```
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
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=Tr
```

```
# link to the monte carlo data folder https://drive.google.com/drive/folders/10GU2YP8ijheI8hR6IYnzU2F7A8b1Mgiu?usp=drive_link --

#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_separate_csv/'

#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_separate_csv/each_ticker_as_
```

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

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

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

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



# 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
```

```
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
```

```
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
```

```
# 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]
```

```
16.11740910042584
```

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

```
95.82491302490234
```

```
# https://www.investopedia.com/articles/basics/10/guide-to-calculating-roi.asp#:~:text=Return%20on%20investment%20(ROI)%20is%20a

# 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
```

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

```
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, 'EPD': 36.283392191
```

```
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
```

```
# 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
```

```
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
```

| Date | Close | High | Low | Open | Volume |
|---|---|---|---|---|---|
| 1999-01-04 | 77.577950 | 78.957288 | 76.750346 | 77.794703 | 9450400 |
| 1999-01-05 | 78.464668 | 78.740536 | 77.518836 | 77.518836 | 8031000 |
| 1999-01-06 | 80.356316 | 80.553364 | 79.292255 | 79.331664 | 7737700 |
| 1999-01-07 | 79.962227 | 80.218390 | 79.311967 | 79.686359 | 5504900 |
| 1999-01-08 | 80.553391 | 81.026307 | 79.430215 | 80.829258 | 6224400 |

```
spy_df.tail()
```

| Date | Close | High | Low | Open | Volume |
|---|---|---|---|---|---|
| 2023-12-22 | 467.651306 | 469.359407 | 465.726021 | 467.858638 | 67126600 |
| 2023-12-26 | 469.625977 | 470.544191 | 467.986997 | 468.066000 | 55387000 |
| 2023-12-27 | 470.475098 | 470.623192 | 468.875619 | 469.418642 | 68000300 |
| 2023-12-28 | 470.652802 | 471.501895 | 470.228255 | 470.840398 | 77158100 |
| 2023-12-29 | 469.290283 | 470.988501 | 467.305730 | 470.455331 | 122234100 |

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

```
<ipython-input-18-58fec3316e36>:1: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a future v
  spy_df['Open'][0]
77.79470274840207
```

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

```
<ipython-input-19-ee0949ca8fe6>:1: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a future v
  spy_df['Close'][-1]
469.290283203125
```

```
# 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]


