hw2

October 27, 2024

0.0.1

[1]:
```
import warnings
warnings.filterwarnings("ignore")
```

[2]:
```
!pip install yfinance -q

import yfinance as yf
```

[3]:
```
#python             3.6-3.9          numpy==1.19.5.                    ta-lib
!pip uninstall -y numpy
!pip install numpy==1.24.4 -q

import numpy as np
```

```
Found existing installation: numpy 1.24.4
Uninstalling numpy-1.24.4:
  Successfully uninstalled numpy-1.24.4
```

[4]:
```
#               MacOS 10.15
#                brew install ta-lib
!pip install TA-Lib -q

import talib
```

[5]:
```
!pip install plotly -q

import plotly.graph_objects as go
from plotly.subplots import make_subplots

def linear_plot(df, title):
    fig = go.Figure([go.Scatter(x=df['date'], y=df['close'], mode='lines')])
    fig.update_layout(plot_bgcolor='white',
                      xaxis_title='Date',
                      yaxis_title='Price',
                      title=title)
    fig.show()
```

[6]:
```
import pandas as pd
import itertools
```

0.0.2

[7]:
```
#     10                  sp500
gold_df = yf.download('GC=F', period='10y', interval='1mo')
sp500_df = yf.download('^GSPC', period='10y', interval='1mo')
gold_df = gold_df.reset_index()

#       sp500_df   gold_df        'Date',            '_gold'         'Close'  ␣
 ↪gold_df
sp500_df = sp500_df.merge(
    gold_df[['Date', 'Close']],
    on='Date',
    how='left',
    suffixes=('', '_gold')
)

#                          'Close_gold'
sp500_df['Close_gold'] = sp500_df['Close_gold'].ffill()

#       gold/sp500 ratio
sp500_df["gsp_ratio"] = sp500_df['Close_gold'] / sp500_df["Close"].values
sp500_df["gsp_ratioSMA"] = talib.SMA(sp500_df["gsp_ratio"], timeperiod=12)
sp500_df.dropna(inplace=True)
fig = make_subplots(specs=[[{"secondary_y": True}]])

fig.add_trace(go.Scatter(x=sp500_df["Date"], y=sp500_df["gsp_ratio"],␣
 ↪mode='lines', name='gold/sp500 ratio'), secondary_y=False)
fig.add_trace(go.Scatter(x=sp500_df["Date"], y=sp500_df["gsp_ratioSMA"],␣
 ↪mode='lines', name='sma_ratio'), secondary_y=False)
fig.add_trace(go.Scatter(x=sp500_df["Date"], y=sp500_df["Close"],␣
 ↪mode='markers+lines', name='SP500 Price'), secondary_y=True)

#
fig.update_layout(
    title='GOLD to SP500 Ratio vs SP500 Price (Last 10 Years)',
    yaxis=dict(title='Ratio GOLD to SP500'),
    yaxis2=dict(title='SP500 Price')
)

fig.show()
```

```
[**********************100%**********************]  1 of 1 completed
[**********************100%**********************]  1 of 1 completed
```

[8]:
```
#     10            BTC   sp500
btc_df = yf.download("BTC-USD", period='10y', interval='1mo')
sp500_df = yf.download('^GSPC', period='10y', interval='1mo')
btc_df = btc_df.reset_index()

#       sp500_df   btc_df        'Date',           '_btc'        'Close'  ␣
 ↪btc_df
sp500_df = sp500_df.merge(
    btc_df[['Date', 'Close']],
    on='Date',
    how='left',
    suffixes=('', '_btc')
)

#                          'Close_btc'
sp500_df['Close_btc'] = sp500_df['Close_btc'].ffill()

#       btc/sp500 ratio
sp500_df["btc_sp_ratio"] = sp500_df['Close_btc'] / sp500_df["Close"].values
sp500_df["btc_sp_ratioSMA"] = talib.SMA(sp500_df["btc_sp_ratio"], timeperiod=12)
sp500_df.dropna(inplace=True)
fig = make_subplots(specs=[[{"secondary_y": True}]])

fig.add_trace(go.Scatter(x=sp500_df["Date"], y=sp500_df["btc_sp_ratio"],␣
 ↪mode='lines', name='BTC/sp500 ratio'), secondary_y=False)
fig.add_trace(go.Scatter(x=sp500_df["Date"], y=sp500_df["btc_sp_ratioSMA"],␣
 ↪mode='lines', name='sma_ratio'), secondary_y=False)
fig.add_trace(go.Scatter(x=sp500_df["Date"], y=sp500_df["Close"],␣
 ↪mode='markers+lines', name='SP500 Price'), secondary_y=True)

#
fig.update_layout(
    title='BTC to SP500 Ratio vs SP500 Price (Last 10 Years)',
    yaxis=dict(title='Ratio BTC to SP500'),
    yaxis2=dict(title='SP500 Price')
)

fig.show()
```

```
[**********************100%**********************]  1 of 1 completed
[**********************100%**********************]  1 of 1 completed
```

