import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

sns.set\_style('whitegrid')

plt.style.use("fivethirtyeight")

from pandas\_datareader.data import DataReader

from datetime import datetime

from math import sqrt

from sklearn.metrics import mean\_squared\_error

from sklearn.preprocessing import MinMaxScaler

# Ignore warnings

import warnings

warnings.filterwarnings('ignore')

KO\_Data = pd.read\_csv(r"C:/Users/kkhus/Downloads/Coca-Cola\_stock\_history*.*csv")

KO\_Data.plot(*subplots* = True, *figsize* = (10,12))

plt.title('Coca Cola Stock Attributes')

plt.show()

def plot\_close\_val(*data\_frame*, *column*, *stock*):

    plt.figure(*figsize*=(16,6))

    plt.title(*column* + ' Price History for ' + *stock* )

    plt.plot(*data\_frame*[*column*])

    plt.xlabel('Date', *fontsize*=18)

    plt.ylabel(*column* + ' Price USD ($) for ' + *stock*,

*fontsize*=18)

    plt.show()

#Test the function

plot\_close\_val(KO\_Data, 'Close', 'Coca Cola')

plot\_close\_val(KO\_Data, 'Open', 'Coca Cola')

KO\_Data[["Volume"]].plot()

ko\_info = pd.read\_csv(r"C:/Users/kkhus/Downloads/Coca-Cola\_stock\_info*.*csv",

*header*=None,

*names*=(['Description','Information']))

ko\_info.dropna()

ko\_info.drop(ko\_info.loc[ko\_info['Information']=='nan'].index,

*inplace*=True)

ko = ko\_info.sort\_values('Information').style

ko

KO\_Data['Date'] = pd.to\_datetime(KO\_Data['Date'], *errors*='coerce', *infer\_datetime\_format*=True)

start\_price = KO\_Data['Close'].iloc[0]

end\_price = KO\_Data['Close'].iloc[-1]

print(KO\_Data[KO\_Data['Date'].isna()])

# Number of years (based on your date column)

years = (KO\_Data['Date'].iloc[-1] - KO\_Data['Date'].iloc[0]).days / 365

cagr = (end\_price / start\_price) \*\* (1/years) - 1

print(f"CAGR: {cagr\*100:.2f}%")

adj\_close\_px = KO\_Data['Close']

moving\_avg = adj\_close\_px.rolling(*window*=40).mean()

moving\_avg[-10:]

KO\_Data['42'] = adj\_close\_px.rolling(*window*=40).mean()

KO\_Data['252'] = adj\_close\_px.rolling(*window*=252).mean()

KO\_Data[['Close', '42', '252']].plot()

plt.show()

daily\_close\_px = KO\_Data[['Close']]

daily\_pct\_change = daily\_close\_px.pct\_change()

daily\_pct\_change.hist(*bins*=50, *sharex*=True, *figsize*=(12,8))

plt.show()

min\_periods = 75

vol = daily\_pct\_change.rolling(min\_periods).std() \* np.sqrt(min\_periods)

vol.plot(*figsize*=(10, 8))

plt.show()

pd.plotting.scatter\_matrix(daily\_pct\_change, *diagonal*='kde',

*alpha*=0.1,*figsize*=(12,12))

plt.show()

import plotly.graph\_objects as go

KO\_Data=KO\_Data.reset\_index()

fig = go.Figure(*data*=go.Ohlc(*x*=KO\_Data['Date'],

*open*=KO\_Data['Open'],

*high*=KO\_Data['High'],

*low*=KO\_Data['Low'],

*close*=KO\_Data['Close']))

fig.show()

KO\_Data['SMA5'] = KO\_Data.Close.rolling(5).mean()

KO\_Data['SMA20'] = KO\_Data.Close.rolling(20).mean()

KO\_Data['SMA50'] = KO\_Data.Close.rolling(50).mean()

KO\_Data['SMA200'] = KO\_Data.Close.rolling(200).mean()