sp500_return = (spy_close - spy_open) / spy_open
print(sp500_return)
```

```
5.032419517314299
```

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

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

#-------------------------------------        Part 2: Applies a moving average strategy using the moving average, bollinger and r

#-------------------------------------        uses a 200 day moving average, calculates the bollinger band (a band +/- 2 standara

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

#-------------------------------------        Sells when rsi is > 70 and price is > than the upper threshold, buys when rsi < 30

# starting with the first buy when the stock price and rsi move below the thresholds, we invest all of the amount (initially $10,


# 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-reversion/blob/main/Mean%20Rev

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


# applying the transformation from https://github.com/bryancwh/algo-trading-mean-reversion/blob/main/Mean%20Reversion.ipynb to e

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

# 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, bollinger_std = 2, rsi_

  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
```

```
# making dfs with various valus for testing

stock_dfs_original = apply_mean_reversion_strategy(stock_dfs, ma_period = 200, bollinger_period = 20, rsi_period = 6, bollinger_

stock_dfs_bollinger_period_20_rsi_period_14_rsilower_30_rsi_upper_70 = apply_mean_reversion_strategy(stock_dfs, ma_period = 200,

stock_dfs_bollinger_period_20_rsi_period_14_rsilower_20_rsi_upper_80 = apply_mean_reversion_strategy(stock_dfs, ma_period = 200,