, BTC 10 S&P 500, S&P 500
. , , . ,
, , S&P 500

[9]:
```
!pip install scikit-learn -q

from sklearn.linear_model import LinearRegression
```

, S&P 500 , . -
Nvidia (NVDA)

[10]:
```
from datetime import datetime, timedelta
import plotly.express as px

def calculate_alpha_beta(stock_data, sp500_data):
    stock_returns = stock_data.pct_change().dropna()
    sp500_returns = sp500_data.pct_change().dropna()

    df = pd.DataFrame({'stock': stock_returns, 'sp500': sp500_returns}).dropna()

    X = df['sp500'].values.reshape(-1, 1)
    y = df['stock'].values
    reg = LinearRegression().fit(X, y) #

    beta = reg.coef_[0]
    alpha = reg.intercept_

    return alpha, beta

#
end_date = datetime.now()
end_date_dt = pd.to_datetime(end_date).tz_localize('UTC')
train_start = end_date - timedelta(days=365*2)  # 1
train_start_dt = pd.to_datetime(train_start).tz_localize('UTC')  # 1
test_start = end_date - timedelta(days=365)
test_start_dt = pd.to_datetime(test_start).tz_localize('UTC')# 1

#
stocks = ['AAPL', 'MSFT', 'AMZN', 'GOOGL', 'META', 'NVDA', 'TSLA', 'JPM',␣
 ↪'JNJ', 'V',
          'PG', 'UNH', 'HD', 'MA', 'DIS', 'ADBE', 'CRM', 'NFLX', 'PYPL', 'INTC']

#           S&P 500
data = yf.download(stocks + ['^GSPC'], start=train_start, end=end_date)
sp500 = data['Close']['^GSPC']

results = {}

for stock in stocks:
    try:
        stock_data = data['Close'][stock]
```

```
        #
        train_stock_data = stock_data[stock_data.index < test_start_dt]
        print(stock)
        print(f"Train Period : {train_stock_data.index[0]} - {train_stock_data.
 ↪index[-1]} ")
        print(f"Test Period : {stock_data.loc[test_start_dt:].index[0]} -␣
 ↪{stock_data.loc[test_start_dt:].index[-1]}")

        train_sp500_data = sp500[sp500.index < test_start_dt]
        alpha, beta = calculate_alpha_beta(train_stock_data, train_sp500_data)

        #
        test_return = (stock_data.loc[test_start_dt:].iloc[-1] / stock_data.
 ↪loc[test_start_dt:].iloc[0]) - 1

        results[stock] = {'Alpha': alpha, 'Beta': beta, 'Test_Return':␣
 ↪test_return}
    except Exception as e:
        print(f"             {stock}: {e}")

#      DataFrame
results_df = pd.DataFrame(results).T
results_df = results_df.sort_values('Alpha', ascending=False)

#                   Plotly
fig = px.scatter(results_df, x='Alpha', y='Test_Return', text=results_df.index,
                 title='Alpha vs Actual Return', labels={'Alpha': 'Alpha␣
 ↪(Training Period)', 'Test_Return': 'Return (Test Period)'})
fig.update_traces(textposition='top center')
fig.update_layout(showlegend=False)
fig.show()

#                        -5      -5              Plotly
top_5 = results_df.head().index
bottom_5 = results_df.tail().index

fig = px.line()

for stock in top_5:
    stock_data = data['Close'][stock][test_start_dt:]
    fig.add_scatter(x=stock_data.index, y=stock_data / stock_data.iloc[0],␣
 ↪mode='lines', name=stock)

for stock in bottom_5:
    stock_data = data['Close'][stock][test_start_dt:]
```

```
    fig.add_scatter(x=stock_data.index, y=stock_data / stock_data.iloc[0],␣
 ↪mode='lines', name=stock, line=dict(dash='dash'))

fig.update_layout(title='Price Performance during Test Period',␣
