Project Title	**Coca Cola Stock - Live and Updated**
Tools	ML, Python, SQL, Excel
Domain	Data Analyst,
Project Difficulties level	intermediate

Dataset : Dataset is available in the given link. You can download it at your convenience.

Click here to download data set

About Dataset

The Coca-Cola Company is an North American multinational beverage corporation incorporated under Delaware's General Corporation Law[a] and headquartered in Atlanta, Georgia. The Coca-Cola Company has interests in the manufacturing, retailing, and marketing of non-alcoholic beverage concentrates and syrups, and alcoholic beverages. The company produces Coca-Cola, the sugary drink for which it is best known for, invented in 1886 by pharmacist John Stith Pemberton. At the time, the product was made with coca leaves, which added an amount of cocaine to the drink, and with kola nuts, which added caffeine, so that the coca and the kola together provided a stimulative effect. This stimulative effect is the reason the drink was sold to the public as a healthy "tonic", and the coca and the kola are also the source of the name of the product and of the company.In 1889, the formula and brand were sold for

$2,300 (roughly $68,000 in 2021) to Asa Griggs Candler, who incorporated The Coca-Cola Company in Atlanta in 1892.

Since 1919, Coca-Cola has been a publicly traded company. Its stock is listed on the New York Stock Exchange under the ticker symbol "KO". One share of stock purchased in 1919 for $40, with all dividends reinvested, would have been worth $9.8 million in 2012, a 10.7% annual increase adjusted for inflation. A predecessor bank of SunTrust received $100,000 for underwriting Coca-Cola's 1919 public offering; the bank sold that stock for over $2 billion in 2012. In 1987, Coca-Cola once again became one of the 30 stocks which makes up the Dow Jones Industrial Average, which is commonly referenced as a proxy for stock market performance; it had previously been a Dow stock from 1932 to 1935. Coca-Cola has paid a dividend since 1920 and, as of 2019, had increased it each year for 57 years straight.

Example: You can get the basic idea how you can create a project from here

To create a major ML project using Coca-Cola stock data with the specified columns, here's a structured step-by-step guide. This explanation will include **EDA**, **data cleaning**, **data visualization**, **handling missing values**, **statistical operations**, and more. The project is tailored for an experienced developer.

Step 1: Problem Definition

- **Objective**: Predict Coca-Cola's stock prices (e.g., Close price) and analyze trends.
- **Data**: Historical data with `Date, Open, High, Low, Close, Volume, Dividends, Stock Splits`.
- **Deliverables**:

1. Insights from the data (visualizations and statistics).
2. An ML model to predict stock prices.
3. A live-updating system for predictions.

Step 2: Data Collection

Use **Yahoo Finance API** for historical data. We'll fetch data from 2015 to the present.

Code:

python

code

```
import yfinance as yf
import pandas as pd

# Fetch Coca-Cola stock data
ticker = 'KO'  # Coca-Cola stock ticker
data = yf.download(ticker, start='2015-01-01',
end='2023-12-31')

# Reset index for easier handling
data.reset_index(inplace=True)

# Display data structure
print(data.info())
print(data.head())
```

Step 3: Data Cleaning

3.1 Handle Missing Values

- Check for missing values and decide on imputation or deletion.

Code:

python

code

```
# Check for missing values
print(data.isnull().sum())

# Fill missing numerical values with the column mean
data.fillna(method='ffill', inplace=True)  # Forward fill for stock data continuity
data.fillna(0, inplace=True)  # Replace remaining missing dividends/splits with 0

# Confirm no missing values remain
print(data.isnull().sum())
```

Step 4: Feature Engineering

1. **Add Moving Averages**: 20-day and 50-day for trend detection.
2. **Add Daily Returns**: Helps capture volatility.

3. **Add Volatility**: Standard deviation over a rolling window.

Code:

python

code

```
# Add Moving Averages
data['MA_20'] = data['Close'].rolling(window=20).mean()
data['MA_50'] = data['Close'].rolling(window=50).mean()

# Add Daily Returns
data['Daily_Return'] = data['Close'].pct_change()

# Add Volatility (standard deviation of returns over a rolling
window)
data['Volatility'] =
data['Daily_Return'].rolling(window=20).std()

# Drop rows with NA due to rolling calculations
data.dropna(inplace=True)

print(data.head())
```

Step 5: Exploratory Data Analysis (EDA)

5.1 Summary Statistics

Use descriptive statistics to summarize the data.

Code:

python

code

```
# Summary statistics
print(data.describe())
```

5.2 Data Visualization

Visualize the trends and relationships in the data.

Code:

python

code

```
import matplotlib.pyplot as plt
import seaborn as sns

# Line plot for stock prices
plt.figure(figsize=(12, 6))
plt.plot(data['Date'], data['Close'], label='Close Price')
plt.plot(data['Date'], data['MA_20'], label='MA 20',
linestyle='--')
plt.plot(data['Date'], data['MA_50'], label='MA 50',
linestyle='--')
plt.title('Coca-Cola Stock Prices with Moving Averages')
plt.xlabel('Date')
```

```
plt.ylabel('Price')
plt.legend()
plt.show()

# Correlation heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(data.corr(), annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.show()
```

Step 6: Data Splitting

Split the data into training and testing sets for model training.

Code:

python

code

```
from sklearn.model_selection import train_test_split

# Features and target
features = ['Open', 'High', 'Low', 'Volume', 'Dividends',
'Stock Splits', 'MA_20', 'MA_50', 'Daily_Return', 'Volatility']
target = 'Close'

X = data[features]
```

```
y = data[target]

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42, shuffle=False)
```

Step 7: Model Training

7.1 Use Random Forest for Initial Predictions

A good baseline model for tabular data is Random Forest.

Code:

python
code

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error,
mean_squared_error

# Initialize the model
model = RandomForestRegressor(n_estimators=100,
random_state=42)

# Train the model
model.fit(X_train, y_train)
```

```
# Predict on test set
y_pred = model.predict(X_test)

# Evaluate model
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)

print(f"Mean Squared Error: {mse}")
print(f"Mean Absolute Error: {mae}")
```

Step 8: Live Prediction System

Integrate a system to fetch live data and make predictions.

Fetching Live Data:

python
code

```
# Fetch latest stock data
live_data = yf.download(ticker, period='1d', interval='1m')

# Prepare live data for prediction
live_data['MA_20'] =
live_data['Close'].rolling(window=20).mean()
live_data['MA_50'] =
live_data['Close'].rolling(window=50).mean()
```

```
live_data['Daily_Return'] = live_data['Close'].pct_change()
live_data['Volatility'] =
live_data['Daily_Return'].rolling(window=20).std()

# Ensure no missing values
live_data.fillna(0, inplace=True)

# Use the latest data point for prediction
latest_features = live_data[features].iloc[-1:].dropna()
live_prediction = model.predict(latest_features)

print(f"Predicted Closing Price: {live_prediction[0]}")
```

Step 9: Deploy the System

Deploy using **Streamlit** or **Flask** for a web-based dashboard. Example with Streamlit:

Code:

python

code

```
import streamlit as st

st.title('Coca-Cola Stock Price Prediction')

# Upload visualization
```

```
st.line_chart(data[['Close', 'MA_20', 'MA_50']])


# Show prediction
st.write(f"Predicted Closing Price: {live_prediction[0]}")
```

Sample Code and output

Stock Analysis of Coca Cola Stock¶

1. Import Libraries

In [2]:

