402344 on 1999-10-21 00:00:00
Buying at 17.390628814697266 on 2000-01-04 00:00:00
Selling at 18.156892776489254 on 2000-04-05 00:00:00
Buying at 18.208953857421875 on 2000-07-03 00:00:00
Selling at 18.64173889160156 on 2000-08-03 00:00:00
Buying at 20.35263061523437 on 2000-12-07 00:00:00
Selling at 20.340789794921875 on 2001-04-24 00:00:00
Buying at 19.96890640258789 on 2001-07-12 00:00:00
Selling at 18.811023712158203 on 2001-12-27 00:00:00
Buying at 20.56357192993164 on 2002-04-08 00:00:00
Selling at 19.443622589111328 on 2002-07-01 00:00:00
Buying at 17.745586395263672 on 2002-07-11 00:00:00
Selling at 17.474584579467773 on 2002-10-15 00:00:00
Buying at 16.228178024291992 on 2003-01-22 00:00:00
Selling at 16.86028480529785 on 2003-03-06 00:00:00
Buying at 17.77635955810547 on 2003-07-01 00:00:00
Selling at 17.94340705871582 on 2003-08-08 00:00:00
Buying at 18.096776962280277 on 2003-10-31 00:00:00
Selling at 18.323244094848636 on 2003-12-05 00:00:00
Buying at 21.06874084472656 on 2004-03-12 00:00:00
Selling at 22.58726692199707 on 2004-06-17 00:00:00
Buying at 30.198169708251957 on 2005-03-28 00:00:00
Selling at 31.36215591430664 on 2005-06-20 00:00:00
Buying at 30.531797409057617 on 2005-10-05 00:00:00
Selling at 31.05725860595703 on 2005-11-22 00:00:00
Buying at 30.15667152404785 on 2005-12-20 00:00:00
Selling at 32.66581344604492 on 2006-01-31 00:00:00
Buying at 30.97106170654297 on 2006-03-10 00:00:00
Selling at 31.94970321655273 on 2006-03-17 00:00:00
Buying at 31.500288009643555 on 2006-05-18 00:00:00
Selling at 32.268184661865234 on 2006-06-30 00:00:00
Buying at 34.18806457519531 on 2006-09-12 00:00:00
Selling at 36.6768913269043 on 2006-10-17 00:00:00
Buying at 38.59252166748047 on 2007-01-04 00:00:00
Selling at 40.22250747680664 on 2007-03-26 00:00:00
Buying at 44.71571350097656 on 2007-08-06 00:00:00
Selling at 49.5053939819336 on 2007-09-19 00:00:00
```

```
Buying at 47.25367736816406 on 2007-11-02 00:00:00
Selling at 49.36336135864258 on 2007-12-07 00:00:00
Buying at 45.26860427856445 on 2008-01-17 00:00:00
Selling at 47.09488677978516 on 2008-02-21 00:00:00
Buying at 46.66852951049805 on 2008-06-04 00:00:00
Selling at 44.09568786621094 on 2008-11-28 00:00:00
Buying at 39.77827453613281 on 2009-02-18 00:00:00
Selling at 38.5305061340332 on 2009-05-11 00:00:00
Buying at 37.2100830078125 on 2009-08-18 00:00:00
Selling at 40.81972122192383 on 2009-10-15 00:00:00
Buying at 38.93519973754883 on 2009-12-15 00:00:00
Selling at 37.66515350341797 on 2010-03-08 00:00:00
Buying at 37.48952102661133 on 2010-05-05 00:00:00
Selling at 35.77620315551758 on 2010-08-03 00:00:00
Buying at 47.724609375 on 2011-03-11 00:00:00
Selling at 51.01396179199219 on 2011-04-27 00:00:00
Buying at 46.30036926269531 on 2011-06-16 00:00:00
Selling at 44.87060928344727 on 2011-10-11 00:00:00
Buying at 43.7353630065918 on 2011-11-25 00:00:00
Selling at 50.47023391723633 on 2011-12-27 00:00:00
Buying at 49.69496536254883 on 2012-02-01 00:00:00
Selling at 52.4621696472168 on 2012-07-30 00:00:00
Buying at 54.20216751098633 on 2012-10-24 00:00:00
Selling at 55.317474365234375 on 2013-03-27 00:00:00
Buying at 52.89299392700195 on 2013-04-16 00:00:00
Selling at 55.97686004638672 on 2013-05-09 00:00:00
Buying at 55.02372741699219 on 2013-06-21 00:00:00
Selling at 57.07757186889648 on 2013-07-10 00:00:00
Buying at 56.24741363525391 on 2013-08-06 00:00:00
Selling at 55.280181884765625 on 2013-09-19 00:00:00
Buying at 53.249290466308594 on 2013-10-01 00:00:00
Selling at 56.95816421508789 on 2013-11-05 00:00:00
Buying at 59.12861633300781 on 2014-01-24 00:00:00
Selling at 61.3240852355957 on 2014-03-31 00:00:00
Buying at 63.45336532592773 on 2014-06-03 00:00:00
Selling at 65.052490234375 on 2014-06-16 00:00:00
Buying at 62.4483757019043 on 2014-08-01 00:00:00
Selling at 56.89017105102539 on 2015-04-15 00:00:00
Buying at 55.42656707763672 on 2015-05-27 00:00:00
Selling at 50.48932266235352 on 2015-10-05 00:00:00
Buying at 49.48398971557617 on 2015-12-08 00:00:00
Selling at 53.0976791381836 on 2016-02-05 00:00:00
```