 ↪xaxis_title='Date', yaxis_title='Normalized Price')
fig.show()

print("  -5        :")
print(top_5)

print("\n  -5         :")
print(bottom_5)
```

```
[**********100%**********]  21 of 21 completed
```

```
AAPL
Train Period : 2022-10-31 00:00:00+00:00 - 2023-10-27 00:00:00+00:00
Test Period : 2023-10-30 00:00:00+00:00 - 2024-10-25 00:00:00+00:00
MSFT
Train Period : 2022-10-31 00:00:00+00:00 - 2023-10-27 00:00:00+00:00
Test Period : 2023-10-30 00:00:00+00:00 - 2024-10-25 00:00:00+00:00
AMZN
Train Period : 2022-10-31 00:00:00+00:00 - 2023-10-27 00:00:00+00:00
Test Period : 2023-10-30 00:00:00+00:00 - 2024-10-25 00:00:00+00:00
GOOGL
Train Period : 2022-10-31 00:00:00+00:00 - 2023-10-27 00:00:00+00:00
Test Period : 2023-10-30 00:00:00+00:00 - 2024-10-25 00:00:00+00:00
META
Train Period : 2022-10-31 00:00:00+00:00 - 2023-10-27 00:00:00+00:00
Test Period : 2023-10-30 00:00:00+00:00 - 2024-10-25 00:00:00+00:00
NVDA
Train Period : 2022-10-31 00:00:00+00:00 - 2023-10-27 00:00:00+00:00
Test Period : 2023-10-30 00:00:00+00:00 - 2024-10-25 00:00:00+00:00
TSLA
Train Period : 2022-10-31 00:00:00+00:00 - 2023-10-27 00:00:00+00:00
Test Period : 2023-10-30 00:00:00+00:00 - 2024-10-25 00:00:00+00:00
JPM
Train Period : 2022-10-31 00:00:00+00:00 - 2023-10-27 00:00:00+00:00
Test Period : 2023-10-30 00:00:00+00:00 - 2024-10-25 00:00:00+00:00
JNJ
Train Period : 2022-10-31 00:00:00+00:00 - 2023-10-27 00:00:00+00:00
Test Period : 2023-10-30 00:00:00+00:00 - 2024-10-25 00:00:00+00:00
V
Train Period : 2022-10-31 00:00:00+00:00 - 2023-10-27 00:00:00+00:00
Test Period : 2023-10-30 00:00:00+00:00 - 2024-10-25 00:00:00+00:00
PG
Train Period : 2022-10-31 00:00:00+00:00 - 2023-10-27 00:00:00+00:00
Test Period : 2023-10-30 00:00:00+00:00 - 2024-10-25 00:00:00+00:00
```

```
UNH
Train Period : 2022-10-31 00:00:00+00:00 - 2023-10-27 00:00:00+00:00
Test Period : 2023-10-30 00:00:00+00:00 - 2024-10-25 00:00:00+00:00
HD
Train Period : 2022-10-31 00:00:00+00:00 - 2023-10-27 00:00:00+00:00
Test Period : 2023-10-30 00:00:00+00:00 - 2024-10-25 00:00:00+00:00
MA
Train Period : 2022-10-31 00:00:00+00:00 - 2023-10-27 00:00:00+00:00
Test Period : 2023-10-30 00:00:00+00:00 - 2024-10-25 00:00:00+00:00
DIS
Train Period : 2022-10-31 00:00:00+00:00 - 2023-10-27 00:00:00+00:00
Test Period : 2023-10-30 00:00:00+00:00 - 2024-10-25 00:00:00+00:00
ADBE
Train Period : 2022-10-31 00:00:00+00:00 - 2023-10-27 00:00:00+00:00
Test Period : 2023-10-30 00:00:00+00:00 - 2024-10-25 00:00:00+00:00
CRM
Train Period : 2022-10-31 00:00:00+00:00 - 2023-10-27 00:00:00+00:00
Test Period : 2023-10-30 00:00:00+00:00 - 2024-10-25 00:00:00+00:00
NFLX
Train Period : 2022-10-31 00:00:00+00:00 - 2023-10-27 00:00:00+00:00
Test Period : 2023-10-30 00:00:00+00:00 - 2024-10-25 00:00:00+00:00
PYPL
Train Period : 2022-10-31 00:00:00+00:00 - 2023-10-27 00:00:00+00:00
Test Period : 2023-10-30 00:00:00+00:00 - 2024-10-25 00:00:00+00:00
INTC
Train Period : 2022-10-31 00:00:00+00:00 - 2023-10-27 00:00:00+00:00
Test Period : 2023-10-30 00:00:00+00:00 - 2024-10-25 00:00:00+00:00

 -5        :
Index(['META', 'NVDA', 'ADBE', 'MSFT', 'NFLX'], dtype='object')

 -5        :
Index(['TSLA', 'HD', 'JNJ', 'DIS', 'PYPL'], dtype='object')
```