stock_dfs_bollinger_period_30_rsi_period_6_rsilower_30_rsi_upper_70 = apply_mean_reversion_strategy(stock_dfs, ma_period = 200,

stock_dfs_bollinger_period_50_rsi_period_6_rsilower_30_rsi_upper_70 = apply_mean_reversion_strategy(stock_dfs, ma_period = 200,

stock_dfs_bollinger_period_20_not_using_rsi = apply_mean_reversion_strategy(stock_dfs, ma_period = 200, bollinger_period = 20, r

stock_dfs_bollinger_period_50_not_using_rsi = apply_mean_reversion_strategy(stock_dfs, ma_period = 200, bollinger_period = 50, r

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

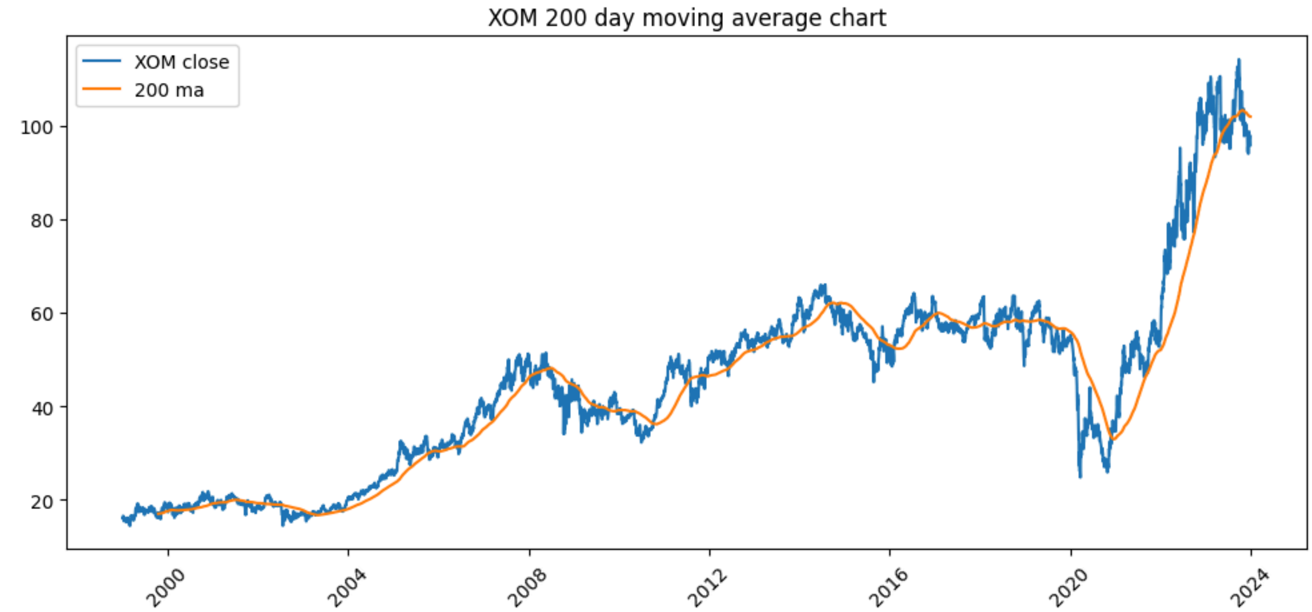
| | date | close | high | low | open | volume | ma_200 | ma_20 | std | upper_bollinger | lower_bollinger | de |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **500** | 2000-12-26 | 20.482729 | 20.540549 | 20.077989 | 20.077989 | 5580600 | 19.174942 | 20.226871 | 0.486072 | 21.199015 | 19.254726 | 0.404 |
| **501** | 2000-12-27 | 20.294815 | 20.656190 | 20.164720 | 20.540550 | 10437800 | 19.188477 | 20.165437 | 0.379542 | 20.924522 | 19.406353 | -0.187 |

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

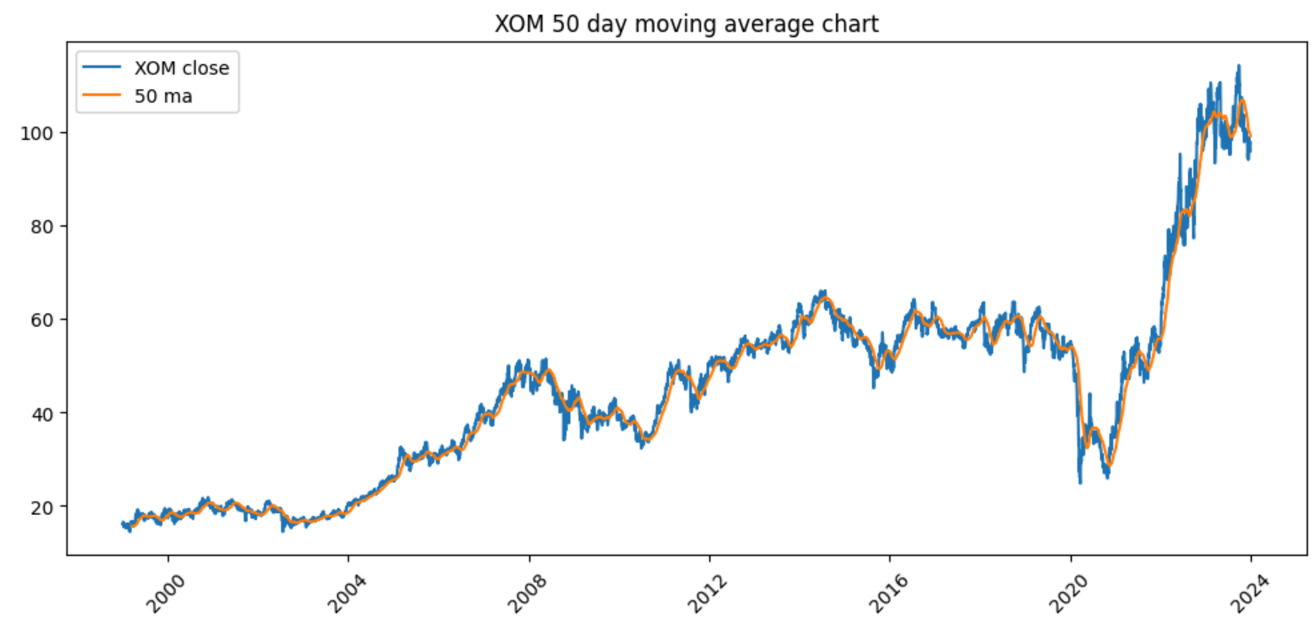
| | date | close | high | low | open | volume | ma_200 | ma_50 | std | upper_bollinger | lower_bollinger | de |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **500** | 2000-12-26 | 20.482729 | 20.540549 | 20.077989 | 20.077989 | 5580600 | 19.174942 | 20.484620 | 0.557019 | 21.598658 | 19.370581 | 0.404 |
| **501** | 2000-12-27 | 20.294815 | 20.656190 | 20.164720 | 20.540550 | 10437800 | 19.188477 | 20.486301 | 0.556302 | 21.598905 | 19.373698 | -0.187 |