```
Buying at 60.85784912109375 on 2016-07-28 00:00:00
Selling at 59.09048080444336 on 2016-09-09 00:00:00
Buying at 57.56629180908203 on 2016-09-14 00:00:00
Selling at 62.45574951171875 on 2016-12-12 00:00:00
Buying at 58.98904037475586 on 2017-01-10 00:00:00
Selling at 56.8128547668457 on 2017-03-31 00:00:00
Buying at 56.120079040527344 on 2017-04-20 00:00:00
Selling at 57.72695541381836 on 2017-05-12 00:00:00
Buying at 56.43326187133789 on 2017-06-01 00:00:00
Selling at 61.78324508666992 on 2018-01-03 00:00:00
Buying at 56.80925369262695 on 2018-02-05 00:00:00
Selling at 53.89866256713867 on 2018-04-06 00:00:00
Buying at 58.11003494262695 on 2018-08-02 00:00:00
Selling at 61.075103759765625 on 2018-09-12 00:00:00
Buying at 59.78940963745117 on 2018-10-12 00:00:00
Selling at 55.52707290649414 on 2019-02-04 00:00:00
Buying at 57.99696731567383 on 2019-05-02 00:00:00
Selling at 58.40369415283203 on 2019-06-24 00:00:00
Buying at 56.91609954833984 on 2019-07-19 00:00:00
Selling at 55.3800048828125 on 2019-09-10 00:00:00
Buying at 51.59937286376953 on 2019-10-02 00:00:00
Selling at 56.16377258300781 on 2019-11-05 00:00:00
Buying at 52.252777099609375 on 2020-01-22 00:00:00
Selling at 34.02315902709961 on 2020-04-09 00:00:00
Buying at 34.296852111816406 on 2020-07-10 00:00:00
Selling at 30.954837799072266 on 2020-11-10 00:00:00
Buying at 47.9606819152832 on 2021-07-19 00:00:00
Selling at 50.65692901611328 on 2021-09-24 00:00:00
Buying at 54.83214569091797 on 2021-11-22 00:00:00
Selling at 58.77228164672852 on 2022-01-04 00:00:00
Buying at 83.29706573486328 on 2022-06-21 00:00:00
Selling at 84.3532943725586 on 2022-07-28 00:00:00
Buying at 77.21092987060547 on 2022-09-26 00:00:00
Selling at 104.8672103881836 on 2023-01-13 00:00:00
Buying at 94.35489654541016 on 2023-03-16 00:00:00
Selling at 107.42057037353516 on 2023-04-04 00:00:00
Buying at 100.79901123046876 on 2023-05-03 00:00:00
Selling at 101.91862487792967 on 2023-06-08 00:00:00
Buying at 95.5033721923828 on 2023-07-17 00:00:00
Selling at 105.43241119384766 on 2023-08-14 00:00:00
Buying at 103.51390075683594 on 2023-10-05 00:00:00
```

```
Evaluation Metrics:
-----------------------------------
Number of Trades: 62
Number of Gains: 42
Number of Losses: 20
Total Returns: 239.87%
Win Rate: 67.74%
Average Gain: 6.4%
Average Loss: -6.08%
Max Return: 35.82%
Max Loss: -34.89%
```

In [35]:
```
# same function as above but removed the print staements and made it just return the total return for the s

# from https://github.com/bryancwh/algo-trading-mean-reversion/blob/main/back_testing.py


def backtest_dataframe_return_just_total_return(df):
    position = 0
    net_profit = 0
    percentage_change = []
    df['buy_date'] = ''
    df['sell_date'] = ''

    for i in df.index:
        close = df["close"][i]
        date = df['date'][i]

        # Buy action
        if df["signal"][i] == 1:
            if(position == 0):
                buy_price = close
                position = 1
                df.at[i, 'buy_date'] = date
                #print(f"Buying at {str(buy_price)} on {str(date)}")

        # Sell action
        elif (df["signal"][i] == -1):
            if(position == 1):
                sell_price = close
                bought = 0
```

```
                position = 0
                df.at[i, 'sell_date'] = date
                #print(f"Selling at {str(sell_price)} on {str(date)}")

                # Get percentage change of trade
                pc = (sell_price/buy_price-1)*100
                percentage_change.append(pc)
                net_profit += (sell_price - buy_price)

    # Calculate trade statistics
    gains = 0
    ng = 0
    losses = 0
    nl = 0
    totalR = 1

    for i in percentage_change:
        if(i > 0):
            gains += i
            ng += 1
        else:
            losses += i
            nl += 1
        totalR = totalR * ((i/100)+1)

    totalR = round((totalR-1), 2)

    if(ng > 0):
        avgGain = round(gains/ng, 2)
        maxR = round(max(percentage_change), 2)
    else:
        avgGain = 0
        maxR = "undefined"

    if(nl > 0):
        avgLoss = round(losses/nl, 2)
        maxL = round(min(percentage_change), 2)
    else:
        avgLoss = 0
        maxL = "undefined"

    if(ng > 0 or nl > 0):
```

```
        win_rate = round((ng/(ng+nl))*100, 2)
    else:
        win_rate = 0

    return totalR
```

In [35]:

In [36]:
```
# return on individual stocks using default ma strategy


ma_strategy_return = {}

for ticker in tickers:
    df = stock_dfs_original[ticker]

    total_return = backtest_dataframe_return_just_total_return(df)
    ma_strategy_return[ticker] = total_return


print(ma_strategy_return)
```

```
{'XOM': 2.4, 'CVX': 3.93, 'COP': 2.07, 'EOG': 3.09, 'EPD': 3.28, 'WMB': -0.83, 'OKE': 1.47, 'LNG': -0.63,
'OXY': 1.31, 'HES': 2.49}
```

In [37]:
```
total_ma_strategy_returns_energy_stocks = 0

for ticker, ma_returns in ma_strategy_return.items():
  final_value = 10000 * ma_returns
  total_ma_strategy_returns_energy_stocks += final_value

print(f'{total_ma_strategy_returns_energy_stocks:.2f}')
```

```
185800.00
```

In [38]:
```
# return on 100k using default ma strategy

overall_ma_energy_return = total_ma_strategy_returns_energy_stocks / 100000
print(f'{overall_ma_energy_return:.2f}')
```