[11]:
```python
#
def plot_regression(stock, stock_data, sp500_data):
    stock_returns = stock_data.pct_change().dropna()
    sp500_returns = sp500_data.pct_change().dropna()

    df = pd.DataFrame({'stock': stock_returns, 'sp500': sp500_returns}).dropna()

    X = df['sp500'].values.reshape(-1, 1)
    y = df['stock'].values
    reg = LinearRegression().fit(X, y)

    beta = reg.coef_[0]
    alpha = reg.intercept_
```

```
    regression_line = reg.predict(X)

    fig = go.Figure()

    #
    fig.add_trace(go.Scatter(x=sp500_returns, y=stock_returns,
                             mode='markers', name=f'{stock} Data'))

    #
    fig.add_trace(go.Scatter(x=sp500_returns, y=regression_line,
                             mode='lines', name='Regression Line',
                             line=dict(color='gray')))

    fig.update_layout(title=f'{stock} Alpha: {alpha:.4f}, Beta: {beta:.4f}',
                      xaxis_title='S&P 500 Returns',
                      yaxis_title=f'{stock} Returns')

    fig.show()

#                 -5
for stock in top_5:
    stock_data = data['Close'][stock][train_start_dt:end_date_dt]
    plot_regression(stock, stock_data, sp500[train_start_dt:end_date_dt])
```

[12]:
```
df = yf.download("NVDA", start='2020-01-01', interval='1d').drop(columns=['Adj␣
↪Close'])
df = df.reset_index()
df.columns = df.columns.str.lower()
df
```

```
[**********************100%**********************]  1 of 1 completed
```

[12]:

	date	close	high	low	open	volume
0	2020-01-02 00:00:00+00:00	5.997750	5.997750	5.918000	5.968750	237536000
1	2020-01-03 00:00:00+00:00	5.901750	5.945750	5.852500		
2	2020-01-06 00:00:00+00:00	5.926500	5.931750	5.781750		
3	2020-01-07 00:00:00+00:00	5.998250	6.044250	5.909750		
4	2020-01-08 00:00:00+00:00	6.009500	6.051000	5.953750		
…	…	…	…	…		
1208	2024-10-21 00:00:00+00:00	143.710007	143.710007	138.000000		
1209	2024-10-22 00:00:00+00:00	143.589996	144.419998	141.779999		
1210	2024-10-23 00:00:00+00:00	139.559998	142.429993	137.460007		
1211	2024-10-24 00:00:00+00:00	140.410004	141.350006	138.460007		
1212	2024-10-25 00:00:00+00:00	141.539993	144.130005	140.800003		

```
1        5.877500  205384000
2        5.808000  262636000
3        5.955000  314856000
4        5.994000  277108000
…             …          …
1208   138.130005  264554500
1209   142.910004  226311600
1210   142.029999  285930000
1211   140.820007  172354900
1212   140.929993  204182400

[1213 rows x 6 columns]
```

[13]:
```
linear_plot(df, 'NVidia')
```

[14]:
```
bt_df_sma = df.copy()
bt_df_sma.columns = bt_df_sma.columns.str.capitalize()
bt_df_sma.rename(columns={'Date': 'Datetime'}, inplace=True)
bt_df_sma["Datetime"] = pd.to_datetime(bt_df_sma["Datetime"])
bt_df_sma.set_index('Datetime', inplace=True)
bt_df_sma
```