```
#  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_using_rsi["XOM"]['close']
plt.plot(stock_dfs_bollinger_period_50_not_using_rsi["XOM"]['date'], stock_dfs_bollinger_period_50_not_using_rsi["XOM"]['ma_200'

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

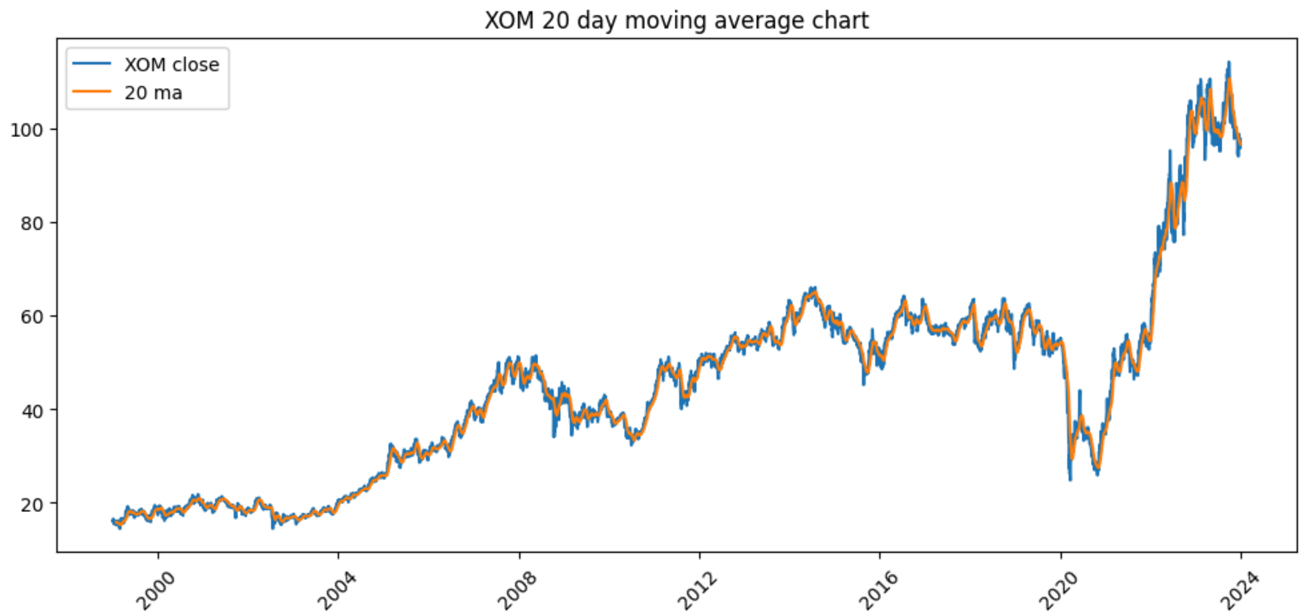



```
#  plot showing XOM 50 day moving avg


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_using_rsi["XOM"]['close']
plt.plot(stock_dfs_bollinger_period_50_not_using_rsi["XOM"]['date'], stock_dfs_bollinger_period_50_not_using_rsi["XOM"]['ma_50']

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

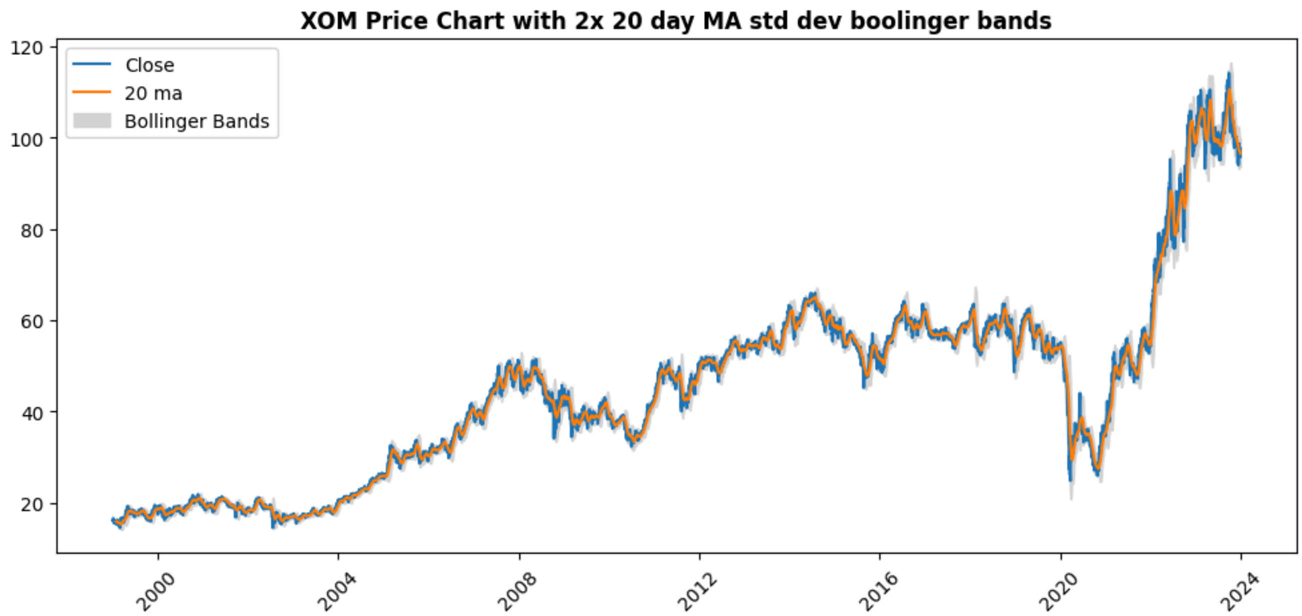