```
1.86
```

In [38]:

In [38]:

In [39]:
```
#########################################################################################################
#########################################################################################################
#########################################################################################################
#########################################################################################################
#########################################################################################################

#-------------------------------------         Part 2b: trying different values for moving average and rsi p

#########################################################################################################
#########################################################################################################
#########################################################################################################
#########################################################################################################
#########################################################################################################
```

In [40]:
```
strategies = {
    "original": stock_dfs_original,
    "bollinger20_rsi14_30_70": stock_dfs_bollinger_period_20_rsi_period_14_rsilower_30_rsi_upper_70,
    "bollinger20_rsi14_20_80": stock_dfs_bollinger_period_20_rsi_period_14_rsilower_20_rsi_upper_80,
    "bollinger30_rsi6_30_70": stock_dfs_bollinger_period_30_rsi_period_6_rsilower_30_rsi_upper_70,
    "bollinger50_rsi6_30_70": stock_dfs_bollinger_period_50_rsi_period_6_rsilower_30_rsi_upper_70,
    "bollinger20_no_rsi": stock_dfs_bollinger_period_20_not_using_rsi,
    "bollinger50_no_rsi": stock_dfs_bollinger_period_50_not_using_rsi,
}
```

In [41]:
```
strategy_returns = {}

for strategy, stock_dict in strategies.items():
    ma_strategy_return = {}

    for ticker in tickers:
        df = stock_dict[ticker]
        total_return = backtest_dataframe_return_just_total_return(df)
        ma_strategy_return[ticker] = total_return

    total_ma_strategy_returns = 0
    for ticker, return_multiplier in ma_strategy_return.items():
```

```
        final_value = 10000 * return_multiplier
        total_ma_strategy_returns += final_value

    print(f"Strategy: {strategy}")
    print()
    print(ma_strategy_return)
    print(f"Total return on 100k: {total_ma_strategy_returns:.2f}")

    overall_return = total_ma_strategy_returns / 100000
    print(f"Overall return : {overall_return:.2f}")
    print("----------------------------------------------------")
    print()

    strategy_returns[strategy] = overall_return
```

```
Strategy: original

{'XOM': 2.4, 'CVX': 3.93, 'COP': 2.07, 'EOG': 3.09, 'EPD': 3.28, 'WMB': -0.83, 'OKE': 1.47, 'LNG': -0.63,
'OXY': 1.31, 'HES': 2.49}
Total return on 100k: 185800.00
Overall return : 1.86
-----------------------------------------------------

Strategy: bollinger20_rsi14_30_70

{'XOM': 2.92, 'CVX': 2.64, 'COP': 1.59, 'EOG': 3.0, 'EPD': 6.56, 'WMB': -0.86, 'OKE': 1.51, 'LNG': -0.73,
'OXY': 1.95, 'HES': 2.5}
Total return on 100k: 210800.00
Overall return : 2.11
-----------------------------------------------------

Strategy: bollinger20_rsi14_20_80

{'XOM': 6.33, 'CVX': 2.13, 'COP': 2.82, 'EOG': 4.85, 'EPD': 5.1, 'WMB': -0.39, 'OKE': 2.42, 'LNG': 0.48, 'O
XY': 3.41, 'HES': -0.03}
Total return on 100k: 271200.00
Overall return : 2.71
-----------------------------------------------------

Strategy: bollinger30_rsi6_30_70

{'XOM': 4.24, 'CVX': 4.87, 'COP': 1.04, 'EOG': 5.09, 'EPD': 3.9, 'WMB': -0.34, 'OKE': 1.86, 'LNG': -0.51,
'OXY': 1.2, 'HES': 0.71}
Total return on 100k: 220600.00
Overall return : 2.21
-----------------------------------------------------

Strategy: bollinger50_rsi6_30_70

{'XOM': 5.33, 'CVX': 2.97, 'COP': 0.74, 'EOG': 1.82, 'EPD': 2.69, 'WMB': -0.3, 'OKE': 2.55, 'LNG': 1.8, 'OX
Y': 3.68, 'HES': 0.88}
Total return on 100k: 221600.00
Overall return : 2.22
-----------------------------------------------------

Strategy: bollinger20_no_rsi
```

```
{'XOM': 2.42, 'CVX': 3.94, 'COP': 2.07, 'EOG': 3.11, 'EPD': 3.28, 'WMB': -0.83, 'OKE': 1.49, 'LNG': -0.63,
'OXY': 1.3, 'HES': 2.49}
Total return on 100k: 186400.00
Overall return : 1.86
-----------------------------------------------------