[14]:
```
                                Close        High         Low        Open  \
Datetime
2020-01-02 00:00:00+00:00    5.997750    5.997750    5.918000    5.968750
2020-01-03 00:00:00+00:00    5.901750    5.945750    5.852500    5.877500
2020-01-06 00:00:00+00:00    5.926500    5.931750    5.781750    5.808000
2020-01-07 00:00:00+00:00    5.998250    6.044250    5.909750    5.955000
2020-01-08 00:00:00+00:00    6.009500    6.051000    5.953750    5.994000
…                                 …           …           …           …
2024-10-21 00:00:00+00:00  143.710007  143.710007  138.000000  138.130005
2024-10-22 00:00:00+00:00  143.589996  144.419998  141.779999  142.910004
2024-10-23 00:00:00+00:00  139.559998  142.429993  137.460007  142.029999
2024-10-24 00:00:00+00:00  140.410004  141.350006  138.460007  140.820007
2024-10-25 00:00:00+00:00  141.539993  144.130005  140.800003  140.929993

                               Volume
Datetime
2020-01-02 00:00:00+00:00  237536000
2020-01-03 00:00:00+00:00  205384000
2020-01-06 00:00:00+00:00  262636000
2020-01-07 00:00:00+00:00  314856000
2020-01-08 00:00:00+00:00  277108000
…                                ...
2024-10-21 00:00:00+00:00  264554500
2024-10-22 00:00:00+00:00  226311600
2024-10-23 00:00:00+00:00  285930000
```

```
2024-10-24 00:00:00+00:00  172354900
2024-10-25 00:00:00+00:00  204182400

[1213 rows x 5 columns]
```

[15]:
```
!pip install backtesting -q

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

Backtesting (https://kernc.github.io/backtesting.py/doc/examples/Quick%20Start%20User%20Guide.html)

[16]:
```
class SmaCross(Strategy):
    # Define the two MA lags as *class variables*
    # for later optimization
    n1 = 10
    n2 = 20

    def init(self):
        # Precompute the two moving averages
        self.sma1 = self.I(SMA, self.data.Close, self.n1)
        self.sma2 = self.I(SMA, self.data.Close, self.n2)

    def next(self):
        # If sma1 crosses above sma2, close any existing
        # short trades, and buy the asset
        if crossover(self.sma1, self.sma2):
            self.position.close()
            self.buy()

        # Else, if sma1 crosses below sma2, close any existing
        # long trades, and sell the asset
        elif crossover(self.sma2, self.sma1):
            self.position.close()
            self.sell()
```

[17]:
```
bt_sma = Backtest(bt_df_sma, SmaCross, cash=1_000_000, commission=.002,␣
 ↪exclusive_orders=True)
stats = bt_sma.run()
print(stats)
```

```
Start                     2020-01-02 00:00…
End                       2024-10-25 00:00…
Duration                   1758 days 00:00:00
Exposure Time [%]                  98.103875
Equity Final [$]              2328218.808992
```

```
Equity Peak [$]                 3386925.725727
Return [%]                          132.821881
Buy & Hold Return [%]              2259.884921
Return (Ann.) [%]                    19.19251
Volatility (Ann.) [%]                66.049472
Sharpe Ratio                          0.290578
Sortino Ratio                         0.576255
Calmar Ratio                          0.303974
Max. Drawdown [%]                   -63.138643
Avg. Drawdown [%]                   -10.558119
Max. Drawdown Duration      911 days 00:00:00
Avg. Drawdown Duration       63 days 00:00:00
# Trades                                   60
Win Rate [%]                         33.333333
Best Trade [%]                       98.297312
Worst Trade [%]                     -27.108527
Avg. Trade [%]                        1.41847
Max. Trade Duration         168 days 00:00:00
Avg. Trade Duration          29 days 00:00:00
Profit Factor                         1.673995
Expectancy [%]                        3.168481
SQN                                   0.556558
_strategy                            SmaCross
_equity_curve                               …
_trades                        Size  Entr…
dtype: object
```

[18]:
```
bt_sma.plot()
```