```
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")
%matplotlib inline

# For reading stock data from yahoo
from pandas_datareader.data import DataReader

# For time stamps
from datetime import datetime
from math import sqrt
from math import sqrt
```

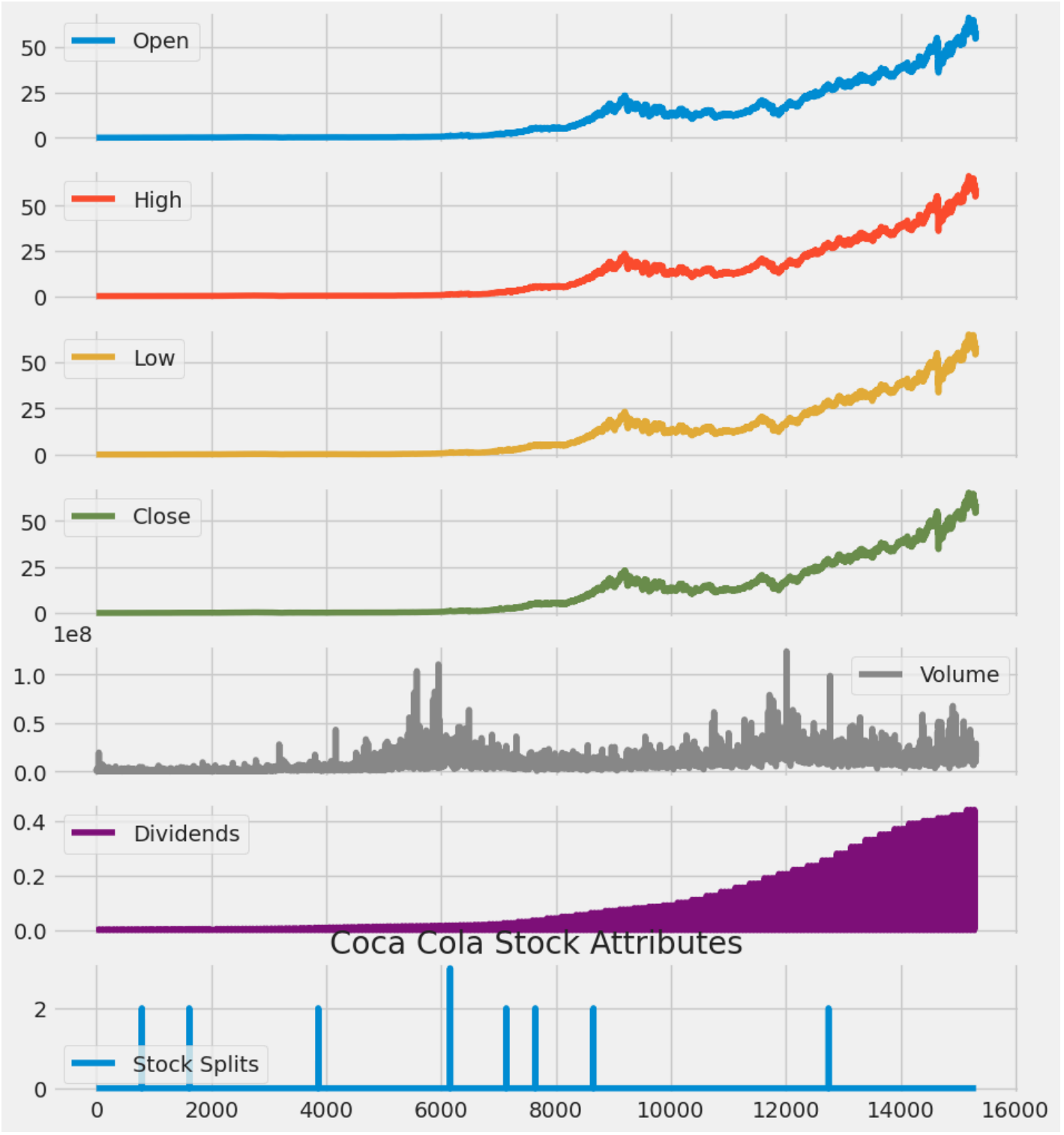
```
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import MinMaxScaler


#ignore the warnings
import warnings
warnings.filterwarnings('ignore')
```

2. Load Dataset

```
In [3]:
KO_Data =
pd.read_csv('../input/coca-cola-stock-live-and-updated/Coca-Col
a_stock_history.csv')
```

3. Basic EDA

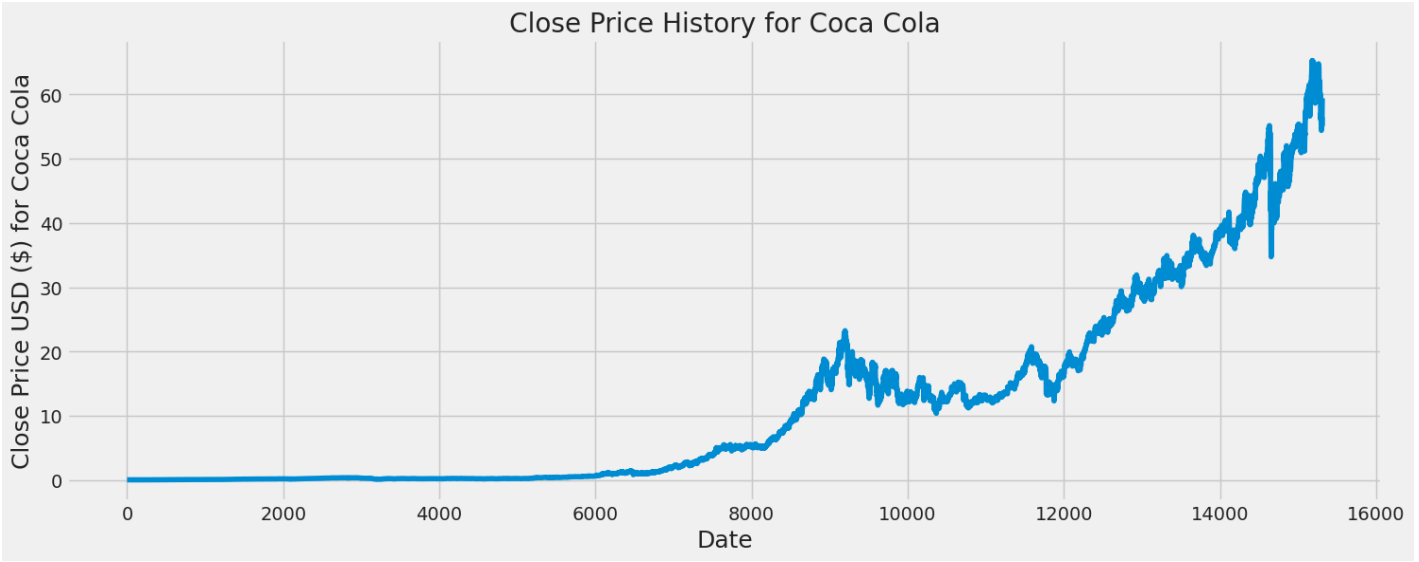
```
In [4]:
KO_Data.plot(subplots = True, figsize = (10,12))
plt.title('Coca Cola Stock Attributes')
plt.show()
```

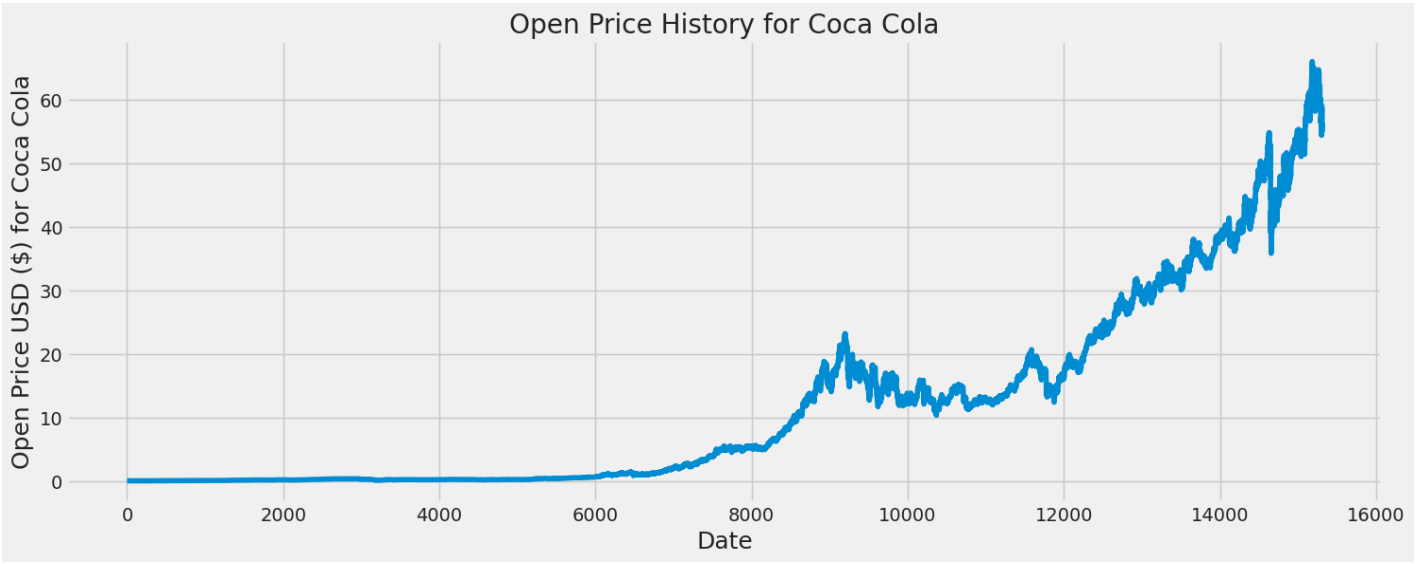


In [5]:

```
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')
```



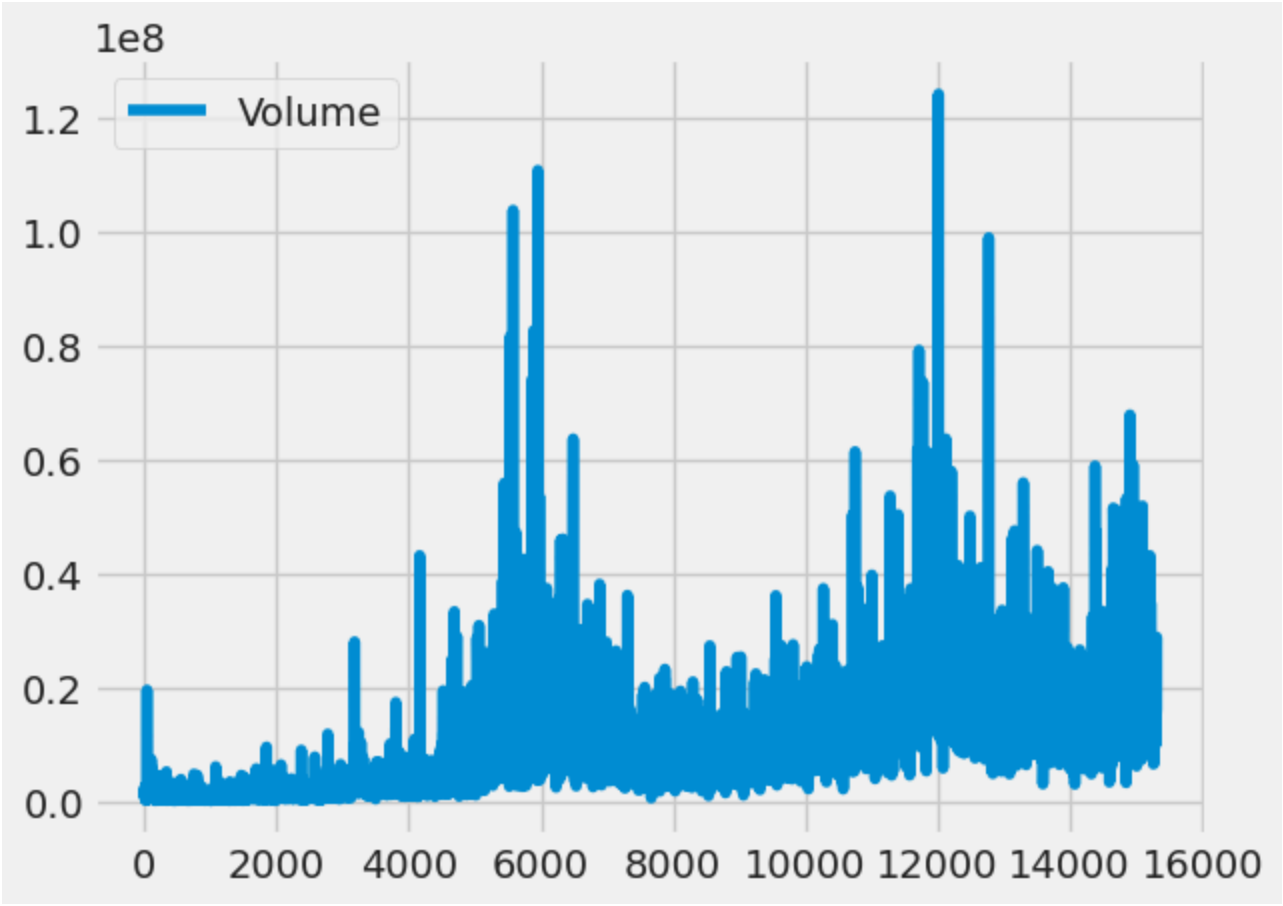


In [6]:

```
KO_Data[["Volume"]].plot()
```

Out[6]:

<Axes: >



4. Basic Company Info

In [7]:

```
ko_info =
pd.read_csv('../input/coca-cola-stock-live-and-updated/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

Out[7]:

	Description	Information
49	gmtOffSetMilliseconds	-18000000
145	bid	0
138	ask	0
77	heldPercentInsiders	0.00636

68	sharesPercentSharesOut	0.0074
100	shortPercentOfFloat	0.0074
108	trailingAnnualDividendYield	0.027917083
147	dividendYield	0.028099999
29	returnOnAssets	0.07831
75	SandP52WeekChange	0.15025425

18	revenueGrowth	0.161
58	52WeekChange	0.21709049
15	profitMargins	0.23313999
19	operatingMargins	0.31123
14	ebitdaMargins	0.35199
33	returnOnEquity	0.39722002
74	lastDividendValue	0.42

93	earningsQuarterlyGrowth	0.423
27	earningsGrowth	0.425
16	grossMargins	0.60723996
71	heldPercentInstitutions	0.7005
84	beta	0.712113
109	payoutRatio	0.82269996
1	maxAge	1

1		
41	quickRatio	1.173
28	currentRatio	1.516
81	shortRatio	1.6
116	trailingAnnualDividendRate	1.67
122	dividendRate	1.68
13	askSize	1000

9		
76	priceToBook	11.606621
17	operatingCashflow	12855000064
20	ebitda	13306000384
88	lastSplitDate	1344816000
35	totalCash	14871000064
70	lastFiscalYearEnd	1609372800

80	mostRecentQuarter	1633046400
91	lastDividendDate	1638230400
82	sharesShortPreviousMonthDate	1638230400
123	exDividendDate	1638230400
95	dateShortInterest	1640908800
78	nextFiscalYearEnd	1672444800
3	debtToEquity	172.826

2		
136	averageVolume	17746368
140	volume	18219394
129	regularMarketVolume	18219394
23	grossProfits	19581000000
86	priceHint	2
7	trailingEps	2.031