Strategy: bollinger50_no_rsi

{'XOM': 5.33, 'CVX': 2.97, 'COP': 0.74, 'EOG': 1.82, 'EPD': 2.69, 'WMB': -0.3, 'OKE': 2.55, 'LNG': 1.8, 'OX
Y': 3.5, 'HES': 0.88}
Total return on 100k: 219800.00
Overall return : 2.20
-----------------------------------------------------
```

In [42]:
```
df_best_stock_in_best_strategy = stock_dfs_bollinger_period_20_rsi_period_14_rsilower_20_rsi_upper_80['XOM
df_worst_stock_in_best_strategy = stock_dfs_bollinger_period_20_rsi_period_14_rsilower_20_rsi_upper_80['WMI
```

In [43]:
```
df_best_stock_in_best_strategy.head()
```

Out[43]:

	date	close	high	low	open	volume	ma_200	ma_20	std	upper_bollinger	...	delta
0	1999-01-04	16.089716	16.422034	16.006637	16.117409	8853600	NaN	NaN	NaN	NaN	...	NaN
1	1999-01-05	15.951257	16.158955	15.882024	16.062029	6652800	NaN	NaN	NaN	NaN	...	-0.138459
2	1999-01-06	16.588202	16.795900	16.103571	16.200497	9965600	NaN	NaN	NaN	NaN	...	0.636945
3	1999-01-07	16.560503	16.602043	16.338958	16.491270	7417200	NaN	NaN	NaN	NaN	...	-0.027699
4	1999-01-08	16.463579	16.546659	16.158955	16.449733	6343400	NaN	NaN	NaN	NaN	...	-0.096924

5 rows × 21 columns

In [44]:
```
df_worst_stock_in_best_strategy.head()
```

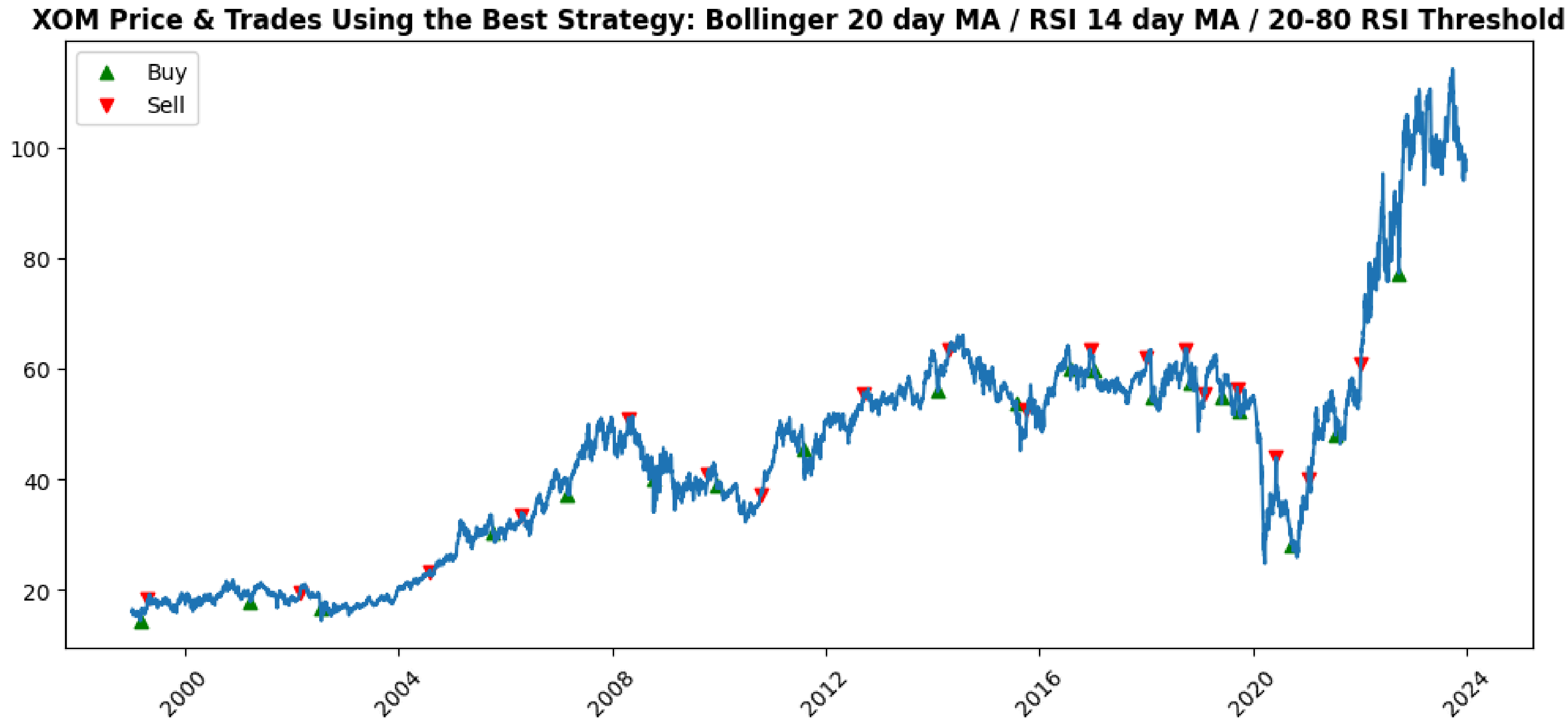
Out [44]:

	date	close	high	low	open	volume	ma_200	ma_20	std	upper_bollinger	...	delta	
0	1999-01-04	8.717077	8.969746	8.608791	8.951699	2419311	NaN	NaN	NaN	NaN	...	NaN	
1	1999-01-05	8.626838	8.771220	8.590742	8.662933	2414634	NaN	NaN	NaN	NaN	...	-0.090240	0.0
2	1999-01-06	8.897552	8.933647	8.753170	8.807313	2392051	NaN	NaN	NaN	NaN	...	0.270714	0.2
3	1999-01-07	8.897552	8.933647	8.680978	8.771217	1840173	NaN	NaN	NaN	NaN	...	0.000000	0.0
4	1999-01-08	8.897552	8.987790	8.771217	8.879504	1381299	NaN	NaN	NaN	NaN	...	0.000000	0.0

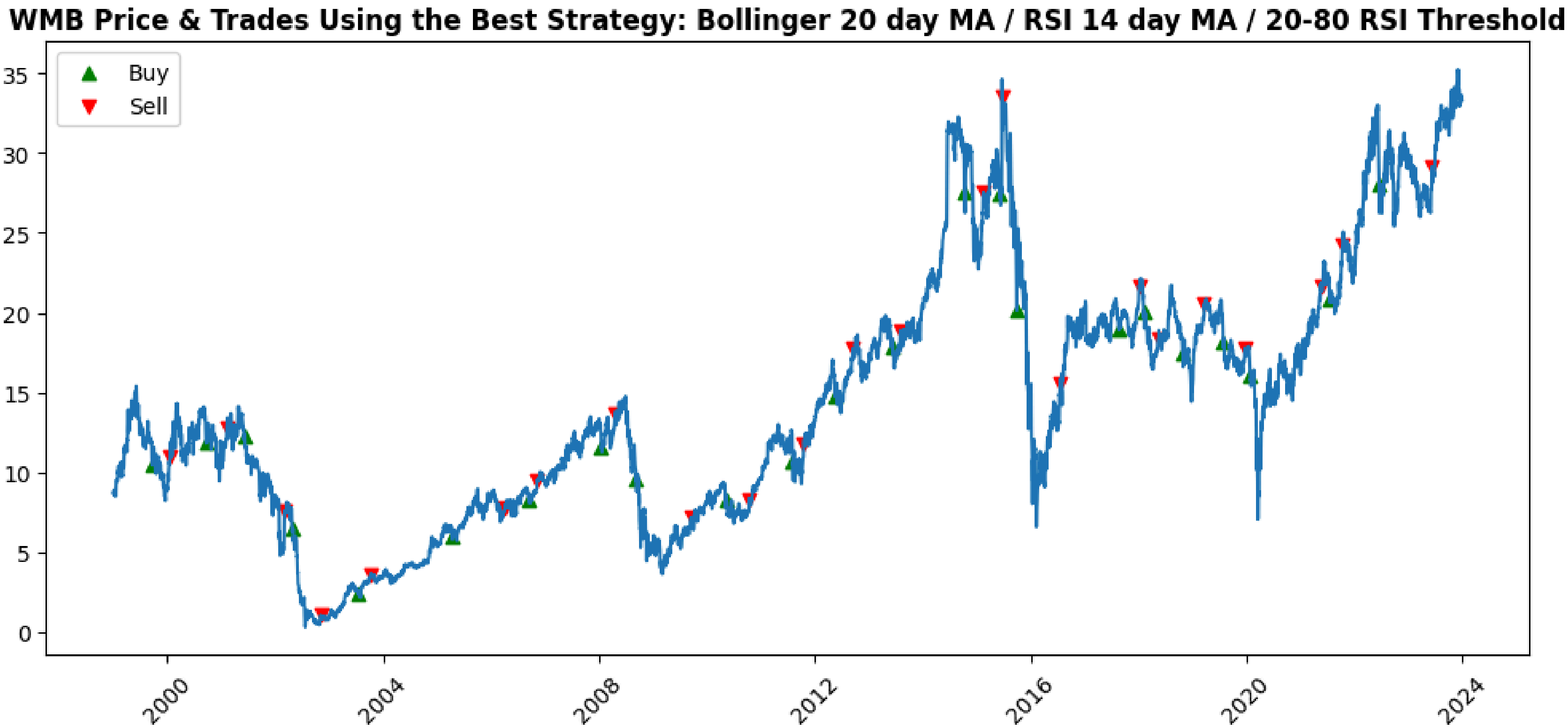
5 rows × 21 columns

In [45]:

```
# best strategy was stock_dfs_bollinger_period_20_rsi_period_14_rsilower_20_rsi_upper_80 with 2.71x return

# plotting the best performing stock (XOM 6.33x returns) in the basket based on the best strategy

plt.figure(figsize=(12,5))
plt.xticks(rotation=45)

plt.plot(df_best_stock_in_best_strategy['date'], df_best_stock_in_best_strategy['close'])
plt.scatter(df_best_stock_in_best_strategy[(df_best_stock_in_best_strategy['signal'] == 1)]['buy_date'], d
plt.scatter(df_best_stock_in_best_strategy[(df_best_stock_in_best_strategy['signal'] == -1)]['sell_date'],

plt.title('XOM Price & Trades Using the Best Strategy: Bollinger 20 day MA / RSI 14 day MA / 20-80 RSI Thre
plt.legend()
plt.show()
```



In [46]:
```
# best strategy was stock_dfs_bollinger_period_20_rsi_period_14_rsilower_20_rsi_upper_80 with 2.71x return

# plotting the worst performing stock (WMB -0.39x returns) in the basket based on the best strategy


plt.figure(figsize=(12,5))
plt.xticks(rotation=45)

plt.plot(df_worst_stock_in_best_strategy['date'], df_worst_stock_in_best_strategy['close'])
plt.scatter(df_worst_stock_in_best_strategy[(df_worst_stock_in_best_strategy['signal'] == 1)]['buy_date'],
plt.scatter(df_worst_stock_in_best_strategy[(df_worst_stock_in_best_strategy['signal'] == -1)]['sell_date']

plt.title('WMB Price & Trades Using the Best Strategy: Bollinger 20 day MA / RSI 14 day MA / 20-80 RSI Thre
plt.legend()
plt.show()
```



In [46]:

```

```

In [47]:

```
##########################################################################################################
##########################################################################################################
##########################################################################################################
##########################################################################################################
##########################################################################################################
#-------------------------------------         Part 3: Monte carlo simulation         ----------------------
#-------------------------------------                 ------------------------------------
#-------------------------------------                 ------------------------------------
##########################################################################################################
##########################################################################################################
##########################################################################################################
##########################################################################################################
##########################################################################################################
```

In [48]:
```
import numpy as np
import pandas as pd
from scipy.stats import skew, skewnorm

def generate_synthetic_data(stock_dfs, start_date='1999-01-04', end_date='2023-12-22'):
    synthetic_dfs = {}

    for ticker, df in stock_dfs.items():
        df['date'] = pd.to_datetime(df['date'])
        df = df.sort_values('date')