```
BokehDeprecationWarning: Passing lists of formats for DatetimeTickFormatter
scales was deprecated in Bokeh 3.0. Configure a single string format for each
scale
BokehDeprecationWarning: Passing lists of formats for DatetimeTickFormatter
scales was deprecated in Bokeh 3.0. Configure a single string format for each
scale
```

[18]:
```
GridPlot(id='p1512', …)
```

, .

TEMA (Triple Exponential Moving Average) MACD (Moving Average Convergence/Divergence). ,

[19]:
```
class TechAnalysisStrategy(Strategy):
    def init(self):
        self.signal = self.I(lambda: self.data.Signal)
        self.previous_signal = 0
        self.size = 0.1
```

```
    def next(self):
        current_signal = self.signal[-1]

        if current_signal != self.previous_signal:
            if current_signal == 1:
                if self.position.is_short:
                    self.position.close()

                if not self.position.is_long:
                    self.buy(size=self.size)

            elif current_signal == -1:
                if self.position.is_long:
                    self.position.close()

                if not self.position.is_short:
                    self.sell(size=self.size)

            elif current_signal == 0:
                if self.position:
                    self.position.close()

        self.previous_signal = current_signal
```

[20]:
```
def apply_strategy(data, params):
    """
                                    .

    :param df: DataFrame       ,                       .
    :param params:                          .
    :return: DataFrame                          .
    """
    df = data.copy()
    #
    tema_period = params['tema_period']
    fastMACD_period = params['fastMACD_period']
    slowMACD_period = params['slowMACD_period']
    signalMACD_period = params['signalMACD_period']

    #
    df['tema'] = talib.TEMA(df['close'], timeperiod=tema_period)
    df['macd'], df['macd_signal'], df['macd_hist'] = talib.MACD(df['close'],␣
↪fastperiod=fastMACD_period, slowperiod=slowMACD_period,␣
↪signalperiod=signalMACD_period)
```

```
    #
    df['signal'] = 0
    df.loc[(df['macd'] > df['macd_signal']) & (df['close'] > df['tema']),␣
↪'signal'] = 1  #
    df.loc[(df['macd'] < df['macd_signal']) & (df['close'] < df['tema']),␣
↪'signal'] = -1  #

    return df[["date", "open", "high", "low", "close", "volume", "signal"]]
```

[21]:
```
def backtest_strategy(df, strategy_class, params, plot=False):
    """
                                    .

    :param df: DataFrame                .
    :param strategy_class:                    .
    :param params:                      .
    :return:              .
    """
    #
    df = apply_strategy(df.copy(), params)

    #
    bt_df = df.copy()
    bt_df.columns = bt_df.columns.str.capitalize()
    bt_df.rename(columns={'Date': 'Datetime'}, inplace=True)
    bt_df["Datetime"] = pd.to_datetime(bt_df["Datetime"])
    bt_df.set_index('Datetime', inplace=True)

    #               Backtest
    bt = Backtest(bt_df, strategy_class, cash=1_000_000, commission=.002,␣
↪exclusive_orders=True, margin=0.1)

    #
    stats = bt.run()
    if plot:
        bt.plot()
    return stats
```

-

[22]:
```
stats = backtest_strategy(df.copy(), TechAnalysisStrategy, {
    'tema_period': 30,
    'fastMACD_period': 12,
    'slowMACD_period': 26,
    'signalMACD_period': 9
```

```
}, True)
print(stats)
```

```
BokehDeprecationWarning: Passing lists of formats for DatetimeTickFormatter
scales was deprecated in Bokeh 3.0. Configure a single string format for each
scale
BokehDeprecationWarning: Passing lists of formats for DatetimeTickFormatter
scales was deprecated in Bokeh 3.0. Configure a single string format for each
scale
```