```
#  plot showing XOM 50 day moving avg
```

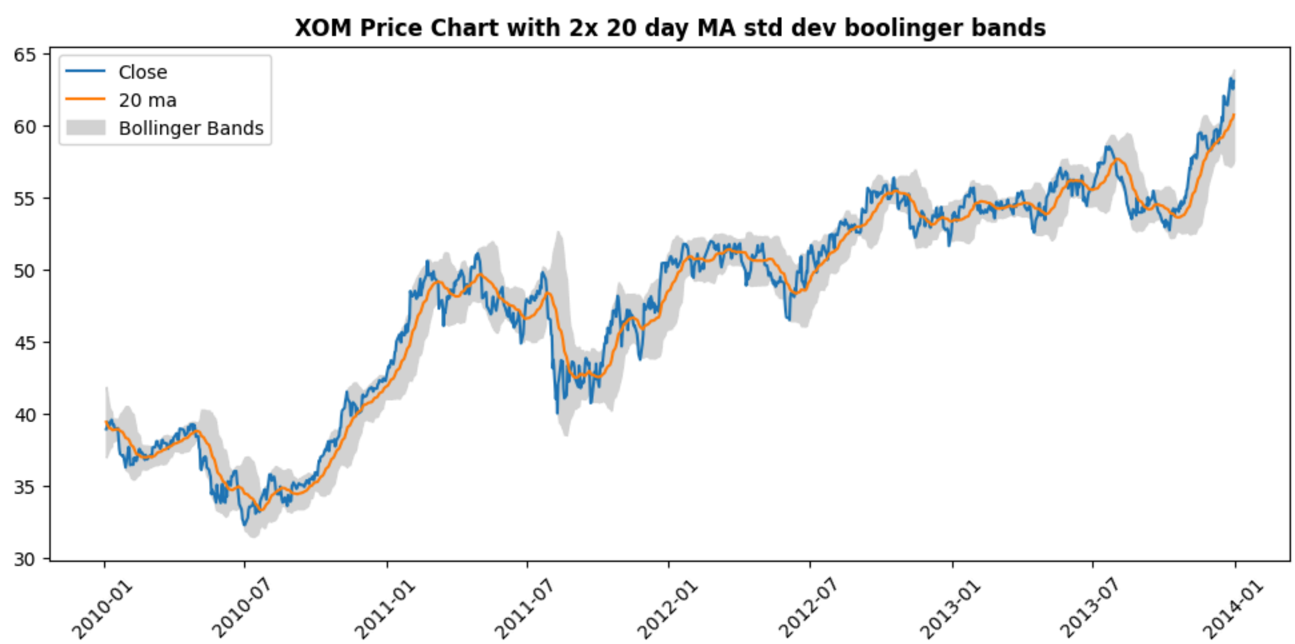
```
plt.figure(figsize=(12,5))
plt.xticks(rotation=45)

plt.plot(stock_dfs_bollinger_period_20_not_using_rsi["XOM"]['date'], stock_dfs_bollinger_period_20_not_using_rsi["XOM"]['close']
plt.plot(stock_dfs_bollinger_period_20_not_using_rsi["XOM"]['date'], stock_dfs_bollinger_period_20_not_using_rsi["XOM"]['ma_20']

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




```
# 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_using_rsi['XOM']['ma_20']
plt.fill_between(x_axis, stock_dfs_bollinger_period_20_not_using_rsi['XOM']['upper_bollinger'], stock_dfs_bollinger_period_20_nc

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




```
# # 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 = 'Bollinger Bands', color='l

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




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

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()

backtest_dataframe(stock_dfs_original["XOM"])
```

```
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
```

```
# same function as above but removed the print staements and made it just return the total return for the strategy

