```
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\Downloads\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)
```