```
Start                     2020-01-02 00:00…
End                       2024-10-25 00:00…
Duration                   1758 days 00:00:00
Exposure Time [%]                  84.171476
Equity Final [$]               887098.985864
Equity Peak [$]               1193052.696269
Return [%]                       -11.290101
Buy & Hold Return [%]           2259.884921
Return (Ann.) [%]                 -2.458095
Volatility (Ann.) [%]             45.673331
Sharpe Ratio                            0.0
Sortino Ratio                           0.0
Calmar Ratio                            0.0
Max. Drawdown [%]                -58.764611
Avg. Drawdown [%]                -27.149549
Max. Drawdown Duration      989 days 00:00:00
Avg. Drawdown Duration      406 days 00:00:00
# Trades                                142
Win Rate [%]                      36.619718
Best Trade [%]                    39.897216
Worst Trade [%]                  -19.036421
Avg. Trade [%]                     -0.08434
Max. Trade Duration          40 days 00:00:00
Avg. Trade Duration          10 days 00:00:00
Profit Factor                      1.106542
Expectancy [%]                     0.299805
SQN                               -0.132554
_strategy                 TechAnalysisStra…
_equity_curve                             …
_trades                            Size  Ent…
dtype: object
```

. , Walk Forward Optimization
.

[23]:
```
def get_best_strategy(buffer, strategy_class):
    #
    tema_period_list = [7, 14, 28]
```

```
    fastMACD_period_list = [12, 35, 56]
    slowMACD_period_list = [9, 23, 39]
    signalMACD_period_list = [28, 40,80]

    #
    best_params = None
    best_performance = -float('inf')

    #
    for tema_period, fastMACD_period, slowMACD_period, signalMACD_period in␣
 ↪itertools.product(tema_period_list, fastMACD_period_list,␣
 ↪slowMACD_period_list, signalMACD_period_list):

        #
        params = {
            'tema_period': tema_period,
            'fastMACD_period': fastMACD_period,
            'slowMACD_period': slowMACD_period,
            'signalMACD_period': signalMACD_period
        }

        #
        stats = backtest_strategy(buffer.copy(), strategy_class, params)

        #           ,                              (     ,           )
        performance = stats['Profit Factor']

        #
        if performance > best_performance:
            best_performance = performance
            best_params = params

    print(f"               : {best_performance}")
    print(f"         : {best_params}")
    return best_params
```

[24]:
```
train_size = 200  #
test_size = 90    #

#         DataFrame
signals_df = pd.DataFrame()

#
num_iterations = (len(df) - train_size) // test_size

print(f"          {num_iterations + 1}      ")
```

```python
for i in range(num_iterations + 1):
    print(f"     {i + 1}")
    #
    start_train = i * test_size
    end_train = start_train + train_size
    start_test = end_train
    end_test = start_test + test_size

    #                                ,
    if end_test > len(df):
        end_test = len(df)

    #
    train_data = df.iloc[start_train:end_train].copy()
    test_data = df.iloc[start_test:end_test].copy()

    #
    best_params = get_best_strategy(train_data, TechAnalysisStrategy)

    #
    combined_data = pd.concat([train_data, test_data]).reset_index(drop=True)

    #
    combined_with_signal = apply_strategy(combined_data.copy(), best_params)

    #                  ,
    test_with_signal = combined_with_signal.iloc[-test_size:].copy()

    #                         signals_df
    signals_df = pd.concat([signals_df, test_with_signal], ignore_index=True)
```

```
         12
     1
              : 3.0121555477793014
          : {'tema_period': 28, 'fastMACD_period': 56, 'slowMACD_period':
39, 'signalMACD_period': 80}
     2
              : 0.726317163212322
          : {'tema_period': 7, 'fastMACD_period': 56, 'slowMACD_period':
9, 'signalMACD_period': 80}
     3
              : 9.791401350844502
          : {'tema_period': 28, 'fastMACD_period': 56, 'slowMACD_period':
39, 'signalMACD_period': 80}
     4
              : 6.038307308579817
          : {'tema_period': 28, 'fastMACD_period': 12, 'slowMACD_period':
23, 'signalMACD_period': 28}
```