        # Calculate historical daily returns
        df['return'] = (df['close'].pct_change()) #/100   # *** use log returns intead? if we stick with pe
        historical_returns = df['return'].dropna()

        # Calculate parameters of the historical returns
        mean_return = historical_returns.mean()
        volatility = historical_returns.std()
        historical_skew = skew(historical_returns)

        # print(f"{ticker} - Mean: {mean_return:.4f}, Volatility: {volatility:.4f}, Skewness: {historical_s

        # Generate date range for the synthetic data
        synthetic_dates = pd.bdate_range(start=start_date, end=end_date, freq='B')
        num_days = len(synthetic_dates)

        # Generate synthetic returns with skewness
        volatility = max(abs(volatility), 0.001)

        synthetic_returns = skewnorm.rvs(a=historical_skew,
                                         loc=mean_return,
                                         scale=volatility,
                                         size=num_days)

        # Generate synthetic price series
        initial_price = df['close'].iloc[0]  # Start from the first known price
        synthetic_prices = [initial_price]

        for ret in synthetic_returns:
            synthetic_prices.append(synthetic_prices[-1] * (1 + ret))
```

```
        # Create a synthetic dataframe
        synthetic_df = pd.DataFrame({
            'date': synthetic_dates,
            'close': synthetic_prices[:-1]  # Match dates length
        })

        synthetic_dfs[ticker] = synthetic_df

    return synthetic_dfs

##############################################

def apply_mean_reversion_strategy(data):
  def gain(value):
      if value < 0:
          return 0
      else:
          return value

  def loss(value):
      if value > 0:
          return 0
      else:
          return abs(value)


  for ticker, df in data.items():
      df['date'] = pd.to_datetime(df['date'])

      # moving average
      df['ma_200'] = df['close'].rolling(200).mean()

      #Bollinger
      bollinger_period = 20
      df['ma_20'] = df['close'].rolling(bollinger_period).mean()
      df['std'] = df['close'].rolling(bollinger_period).std()
      df['upper_bollinger'] = df['ma_20'] + (2 * df['std'])
      df['lower_bollinger'] = df['ma_20'] - (2 * df['std'])

      # rsi
      rsi_period = 14    # *** changed from 6 to 14 to match best historical strategy
      df['delta'] = df['close'].diff()
```

```
      df['gain'] = df['delta'].apply(lambda x: gain(x))
      df['loss'] = df['delta'].apply(lambda x: loss(x))
      df['ema_gain'] = df['gain'].ewm(span=rsi_period, adjust=False).mean()
      df['ema_loss'] = df['loss'].ewm(span=rsi_period, adjust=False).mean()
      df['rs'] = df['ema_gain'] / df['ema_loss']
      df['rsi'] = df['rs'].apply(lambda x: 100 - (100/(x+1)))

      # buy
      df['signal'] = np.where(
          (df['rsi'] < 20) & (df['close'] < df['lower_bollinger']),          # *** changed from < 30 to < 2
          1, np.nan
      )

      # sell
      df['signal'] = np.where(
          (df['rsi'] > 80) & (df['close'] > df['upper_bollinger']),        # *** changed from > 70 to > 80
          -1, df['signal']
      )

      #buy/sell next trading day
      df['signal'] = df['signal'].shift()
      df['signal'] = df['signal'].fillna(0)

      stock_dfs[ticker] = df

  return stock_dfs

# ##########################################################

def backtest_dataframe_return_just_total_return(df):
    position = 0
    net_profit = 0
    percentage_change = []
    df['buy_date'] = ''
    df['sell_date'] = ''

    for i in df.index:
        close = df["close"][i]
        date = df['date'][i]

        # Buy action
        if df["signal"][i] == 1:
```

```
            if(position == 0):
                buy_price = close
                position = 1
                df.at[i, 'buy_date'] = date
                #print(f"Buying at {str(buy_price)} on {str(date)}")

        # Sell action
        elif (df["signal"][i] == -1):
            if(position == 1):
                sell_price = close
                bought = 0
                position = 0
                df.at[i, 'sell_date'] = date
                #print(f"Selling at {str(sell_price)} on {str(date)}")

                # Get percentage change of trade
                pc = (sell_price/buy_price-1)*100
                percentage_change.append(pc)
                net_profit += (sell_price - buy_price)

    # Calculate trade statistics
    gains = 0
    ng = 0
    losses = 0
    nl = 0
    totalR = 1

    for i in percentage_change:
        if(i > 0):
            gains += i
            ng += 1
        else:
            losses += i
            nl += 1
        totalR = totalR * ((i/100)+1)

    totalR = round((totalR-1), 2)

    if(ng > 0):
        avgGain = round(gains/ng, 2)
        maxR = round(max(percentage_change), 2)
    else:
```

```
        avgGain = 0
        maxR = "undefined"

    if(nl > 0):
        avgLoss = round(losses/nl, 2)
        maxL = round(min(percentage_change), 2)
    else:
        avgLoss = 0
        maxL = "undefined"

    if(ng > 0 or nl > 0):
        win_rate = round((ng/(ng+nl))*100, 2)
    else:
        win_rate = 0

    return totalR
###############################################################

def monte_carlo_simulation(stock_dfs, num_simulations=20):

    simulation_results = []

    for sim in range(num_simulations):
        print(f"\nRunning simulation {sim + 1}/{num_simulations}...")