3		
42	recommendationMean	2.1
60	forwardEps	2.43
153	trailingPegRatio	2.6848
96	pegRatio	2.77
119	averageVolume10days	20867790
11	averageDailyVolume10Day	20867790

3		
57	enterpriseToEbitda	21.583
98	forwardPE	24.526747
101	sharesShortPriorMonth	24026403
30	numberOfAnalystOpinions	25
133	marketCap	257437417472
85	enterpriseValue	287178719232

128	trailingPE	29.34515
89	lastSplitFactor	2:1
143	fiveYearAvgDividendYield	3.21
38	totalCashPerShare	3.443
1	zip	30313
67	sharesShort	31874471
3	totalRevenue	37802000384

7		
83	floatShares	3890760972
6	phone	404 676 2121
36	totalDebt	41707999232
102	impliedSharesOutstanding	4311130112
62	sharesOutstanding	4319419904
144	fiftyTwoWeekLow	48.11

66	bookValue	5.135
107	twoHundredDayAverage	55.77645
115	fiftyDayAverage	57.6512
21	targetLowPrice	58
137	dayLow	59.21
126	regularMarketDayLow	59.21

150	regularMarketPrice	59.6
26	currentPrice	59.6
117	open	59.79
106	regularMarketOpen	59.79
114	regularMarketPreviousClose	59.82
10	previousClose	59.82

5		
94	priceToSalesTrailing12Months	6.8101535
149	dayHigh	60.345
111	regularMarketDayHigh	60.345
141	fiftyTwoWeekHigh	61.45
31	targetMeanPrice	63.72
2	targetMedianPric	64

5	e	
55	enterpriseToRevenue	7.597
34	targetHighPrice	70
24	freeCashflow	7007374848
40	revenuePerShare	8.771
148	bidSize	800
3	fullTimeEmployees	80300

72	netIncomeToCommon	8812999680
46	exchangeTimezoneName	America/New_York
5	city	Atlanta
13	industry	Beverages—Non-Alcoholic
44	shortName	Coca-Cola Company (The)
2	sector	Consumer Defensive
50	quoteType	EQUITY

47	exchangeTimezoneShortName	EST
146	tradeable	False
48	isEsgPopulated	False
7	state	GA
51	symbol	KO
43	exchange	NYQ
12	address1	One Coca-Cola Plaza

45	longName	The Coca-Cola Company
4	longBusinessSummary	The Coca-Cola Company, a beverage company, manufactures, markets, and sells various nonalcoholic beverages worldwide. The company provides sparkling soft drinks; water, enhanced water, and sports drinks; juice, dairy, and plantÂ‑based beverages; tea and coffee; and energy drinks. It also offers beverage concentrates and syrups, as well as fountain syrups to fountain retailers, such as restaurants and convenience stores. The company sells its products under the Coca-Cola, Diet Coke/Coca-Cola Light, Coca-Cola Zero Sugar, Fanta, Fresca, Schweppes, Sprite, Thums Up, Aquarius, Ciel, Dasani, glacÃ©au smartwater, glacÃ©au vitaminwater, Ice Dew, I LOHAS, Powerade, Topo Chico, AdeS, Del Valle, fairlife, innocent, Minute Maid, Minute Maid Pulpy, Simply, Ayataka, Costa, dogadan, FUZE TEA, Georgia, Gold Peak, HONEST TEA, and Kochakaden brands. It operates through a network of independent bottling partners, distributors, wholesalers, and retailers, as well as through bottling and distribution operators. The company was founded in 1886 and is headquartered in Atlanta, Georgia.

127	currency	USD
39	financialCurrency	USD
8	country	United States
0	Key	Value
9	companyOfficers	[]
22	recommendationKey	buy
52	messageBoardId	finmb_26642

152	logo_url	https://logo.clearbit.com/coca-colacompany.com
10	website	https://www.coca-colacompany.com
53	market	us_market
54	annualHoldingsTurnover	nan
56	beta3Year	nan
59	morningStarRiskRating	nan
6	revenueQuarterly	nan

1	Growth	
63	fundInceptionDate	nan
64	annualReportExpenseRatio	nan
65	totalAssets	nan
69	fundFamily	nan
79	yield	nan
87	threeYearAverageReturn	nan

90	legalType	nan
92	morningStarOverallRating	nan
97	ytdReturn	nan
99	lastCapGain	nan
103	category	nan
104	fiveYearAverageReturn	nan

110	volume24Hr	nan
112	navPrice	nan
118	toCurrency	nan
120	expireDate	nan
121	algorithm	nan
12	circulatingSupply	nan

4		
125	startDate	nan
130	lastMarket	nan
131	maxSupply	nan
132	openInterest	nan
134	volumeAllCurrencies	nan

135	strikePrice	nan
142	fromCurrency	nan
151	preMarketPrice	nan

5. Basic CAGR

unfold_moreShow hidden cell

In [ ]:

5.1 Basic Rolling Averages

In [9]:

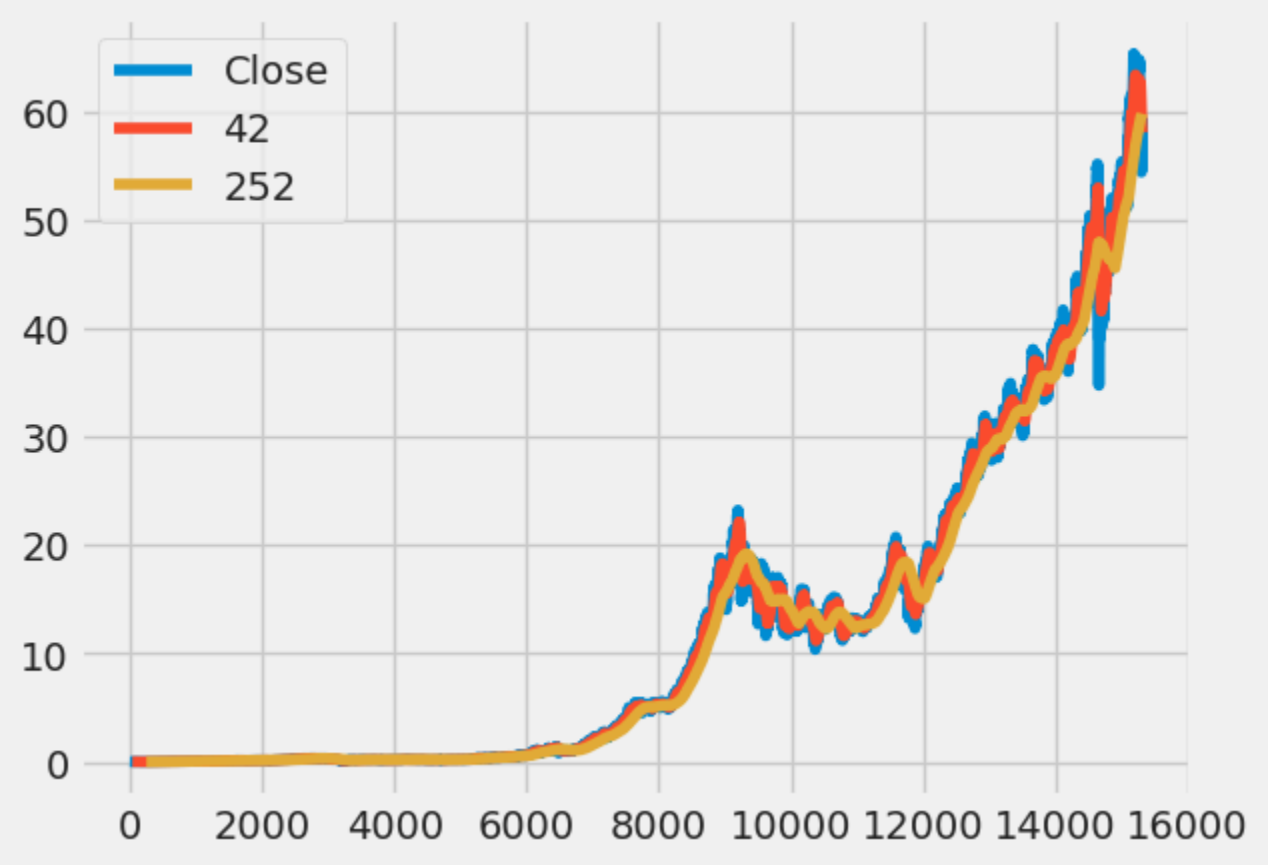
```
# Isolate the adjusted closing prices
```

```
adj_close_px = KO_Data['Close']


# Calculate the moving average
moving_avg = adj_close_px.rolling(window=40).mean()


# Inspect the result
moving_avg[-10:]
```

```
Out[9]:
15301    59.573229
15302    59.329031
15303    59.103823
15304    58.921440
15305    58.725320
15306    58.504966
15307    58.298918
15308    58.171838
15309    58.088689
15310    58.030935

Name: Close, dtype: float64
```