# 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
```

```
# 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'
```

```
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
```

```
# 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
```

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

#-------------------------------------        Part 2b: trying different values for moving average and rsi periods / rsi threshold

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


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,
}



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'
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'
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, 'OXY': 3.41, 'HES':
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':
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, 'OXY': 3.68, 'HES':
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'
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, 'OXY': 3.5, 'HES': 0
Total return on 100k: 219800.00
Overall return : 2.20
----------------------------------------------------
```

```
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['WMB']
```

```
df_best_stock_in_best_strategy.head()
```

| | date | close | high | low | open | volume | ma_200 | ma_20 | std | upper_bollinger | ... | delta | gain | loss | er |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1999-01-04 | 16.089716 | 16.422034 | 16.006637 | 16.117409 | 8853600 | NaN | NaN | NaN | NaN | ... | NaN | NaN | NaN | |
| 1 | 1999-01-05 | 15.951257 | 16.158955 | 15.882024 | 16.062029 | 6652800 | NaN | NaN | NaN | NaN | ... | -0.138459 | 0.000000 | 0.138459 | ( |
| 2 | 1999-01-06 | 16.588202 | 16.795900 | 16.103571 | 16.200497 | 9965600 | NaN | NaN | NaN | NaN | ... | 0.636945 | 0.636945 | 0.000000 | ( |
| 3 | 1999-01-07 | 16.560503 | 16.602043 | 16.338958 | 16.491270 | 7417200 | NaN | NaN | NaN | NaN | ... | -0.027699 | 0.000000 | 0.027699 | ( |
| 4 | 1999-01-08 | 16.463579 | 16.546659 | 16.158955 | 16.449733 | 6343400 | NaN | NaN | NaN | NaN | ... | -0.096924 | 0.000000 | 0.096924 | ( |

5 rows × 21 columns

```
df_worst_stock_in_best_strategy.head()
```

| | date | close | high | low | open | volume | ma_200 | ma_20 | std | upper_bollinger | ... | delta | gain | loss | ema_ga |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1999-01-04 | 8.717077 | 8.969746 | 8.608791 | 8.951699 | 2419311 | NaN | NaN | NaN | NaN | ... | NaN | NaN | NaN | N |
| 1 | 1999-01-05 | 8.626838 | 8.771220 | 8.590742 | 8.662933 | 2414634 | NaN | NaN | NaN | NaN | ... | -0.090240 | 0.000000 | 0.09024 | 0.0000 |
| 2 | 1999-01-06 | 8.897552 | 8.933647 | 8.753170 | 8.807313 | 2392051 | NaN | NaN | NaN | NaN | ... | 0.270714 | 0.270714 | 0.00000 | 0.0360 |
| 3 | 1999-01-07 | 8.897552 | 8.933647 | 8.680978 | 8.771217 | 1840173 | NaN | NaN | NaN | NaN | ... | 0.000000 | 0.000000 | 0.00000 | 0.0312 |
| 4 | 1999-01-08 | 8.897552 | 8.987790 | 8.771217 | 8.879504 | 1381299 | NaN | NaN | NaN | NaN | ... | 0.000000 | 0.000000 | 0.00000 | 0.0271 |

5 rows × 21 columns

```
# 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'], df_best_stock_in_best_st
plt.scatter(df_best_stock_in_best_strategy[(df_best_stock_in_best_strategy['signal'] == -1)]['sell_date'], df_best_stock_in_best_

plt.title('XOM Price & Trades Using the Best Strategy: Bollinger 20 day MA / RSI 14 day MA / 20-80 RSI Threshold', fontweight="bo
plt.legend()
plt.show()
```


```
# 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'], df_worst_stock_in_best_
plt.scatter(df_worst_stock_in_best_strategy[(df_worst_stock_in_best_strategy['signal'] == -1)]['sell_date'], df_worst_stock_in_be

plt.title('WMB Price & Trades Using the Best Strategy: Bollinger 20 day MA / RSI 14 day MA / 20-80 RSI Threshold', fontweight="bo
plt.legend()
plt.show()
```


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

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


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 pct change I don't thir
        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_skew:.4f}")

        # 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 < 20 to match best histo
            1, np.nan
        )

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