KO\_Data['SMA500'] = KO\_Data.Close.rolling(500).mean()

fig = go.Figure(*data*=[go.Ohlc(*x*=KO\_Data['Date'],*open*=KO\_Data['Open'],

*high*=KO\_Data['High'],*low*=KO\_Data['Low'],*close*=KO\_Data['Close'],

*name* = "OHLC"),

go.Scatter(*x*=KO\_Data.Date,

*y*=KO\_Data.SMA5, *line*=dict(*color*='orange', *width*=1),

*name*="SMA5"),

go.Scatter(*x*=KO\_Data.Date,

*y*=KO\_Data.SMA20, *line*=dict(*color*='green', *width*=1),

*name*="SMA20"),

go.Scatter(*x*=KO\_Data.Date,

*y*=KO\_Data.SMA50, *line*=dict(*color*='blue', *width*=1),

*name*="SMA50"),

go.Scatter(*x*=KO\_Data.Date,

*y*=KO\_Data.SMA200, *line*=dict(*color*='violet', *width*=1),

*name*="SMA200"),

go.Scatter(*x*=KO\_Data.Date,

*y*=KO\_Data.SMA500, *line*=dict(*color*='purple', *width*=1),

*name*="SMA500")])

fig.show()

KO\_Data['EMA5'] = KO\_Data.Close.ewm(*span*=5,

*adjust*=False).mean()

KO\_Data['EMA20'] = KO\_Data.Close.ewm(*span*=20,

*adjust*=False).mean()

KO\_Data['EMA50'] = KO\_Data.Close.ewm(*span*=50,

*adjust*=False).mean()

KO\_Data['EMA200'] = KO\_Data.Close.ewm(*span*=200,

*adjust*=False).mean()

KO\_Data['EMA500'] = KO\_Data.Close.ewm(*span*=500,

*adjust*=False).mean()

fig = go.Figure(*data*=[go.Ohlc(*x*=KO\_Data['Date'],

*open*=KO\_Data['Open'],

*high*=KO\_Data['High'],

*low*=KO\_Data['Low'],

*close*=KO\_Data['Close'], *name* =

"OHLC"),

go.Scatter(*x*=KO\_Data.Date,

*y*=KO\_Data.SMA5, *line*=dict(*color*='orange', *width*=1),

*name*="EMA5"),

go.Scatter(*x*=KO\_Data.Date,

*y*=KO\_Data.SMA20, *line*=dict(*color*='green', *width*=1),

*name*="EMA20"),

go.Scatter(*x*=KO\_Data.Date,

*y*=KO\_Data.SMA50, *line*=dict(*color*='blue', *width*=1),

*name*="EMA50"),

go.Scatter(*x*=KO\_Data.Date,

*y*=KO\_Data.SMA200, *line*=dict(*color*='violet', *width*=1),

*name*="EMA200"),

go.Scatter(*x*=KO\_Data.Date,

*y*=KO\_Data.SMA500, *line*=dict(*color*='purple', *width*=1),

*name*="EMA500")])

fig.show()

KO\_Data.head()

try:

    from finta import TA

    from backtesting import Backtest, Strategy

    from backtesting.lib import crossover

except ImportError:

    import subprocess

    import sys

    subprocess.check\_call([sys.executable, "-m", "pip", "install", "finta", "backtesting"])

    # Now import again

    from finta import TA

    from backtesting import Backtest, Strategy

    from backtesting.lib import crossover

fin\_ma = pd.read\_csv(r'C:\Users\kkhus*\D*ownloads\Coca-Cola\_stock\_history*.*csv', *parse\_dates*=True)

print(fin\_ma.head())

ohlc=fin\_ma

print(TA.SMA(ohlc, 42))

function\_dict = {' Simple Moving Average ' : 'SMA',

' Simple Moving Median ' : 'SMM',

' Smoothed Simple Moving Average ' : 'SSMA',

' Exponential Moving Average ' : 'EMA',

' Double Exponential Moving Average ' :

'DEMA', ' Triple Exponential Moving Average ' :

'TEMA',

' Triangular Moving Average ' : 'TRIMA',

' Triple Exponential Moving Average Oscillator ' : 'TRIX',

' Volume Adjusted Moving Average ' : 'VAMA',

' Kaufman Efficiency Indicator ' : 'ER',

' Kaufmans Adaptive Moving Average ' : 'KAMA',

' Zero Lag Exponential Moving Average ' :