        # Generate random data
        synthetic_data = generate_synthetic_data(stock_dfs)

        # Apply the mean reversion strategy to generate trading signals
        simulated_data  = apply_mean_reversion_strategy(synthetic_data)

        # Backtest the strategy
        for ticker, df in simulated_data.items():
          total_return = backtest_dataframe_return_just_total_return(df)

        # Collect results
          simulation_results.append({
              'simulation': sim + 1,
              'ticker': ticker,
              'total_return': total_return
          })
```

```
    # Convert results to a DataFrame for analysis
    results_df = pd.DataFrame(simulation_results)

    return results_df

# Run the Monte Carlo simulation with 1000 simulations
# simulation_results = monte_carlo_simulation(stock_dfs)
print(monte_carlo_simulation(stock_dfs))
```

```
Running simulation 1/20...

Running simulation 2/20...

Running simulation 3/20...

Running simulation 4/20...

Running simulation 5/20...

Running simulation 6/20...

Running simulation 7/20...

Running simulation 8/20...

Running simulation 9/20...

Running simulation 10/20...

Running simulation 11/20...

Running simulation 12/20...

Running simulation 13/20...

Running simulation 14/20...

Running simulation 15/20...

Running simulation 16/20...

Running simulation 17/20...

Running simulation 18/20...

Running simulation 19/20...

Running simulation 20/20...
     simulation ticker  total_return
0             1    XOM          1.68
1             1    CVX          2.01
```

```
2               1    COP          0.01
3               1    EOG         -1.00
4               1    EPD          6.07
..            ...    ...           ...
195            20    WMB          0.00
196            20    OKE         -1.00
197            20    LNG          0.00
198            20    OXY         -1.00
199            20    HES         -1.00

[200 rows x 3 columns]
```

In [49]:
```
# download as html
```

In [ ]:
```
!find / -name "group_3_monte_carlo.ipynb"
```

```
find: ‘/proc/64/task/64/net’: Invalid argument
find: ‘/proc/64/net’: Invalid argument
```

In [50]:
```
from google.colab import files
!jupyter nbconvert --to html "//content/drive/My Drive/shared_folders/MSDS 460/Monte Carlo/monte_carlo_code
files.download("/content/group_3_monte_carlo.html")
```

```
[NbConvertApp] WARNING | pattern '/content/group_3_monte_carlo.ipynb' matched no files
This application is used to convert notebook files (*.ipynb)
        to various other formats.

        WARNING: THE COMMANDLINE INTERFACE MAY CHANGE IN FUTURE RELEASES.

Options
=======
The options below are convenience aliases to configurable class-options,
as listed in the "Equivalent to" description-line of the aliases.
To see all configurable class-options for some <cmd>, use:
    <cmd> --help-all

--debug
    set log level to logging.DEBUG (maximize logging output)
    Equivalent to: [--Application.log_level=10]
--show-config
    Show the application's configuration (human-readable format)
    Equivalent to: [--Application.show_config=True]
--show-config-json
    Show the application's configuration (json format)
    Equivalent to: [--Application.show_config_json=True]
--generate-config
    generate default config file
    Equivalent to: [--JupyterApp.generate_config=True]
-y
    Answer yes to any questions instead of prompting.
    Equivalent to: [--JupyterApp.answer_yes=True]
--execute
    Execute the notebook prior to export.
    Equivalent to: [--ExecutePreprocessor.enabled=True]
--allow-errors
    Continue notebook execution even if one of the cells throws an error and include the error message in t
he cell output (the default behaviour is to abort conversion). This flag is only relevant if '--execute' wa
s specified, too.
    Equivalent to: [--ExecutePreprocessor.allow_errors=True]
--stdin
    read a single notebook file from stdin. Write the resulting notebook with default basename 'notebook.*'
    Equivalent to: [--NbConvertApp.from_stdin=True]
--stdout
    Write notebook output to stdout instead of files.
    Equivalent to: [--NbConvertApp.writer_class=StdoutWriter]
```

```
--inplace
    Run nbconvert in place, overwriting the existing notebook (only
            relevant when converting to notebook format)
    Equivalent to: [--NbConvertApp.use_output_suffix=False --NbConvertApp.export_format=notebook --FilesWri
ter.build_directory=]
--clear-output
    Clear output of current file and save in place,
            overwriting the existing notebook.
    Equivalent to: [--NbConvertApp.use_output_suffix=False --NbConvertApp.export_format=notebook --FilesWri
ter.build_directory= --ClearOutputPreprocessor.enabled=True]
--coalesce-streams
    Coalesce consecutive stdout and stderr outputs into one stream (within each cell).
    Equivalent to: [--NbConvertApp.use_output_suffix=False --NbConvertApp.export_format=notebook --FilesWri
ter.build_directory= --CoalesceStreamsPreprocessor.enabled=True]
--no-prompt
    Exclude input and output prompts from converted document.
    Equivalent to: [--TemplateExporter.exclude_input_prompt=True --TemplateExporter.exclude_output_prompt=T
rue]
--no-input
    Exclude input cells and output prompts from converted document.
            This mode is ideal for generating code-free reports.
    Equivalent to: [--TemplateExporter.exclude_output_prompt=True --TemplateExporter.exclude_input=True --T
emplateExporter.exclude_input_prompt=True]
--allow-chromium-download
    Whether to allow downloading chromium if no suitable version is found on the system.
    Equivalent to: [--WebPDFExporter.allow_chromium_download=True]
--disable-chromium-sandbox
    Disable chromium security sandbox when converting to PDF..
    Equivalent to: [--WebPDFExporter.disable_sandbox=True]
--show-input
    Shows code input. This flag is only useful for dejavu users.
    Equivalent to: [--TemplateExporter.exclude_input=False]
--embed-images
    Embed the images as base64 dataurls in the output. This flag is only useful for the HTML/WebPDF/Slides
exports.
    Equivalent to: [--HTMLExporter.embed_images=True]
--sanitize-html
    Whether the HTML in Markdown cells and cell outputs should be sanitized..
    Equivalent to: [--HTMLExporter.sanitize_html=True]
--log-level=<Enum>
    Set the log level by value or name.
    Choices: any of [0, 10, 20, 30, 40, 50, 'DEBUG', 'INFO', 'WARN', 'ERROR', 'CRITICAL']
```