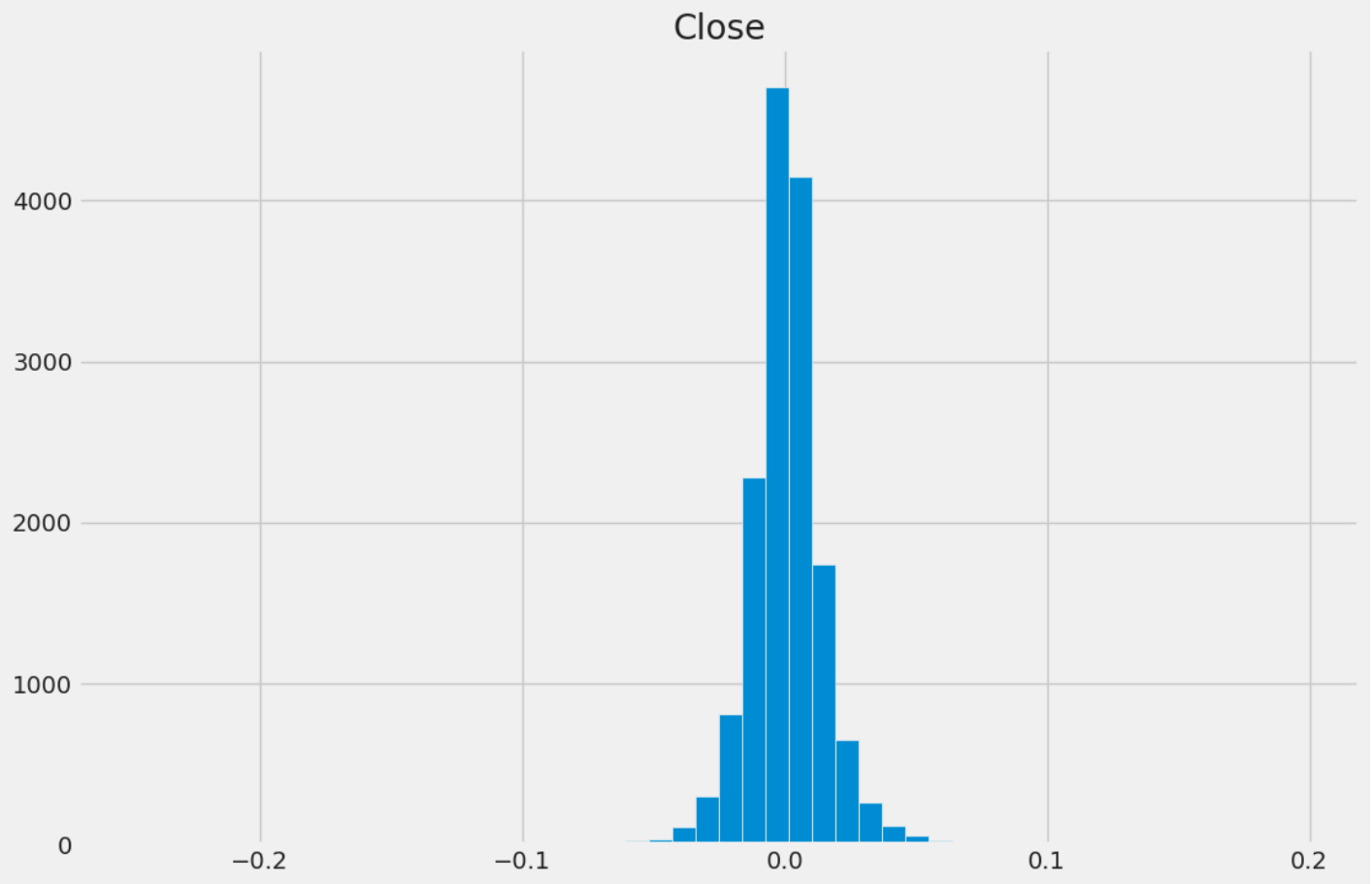
```
In [10]:
# Short moving window rolling mean
KO_Data['42'] = adj_close_px.rolling(window=40).mean()
```

```
# Long moving window rolling mean
KO_Data['252'] = adj_close_px.rolling(window=252).mean()

# Plot the adjusted closing price, the short and long windows of
rolling means
KO_Data[['Close', '42', '252']].plot()

plt.show()
```



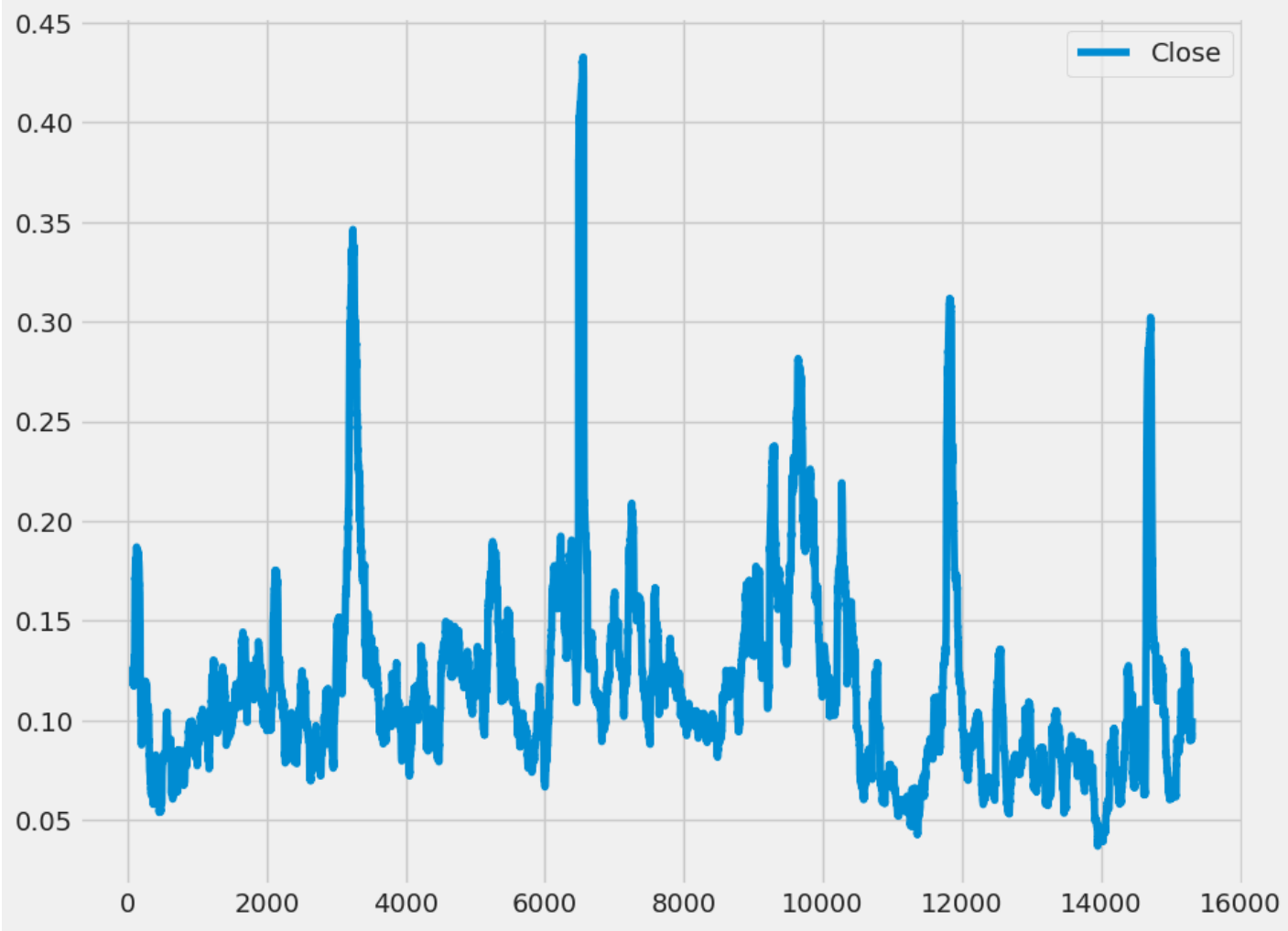
In [11]:

```
daily_close_px = KO_Data[['Close']]
# Calculate the daily percentage change for `daily_close_px`
daily_pct_change = daily_close_px.pct_change()

# Plot the distributions
daily_pct_change.hist(bins=50, sharex=True, figsize=(12,8))

# Show the resulting plot
plt.show()
```