'ZLEMA',

' Weighted Moving Average ' : 'WMA',

' Hull Moving Average ' : 'HMA',

' Elastic Volume Moving Average ' : 'EVWMA',

' Volume Weighted Average Price ' : 'VWAP',

' Smoothed Moving Average ' : 'SMMA',

' Fractal Adaptive Moving Average ' : 'FRAMA',

' Moving Average Convergence Divergence ' :

'MACD', ' Percentage Price Oscillator ' : 'PPO',

' Volume-Weighted MACD ' : 'VW\_MACD',

' Elastic-Volume weighted MACD ' : 'EV\_MACD',

' Market Momentum ' : 'MOM',

' Rate-of-Change ' : 'ROC',

' Relative Strength Index ' : 'RSI',

' Inverse Fisher Transform RSI ' : 'IFT\_RSI',

' True Range ' : 'TR',

' Average True Range ' : 'ATR',

' Stop-and-Reverse ' : 'SAR',' Bollinger Bands ' : 'BBANDS',

' Bollinger Bands Width ' : 'BBWIDTH',

' Momentum Breakout Bands ' : 'MOBO',

' Percent B ' : 'PERCENT\_B',

' Keltner Channels ' : 'KC',

' Donchian Channel ' : 'DO',

' Directional Movement Indicator ' : 'DMI',

' Average Directional Index ' : 'ADX',

' Pivot Points ' : 'PIVOT', ' Fibonacci Pivot Points ' : 'PIVOT\_FIB',

' Stochastic Oscillator Percent K ' : 'STOCH', ' Stochastic oscillator Percent D ' :

'STOCHD',

' Stochastic RSI ' : 'STOCHRSI',

' Williams Percent R ' : 'WILLIAMS',

' Ultimate Oscillator ' : 'UO',

' Awesome Oscillator ' : 'AO',

' Mass Index ' : 'MI', ' Know Sure Thing ' : 'KST',

' True Strength Index ' : 'TSI',

' Typical Price ' : 'TP',

' Accumulation-Distribution Line ' : 'ADL',

' Chaikin Oscillator ' : 'CHAIKIN',

' Money Flow Index ' : 'MFI',

' On Balance Volume ' : 'OBV',

' Weighter OBV ' : 'WOBV', ' Volume Zone Oscillator ' : 'VZO',

' Price Zone Oscillator ' : 'PZO',

' Elders Force Index ' : 'EFI',

' Cummulative Force Index ' : 'CFI',

' Bull power and Bear Power ' : 'EBBP',

' Ease of Movement ' : 'EMV',

' Commodity Channel Index ' : 'CCI',

' Coppock Curve ' : 'COPP',

' Buy and Sell Pressure ' : 'BASP', ' Normalized BASP ' : 'BASPN',

' Chande Momentum Oscillator ' : 'CMO',

' Chandelier Exit ' : 'CHANDELIER',

' Qstick ' : 'QSTICK',

#' Twiggs Money Index ' : 'TMF',

' Wave Trend Oscillator ' : 'WTO',

' Fisher Transform ' : 'FISH',

' Ichimoku Cloud ' : 'ICHIMOKU',

' Adaptive Price Zone ' : 'APZ',

#' Squeeze Momentum Indicator ' : 'SQZMI',

' Volume Price Trend ' : 'VPT',

' Finite Volume Element ' : 'FVE',

' Volume Flow Indicator ' : 'VFI',

' Moving Standard deviation ' : 'MSD',

' Schaff Trend Cycle ' : 'STC'}

for key, value in function\_dict .items():

    function\_name = ("TA." + value + "(ohlc).plot(title='" + key

    + "for Coca Cola / Coke Stock')")

    result = eval(function\_name)

class DemaCross(Strategy):

    def init(*self*):

*self*.ma1 = *self*.I(TA.DEMA, ohlc, 10)

*self*.ma2 = *self*.I(TA.DEMA, ohlc, 20)

    def next(*self*):

        if crossover(*self*.ma1, *self*.ma2):

*self*.buy()

        elif crossover(*self*.ma2, *self*.ma1):

*self*.sell()

ohlc.head()

print(ohlc.Date)

bt = Backtest(ohlc, DemaCross,

*cash*=100000, *commission*=0.015,

*exclusive\_orders*=True)

bt.run()

bt.plot()

data=ohlc

from backtesting import Strategy

from backtesting.lib import crossover

from backtesting.test import SMA

def BBANDS(*data*, *n\_lookback*, *n\_std*):