```
        Default: 30
        Equivalent to: [--Application.log_level]
    --config=<Unicode>
        Full path of a config file.
        Default: ''
        Equivalent to: [--JupyterApp.config_file]
    --to=<Unicode>
        The export format to be used, either one of the built-in formats
                ['asciidoc', 'custom', 'html', 'latex', 'markdown', 'notebook', 'pdf', 'python', 'qtpdf', 'qtpn
    g', 'rst', 'script', 'slides', 'webpdf']
                or a dotted object name that represents the import path for an
                ``Exporter`` class
        Default: ''
        Equivalent to: [--NbConvertApp.export_format]
    --template=<Unicode>
        Name of the template to use
        Default: ''
        Equivalent to: [--TemplateExporter.template_name]
    --template-file=<Unicode>
        Name of the template file to use
        Default: None
        Equivalent to: [--TemplateExporter.template_file]
    --theme=<Unicode>
        Template specific theme(e.g. the name of a JupyterLab CSS theme distributed
        as prebuilt extension for the lab template)
        Default: 'light'
        Equivalent to: [--HTMLExporter.theme]
    --sanitize_html=<Bool>
        Whether the HTML in Markdown cells and cell outputs should be sanitized.This
        should be set to True by nbviewer or similar tools.
        Default: False
        Equivalent to: [--HTMLExporter.sanitize_html]
    --writer=<DottedObjectName>
        Writer class used to write the
                                    results of the conversion
        Default: 'FilesWriter'
        Equivalent to: [--NbConvertApp.writer_class]
    --post=<DottedOrNone>
        PostProcessor class used to write the
                                    results of the conversion
        Default: ''
        Equivalent to: [--NbConvertApp.postprocessor_class]
```

```
--output=<Unicode>
    Overwrite base name use for output files.
                Supports pattern replacements '{notebook_name}'.
    Default: '{notebook_name}'
    Equivalent to: [--NbConvertApp.output_base]
--output-dir=<Unicode>
    Directory to write output(s) to. Defaults
                                to output to the directory of each notebook. To recover
                                previous default behaviour (outputting to the current
                                working directory) use . as the flag value.
    Default: ''
    Equivalent to: [--FilesWriter.build_directory]
--reveal-prefix=<Unicode>
    The URL prefix for reveal.js (version 3.x).
            This defaults to the reveal CDN, but can be any url pointing to a copy
            of reveal.js.
            For speaker notes to work, this must be a relative path to a local
            copy of reveal.js: e.g., "reveal.js".
            If a relative path is given, it must be a subdirectory of the
            current directory (from which the server is run).
            See the usage documentation
            (https://nbconvert.readthedocs.io/en/latest/usage.html#reveal-js-html-slideshow)
            for more details.
    Default: ''
    Equivalent to: [--SlidesExporter.reveal_url_prefix]
--nbformat=<Enum>
    The nbformat version to write.
            Use this to downgrade notebooks.
    Choices: any of [1, 2, 3, 4]
    Default: 4
    Equivalent to: [--NotebookExporter.nbformat_version]

Examples
--------

    The simplest way to use nbconvert is

            > jupyter nbconvert mynotebook.ipynb --to html

            Options include ['asciidoc', 'custom', 'html', 'latex', 'markdown', 'notebook', 'pdf', 'pytho
n', 'qtpdf', 'qtpng', 'rst', 'script', 'slides', 'webpdf'].
```

```
        > jupyter nbconvert --to latex mynotebook.ipynb

        Both HTML and LaTeX support multiple output templates. LaTeX includes
        'base', 'article' and 'report'.  HTML includes 'basic', 'lab' and
        'classic'. You can specify the flavor of the format used.

        > jupyter nbconvert --to html --template lab mynotebook.ipynb

        You can also pipe the output to stdout, rather than a file

        > jupyter nbconvert mynotebook.ipynb --stdout

        PDF is generated via latex

        > jupyter nbconvert mynotebook.ipynb --to pdf

        You can get (and serve) a Reveal.js-powered slideshow

        > jupyter nbconvert myslides.ipynb --to slides --post serve

        Multiple notebooks can be given at the command line in a couple of
        different ways:

        > jupyter nbconvert notebook*.ipynb
        > jupyter nbconvert notebook1.ipynb notebook2.ipynb

        or you can specify the notebooks list in a config file, containing::

            c.NbConvertApp.notebooks = ["my_notebook.ipynb"]

        > jupyter nbconvert --config mycfg.py

To see all available configurables, use `--help-all`.
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-50-578684d4172d> in <cell line: 0>()
      1 from google.colab import files
      2 get_ipython().system('jupyter nbconvert --to html "/content/group_3_monte_carlo.ipynb"')
----> 3 files.download("/content/group_3_monte_carlo.html")

/usr/local/lib/python3.11/dist-packages/google/colab/files.py in download(filename)
    231   if not _os.path.exists(filename):
    232     msg = 'Cannot find file: {}'.format(filename)
--> 233     raise FileNotFoundError(msg)  # pylint: disable=undefined-variable
    234 
    235   comm_manager = _IPython.get_ipython().kernel.comm_manager

FileNotFoundError: Cannot find file: /content/group_3_monte_carlo.html
```