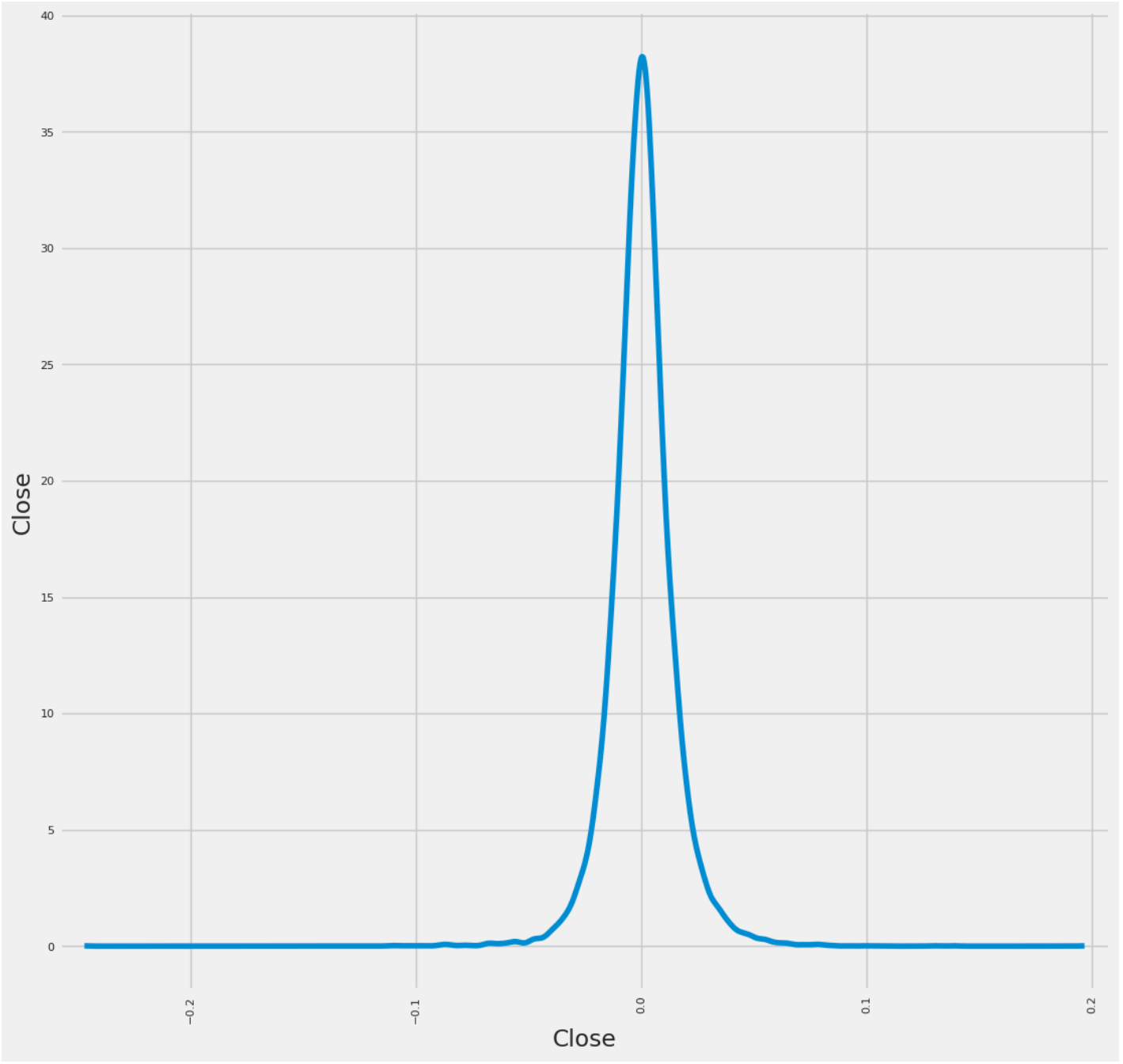
In [12]:

```
# Define the minumum of periods to consider
min_periods = 75

# Calculate the volatility
vol = daily_pct_change.rolling(min_periods).std() *
np.sqrt(min_periods)

# Plot the volatility
vol.plot(figsize=(10, 8))

# Show the plot
plt.show()
```



In [13]:

```
# Plot a scatter matrix with the `daily_pct_change` data
pd.plotting.scatter_matrix(daily_pct_change, diagonal='kde',
alpha=0.1,figsize=(12,12))

# Show the plot
plt.show()
```



5.2 Basic MACD

In [14]:

```
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()
```

```
19701980199020002010202001020304050 60
```

5.2.1 Basic SMA

In [15]:

```
#KO_Data=KO_Data.reset_index()

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()
```

```

```

```
19701980199020002010202001020304050 60
```

```
OHLCSMA5SMA20SMA50SMA200SMA500
```

5.2.2 Basic EMA

```
In [16]:
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()
```

19701980199020002010202001020304050 60

OHLCEMA5EMA20EMA50EMA200EMA500

In [17]:

```
KO_Data.head()
#KO_Data.fillna(0)
#KO_Data.set_index('Date')
```

Out[17]:

	index	Date	Open	High	Low	Close	Volume	Dividends	StockSp	42	...	SMA5	SMA20	SMA50	SMA200	SMA500	EMA5	EMA20	EMA50	EMA200	EMA500

									lits											
0	0	1962-01-02	0.05001616	0.05137816	0.05001616	0.05001616	806400	0.0	0	NaN	..	NaN	NaN	NaN	NaN	NaN	0.05001616	0.05001616	0.05001616	0.05001616	0.05001616
1	1	1962-01-03	0.04927373	0.04927373	0.04815959	0.04890202	1574400	0.0	0	NaN	..	NaN	NaN	NaN	NaN	NaN	0.04964444	0.04991010	0.04997272	0.05000505	0.05001111