    """Bollinger bands indicator"""

    hlc3 = (*data*.High + *data*.Low + *data*.Close) / 3

    mean, std = hlc3.rolling(*n\_lookback*).mean(),

    hlc3.rolling(*n\_lookback*).std()

    upper = mean + *n\_std*\*std

    lower = mean - *n\_std*\*std

    return upper, lower

close = data.Close.values

sma10 = SMA(data.Close, 10)

sma20 = SMA(data.Close, 20)

sma50 = SMA(data.Close, 50)

sma100 = SMA(data.Close, 100)

upper, lower = BBANDS(data, 20, 2)

data['X\_SMA10'] = (close - sma10) / close

data['X\_SMA20'] = (close - sma20) / close

data['X\_SMA50'] = (close - sma50) / close

data['X\_SMA100'] = (close - sma100) / close

data['X\_DELTA\_SMA10'] = (sma10 - sma20) / close

data['X\_DELTA\_SMA20'] = (sma20 - sma50) / close

data['X\_DELTA\_SMA50'] = (sma50 - sma100) / close

data['X\_MOM'] = data.Close.pct\_change(*periods*=2)

data['X\_BB\_upper'] = (upper - close) / close

data['X\_BB\_lower'] = (lower - close) / close

data['X\_BB\_width'] = (upper - lower) / close

#data['X\_Sentiment'] =

~data.index.to\_series().between('2017-09-27', '2017-12-14')

class Sma4Cross(Strategy):

    n1 = 50

    n2 = 100

    n\_enter = 20

    n\_exit = 10

    def init(*self*):

*self*.sma1 = *self*.I(SMA, *self*.data.Close, *self*.n1)

*self*.sma2 = *self*.I(SMA, *self*.data.Close, *self*.n2)

*self*.sma\_enter = *self*.I(SMA, *self*.data.Close,

*self*.n\_enter)

*self*.sma\_exit = *self*.I(SMA, *self*.data.Close,

*self*.n\_exit)

    def next(*self*):

        if not *self*.position:

            if *self*.sma1 > *self*.sma2:

                if crossover(*self*.data.Close, *self*.sma\_enter):

*self*.buy()

                else:

                    if crossover(*self*.sma\_enter, *self*.data.Close):

*self*.sell()

                    else:

                        if (*self*.position.is\_long and

                            crossover(*self*.sma\_exit, *self*.data.Close)

                            or

*self*.position.is\_short and

                        crossover(*self*.data.Close, *self*.sma\_exit)):

*self*.position.close()

%%time

from backtesting import Backtest

from backtesting.test import GOOG

backtest = Backtest(ohlc, Sma4Cross, *commission*=.002)

stats, heatmap = backtest.optimize(

*n1*=range(10, 110, 10),

*n2*=range(20, 210, 20),

*n\_enter*=range(15, 35, 5),

*n\_exit*=range(10, 25, 5),

*constraint*=lambda *p*: *p*.n\_exit < *p*.n\_enter < *p*.n1 < *p*.n2,

*maximize*='Equity Final [$]',

*max\_tries*=200,

*random\_state*=0,

*return\_heatmap*=True)

heatmap

hm = heatmap.groupby(['n1', 'n2']).mean().unstack()

hm

from backtesting.lib import plot\_heatmaps

plot\_heatmaps(heatmap, *agg*='mean')

%%capture

%%time

stats\_skopt, heatmap, optimize\_result = backtest.optimize(

*n1*=[10, 100], # Note: For method="skopt", we

*n2*=[20, 200], # only need interval end-points

*n\_enter*=[10, 40],

*n\_exit*=[10, 30],

*constraint*=lambda *p*: *p*.n\_exit < *p*.n\_enter < *p*.n1 < *p*.n2,

*maximize*='Equity Final [$]',

*method*='skopt',

*max\_tries*=200,

*random\_state*=0,

*return\_heatmap*=True,

*return\_optimization*=True)

from skopt.plots import plot\_objective

\_ = plot\_objective(optimize\_result, *n\_points*=10)