2	2	1962-01-04	0.04902	0.04964	0.04902	0.04927	844800	0.0	0	NaN	...	NaN	NaN	NaN	NaN	NaN	0.04952	0.04984	0.04994	0.04998	0.05000
3	3	1962-01-05	0.04927	0.04989	0.04803	0.04815	1420800	0.0	0	NaN	...	NaN	NaN	NaN	NaN	NaN	0.04906	0.04968	0.04987	0.04999	0.05000
4	4	1966	0.04777	0.04777	0.04767	0.04776	203	0.0	0	Na	.	0.04888	Na	Na	Na	Na	0.04885	0.04894	0.04997	0.04999	0.04999

		2-01-08	87	87	35	64	5200			N	.	03	N	N	N	N	99	95	88	56	92

5 rows × 21 columns

6 FINTA Tech Analysis Ratios

Let us do a financial ratios calculation using FINTA library

- 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'
- Kaufman's 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 Strenght 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 %K 'STOCH'
- Stochastic oscillator %D 'STOCHD'
- Stochastic RSI 'STOCHRSI'
- Williams %R 'WILLIAMS'
- Ultimate Oscillator 'UO'

- Awesome Oscillator 'AO'
- Mass Index 'MI'
- Vortex Indicator 'VORTEX'
- 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'
- Elder's 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'

In [18]:

```
try:
    from finta import TA
    from backtesting import Backtest, Strategy
    from backtesting.lib import crossover
except:
    !pip install finta backtesting
    from finta import TA
    from backtesting import Backtest, Strategy
    from backtesting.lib import crossover
```

```
Collecting finta
  Downloading finta-1.3-py3-none-any.whl (29 kB)
Collecting backtesting
  Downloading Backtesting-0.3.3.tar.gz (175 kB)


━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━

━━━ 175.5/175.5 kB 4.3 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... - done
```

```
Requirement already satisfied: numpy in
/opt/conda/lib/python3.10/site-packages (from finta) (1.23.5)
Requirement already satisfied: pandas in
/opt/conda/lib/python3.10/site-packages (from finta) (2.0.3)
Requirement already satisfied: bokeh>=1.4.0 in
/opt/conda/lib/python3.10/site-packages (from backtesting)
(3.2.2)
Requirement already satisfied: Jinja2>=2.9 in
/opt/conda/lib/python3.10/site-packages (from
bokeh>=1.4.0->backtesting) (3.1.2)
Requirement already satisfied: contourpy>=1 in
/opt/conda/lib/python3.10/site-packages (from
bokeh>=1.4.0->backtesting) (1.1.0)
Requirement already satisfied: packaging>=16.8 in
/opt/conda/lib/python3.10/site-packages (from
bokeh>=1.4.0->backtesting) (21.3)
Requirement already satisfied: pillow>=7.1.0 in
/opt/conda/lib/python3.10/site-packages (from
bokeh>=1.4.0->backtesting) (9.5.0)
Requirement already satisfied: PyYAML>=3.10 in
/opt/conda/lib/python3.10/site-packages (from
bokeh>=1.4.0->backtesting) (6.0)
Requirement already satisfied: tornado>=5.1 in
/opt/conda/lib/python3.10/site-packages (from
bokeh>=1.4.0->backtesting) (6.3.2)
```

```
Requirement already satisfied: xyzservices>=2021.09.1 in
/opt/conda/lib/python3.10/site-packages (from
bokeh>=1.4.0->backtesting) (2023.7.0)
Requirement already satisfied: python-dateutil>=2.8.2 in
/opt/conda/lib/python3.10/site-packages (from pandas->finta)
(2.8.2)
Requirement already satisfied: pytz>=2020.1 in
/opt/conda/lib/python3.10/site-packages (from pandas->finta)
(2023.3)
Requirement already satisfied: tzdata>=2022.1 in
/opt/conda/lib/python3.10/site-packages (from pandas->finta)
(2023.3)
Requirement already satisfied: MarkupSafe>=2.0 in
/opt/conda/lib/python3.10/site-packages (from
Jinja2>=2.9->bokeh>=1.4.0->backtesting) (2.1.3)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in
/opt/conda/lib/python3.10/site-packages (from
packaging>=16.8->bokeh>=1.4.0->backtesting) (3.0.9)
Requirement already satisfied: six>=1.5 in
/opt/conda/lib/python3.10/site-packages (from
python-dateutil>=2.8.2->pandas->finta) (1.16.0)
Building wheels for collected packages: backtesting
  Building wheel for backtesting (setup.py) ... - \ done
  Created wheel for backtesting:
filename=Backtesting-0.3.3-py3-none-any.whl size=173804
```

```
sha256=cacca55c1313b2d6d721b1315d0471739920756e04b6a2004aed8c0f
21dbc56e
  Stored in directory:
/root/.cache/pip/wheels/e2/30/7f/19cbe31987c6ebdb47f1f510343249
066711609e3da2d57176
Successfully built backtesting
Installing collected packages: finta, backtesting
Successfully installed backtesting-0.3.3 finta-1.3
```

BokehJS 3.2.2 successfully loaded.

In [19]:

```
fin_ma =
pd.read_csv('../input/coca-cola-stock-live-and-updated/Coca-Col
a_stock_history.csv',  parse_dates=True)
print(fin_ma.head())
ohlc=fin_ma
print(TA.SMA(ohlc, 42))
#ohlc.index = ohlc[index].dt.date
```

```
        Date      Open      High       Low     Close   Volume
Dividends  \
0  1962-01-02  0.050016  0.051378  0.050016  0.050016   806400
0.0
```

```
1  1962-01-03  0.049273  0.049273  0.048159  0.048902  1574400
0.0
2  1962-01-04  0.049026  0.049645  0.049026  0.049273   844800
0.0
3  1962-01-05  0.049273  0.049892  0.048035  0.048159  1420800
0.0
4  1962-01-08  0.047787  0.047787  0.046735  0.047664  2035200
0.0

   Stock Splits
0             0
1             0
2             0
3             0
4             0
0              NaN
1              NaN
2              NaN
3              NaN
4              NaN
           ...
15306    58.759467
15307    58.572686
15308    58.422110
15309    58.297065
```

```
15310    58.219369
Name: 42 period SMA, Length: 15311, dtype: float64



In [20]:
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',
                #' Vortex Indicator ' : 'VORTEX',
                ' 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',
```

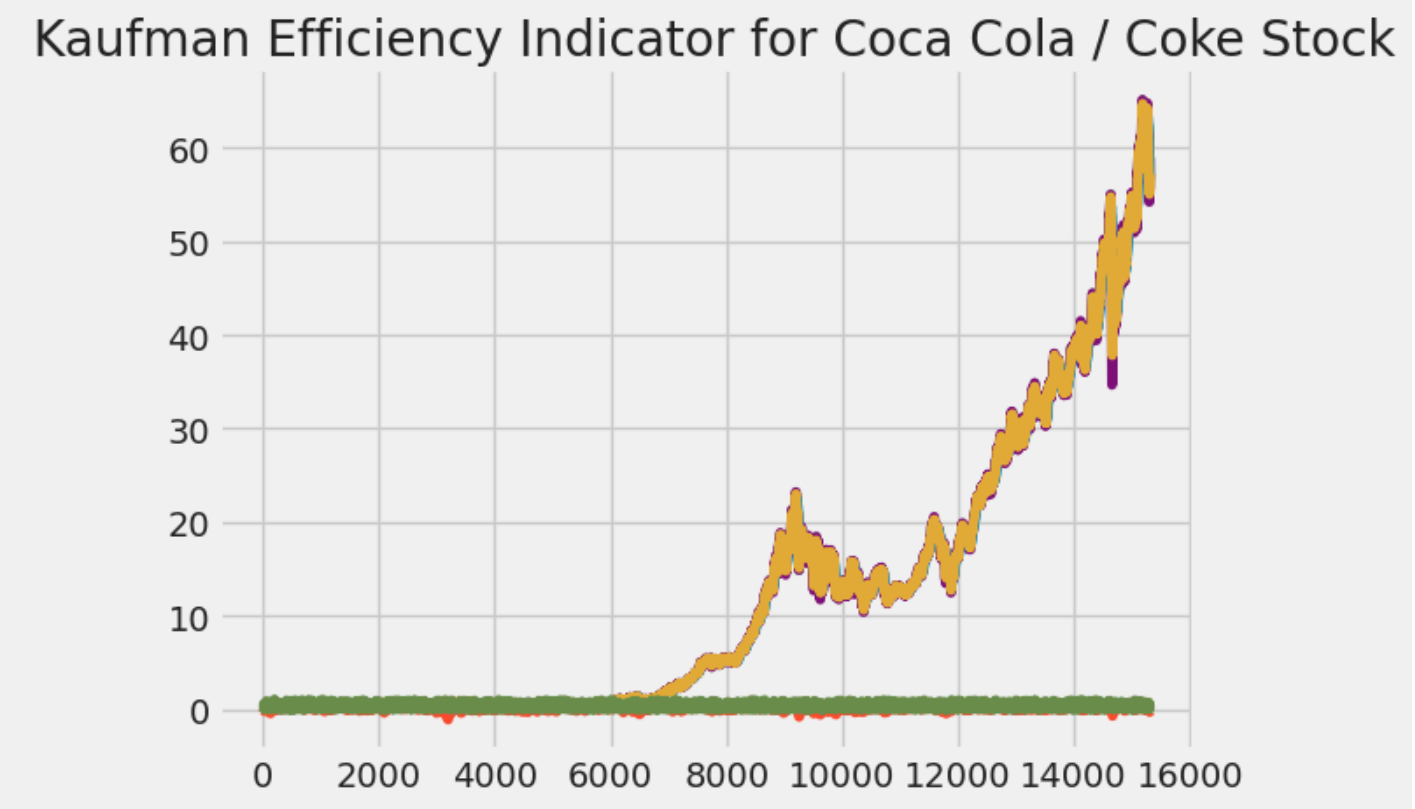
```
                    ' Normalized BASP ' : 'BASPN',
                    ' Chande Momentum Oscillator ' : 'CMO',
                    ' Chandelier Exit ' : 'CHANDELIER',
                    ' Qstick ' : 'QSTICK',
                    #' Twiggs Money Index ' : 'TMF',
                    ' Wave Trend Oscillator ' : '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')"
    #print(function_name)
    result = eval(function_name)
```

```
Unexpected exception formatting exception. Falling back to
standard exception
```

```
Traceback (most recent call last):
  File
"/opt/conda/lib/python3.10/site-packages/IPython/core/interacti
veshell.py", line 3508, in run_code
    exec(code_obj, self.user_global_ns, self.user_ns)
  File "/tmp/ipykernel_20/59537527.py", line 84, in <module>
    result = eval(function_name)
  File "<string>", line 1, in <module>
  File
"/opt/conda/lib/python3.10/site-packages/finta/finta.py", line
34, in wrap
    return func(*args, **kwargs)
  File
"/opt/conda/lib/python3.10/site-packages/finta/finta.py", line
292, in KAMA
    sc.iteritems(), sma.shift().iteritems(),
ohlc[column].iteritems()
  File
"/opt/conda/lib/python3.10/site-packages/pandas/core/generic.py
", line 5989, in __getattr__
    return object.__getattribute__(self, name)
AttributeError: 'Series' object has no attribute 'iteritems'

During handling of the above exception, another exception
```

```
occurred:


Traceback (most recent call last):
  File
"/opt/conda/lib/python3.10/site-packages/IPython/core/interacti
veshell.py", line 2105, in showtraceback
    stb = self.InteractiveTB.structured_traceback(
  File
"/opt/conda/lib/python3.10/site-packages/IPython/core/ultratb.p
y", line 1428, in structured_traceback
    return FormattedTB.structured_traceback(
  File
"/opt/conda/lib/python3.10/site-packages/IPython/core/ultratb.p
y", line 1319, in structured_traceback
    return VerboseTB.structured_traceback(
  File
"/opt/conda/lib/python3.10/site-packages/IPython/core/ultratb.p
y", line 1172, in structured_traceback
    formatted_exception =
self.format_exception_as_a_whole(etype, evalue, etb,
number_of_lines_of_context,
  File
"/opt/conda/lib/python3.10/site-packages/IPython/core/ultratb.p
y", line 1062, in format_exception_as_a_whole
    self.get_records(etb, number_of_lines_of_context,
```

```
tb_offset) if etb else []
  File
"/opt/conda/lib/python3.10/site-packages/IPython/core/ultratb.p
y", line 1154, in get_records
    FrameInfo(
  File
"/opt/conda/lib/python3.10/site-packages/IPython/core/ultratb.p
y", line 780, in __init__
    ix = inspect.getsourcelines(frame)
  File "/opt/conda/lib/python3.10/inspect.py", line 1121, in
getsourcelines
    lines, lnum = findsource(object)
  File "/opt/conda/lib/python3.10/inspect.py", line 958, in
findsource
    raise OSError('could not get source code')
OSError: could not get source code
```



7 Back Testing Trading Strategy

In [ ]:

```
# Defining DEMA cross strategy
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()
```

Let us do a bit of backtesting with a value of $100000

In [ ]:

```
ohlc.head()
print(ohlc.Date)
```

In [ ]:

In [ ]:

```
bt = Backtest(ohlc, DemaCross,
              cash=100000, commission=0.015,
exclusive_orders=True)
```

Back Testing Summary

In [ ]:

```
bt.run()
```

As you can see, if you had invested $100,000 in Coca Cola shares, you would have got by now a return of 118642%!

In [ ]:

```
bt.plot()
```

In [ ]:

```
data=ohlc
```

7.1 BackTesting Trading Strategy Heatmaps

In [ ]:

```
from backtesting import Strategy
from backtesting.lib import crossover
from backtesting.test import SMA
```

In [ ]:

```
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)

# Design matrix / independent features:

# Price-derived features
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
```

```
# Indicator features
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')

# Some datetime features for good measure
#data['X_day'] = data.index.dayofweek
#data['X_hour'] = data.index.hour

#data = data.apply(pd.to_numeric)
#data = data.dropna().astype(np.float64)
#data.fillna(method="ffill")
#data =data[~data.isin([np.nan, np.inf, -np.inf]).any(1)]

#data.replace([np.inf, -np.inf], 0.0, inplace=True)
#data = data.fillna(data.mean(), inplace=True)
#data = data.dropna().astype(np.float64)
```

In [ ]:

```
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:

            # On upwards trend, if price closes above
            # "entry" MA, go long

            # Here, even though the operands are arrays, this
            # works by implicitly comparing the two last values
            if self.sma1 > self.sma2:
                if crossover(self.data.Close, self.sma_enter):
                    self.buy()
```

```
            # On downwards trend, if price closes below
            # "entry" MA, go short

            else:
                if crossover(self.sma_enter, self.data.Close):
                    self.sell()

        # But if we already hold a position and the price
        # closes back below (above) "exit" MA, close the
position

        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()
```

In [ ]:

```
%%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)
```

```
In [ ]:
heatmap
```

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

In [ ]:

```
from backtesting.lib import plot_heatmaps


plot_heatmaps(heatmap, agg='mean')
```

In [ ]:

```
%%capture

!pip install scikit-optimize  # This is a run-time dependency
```

In [ ]:

```
%%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)
```

```
In [ ]:
from skopt.plots import plot_objective
```

```
_ = plot_objective(optimize_result, n_points=10)
```

Reference link