```
     5
               : 3.15579781914034
          : {'tema_period': 28, 'fastMACD_period': 56, 'slowMACD_period':
9, 'signalMACD_period': 40}
     6
               : 11.83645401140871
          : {'tema_period': 14, 'fastMACD_period': 56, 'slowMACD_period':
39, 'signalMACD_period': 80}
     7
               : 1.9345626534129452
          : {'tema_period': 28, 'fastMACD_period': 56, 'slowMACD_period':
9, 'signalMACD_period': 40}
     8
               : 6.9880734382661664
          : {'tema_period': 28, 'fastMACD_period': 35, 'slowMACD_period':
39, 'signalMACD_period': 80}
     9
               : 4.357936768940624
          : {'tema_period': 28, 'fastMACD_period': 35, 'slowMACD_period':
39, 'signalMACD_period': 80}
     10
               : 8.796030349103075
          : {'tema_period': 14, 'fastMACD_period': 35, 'slowMACD_period':
23, 'signalMACD_period': 40}
     11
               : 5.280285173757677
          : {'tema_period': 28, 'fastMACD_period': 35, 'slowMACD_period':
9, 'signalMACD_period': 80}
     12
               : 2.266850813849936
          : {'tema_period': 28, 'fastMACD_period': 35, 'slowMACD_period':
39, 'signalMACD_period': 28}
```

[25]:
```
bt_df = signals_df.copy()
bt_df.columns = bt_df.columns.str.capitalize()
bt_df.rename(columns={'Date': 'Datetime'}, inplace=True)
bt_df["Datetime"] = pd.to_datetime(bt_df["Datetime"])
bt_df.set_index('Datetime', inplace=True)
bt_df = bt_df.sort_index()

bt = Backtest(bt_df, TechAnalysisStrategy, cash=1_000_000, commission=0.002,␣
 ↪exclusive_orders=True,margin=0.1)

#
stats = bt.run()
print(stats)
```

```
Start                     2020-10-16 00:00…
```

```
End                     2024-10-25 00:00…
Duration                 1470 days 00:00:00
Exposure Time [%]                58.240741
Equity Final [$]            1058434.746713
Equity Peak [$]             1130600.028731
Return [%]                       5.843475
Buy & Hold Return [%]          924.798157
Return (Ann.) [%]                1.422798
Volatility (Ann.) [%]           38.346073
Sharpe Ratio                     0.037104
Sortino Ratio                    0.061291
Calmar Ratio                      0.02426
Max. Drawdown [%]              -58.646877
Avg. Drawdown [%]              -19.427943
Max. Drawdown Duration     1404 days 00:00:00
Avg. Drawdown Duration      410 days 00:00:00
# Trades                              136
Win Rate [%]                     34.558824
Best Trade [%]                   31.213336
Worst Trade [%]                 -10.466349
Avg. Trade [%]                   0.041803
Max. Trade Duration          22 days 00:00:00
Avg. Trade Duration           5 days 00:00:00
Profit Factor                    1.113227
Expectancy [%]                   0.212715
SQN                               0.12271
_strategy               TechAnalysisStra…
_equity_curve                          …
_trades                      Size  Entr…
dtype: object
```

[26]:
```
bt.plot()
```

```
BokehDeprecationWarning: Passing lists of formats for DatetimeTickFormatter
scales was deprecated in Bokeh 3.0. Configure a single string format for each
scale
BokehDeprecationWarning: Passing lists of formats for DatetimeTickFormatter
scales was deprecated in Bokeh 3.0. Configure a single string format for each
scale
```

[26]: `GridPlot(id='p3902', …)`

, , .

TA-lib, Optuna.

, ,

backtest

[27]:
```
df = yf.download("BTC-USD", start='2020-01-01', interval='1d').
  ↪drop(columns=['Adj Close'])
df = df.reset_index()
df.columns = df.columns.str.lower()
df
```

```
[**********************100%**********************]  1 of 1 completed
```

[27]:

	date	close	high	low	open	volume
0	2020-01-01 00:00:00+00:00	7200.174316	7254.330566	7174.944336	7194.892090	18565664997
1	2020-01-02 00:00:00+00:00	6985.470215	7212.155273	6935.270020	7202.551270	20802083465
2	2020-01-03 00:00:00+00:00	7344.884277	7413.715332	6914.996094	6984.428711	28111481032
3	2020-01-04 00:00:00+00:00	7410.656738	7427.385742	7309.514160	7345.375488	18444271275
4	2020-01-05 00:00:00+00:00	7411.317383	7544.497070	7400.535645	7410.451660	19725074095
…	…	…	…	…	…	…
1757	2024-10-23 00:00:00+00:00	66432.195312	67402.742188	65188.035156	67362.375000	32263980353
1758	2024-10-24 00:00:00+00:00	68161.054688	68798.960938	66454.101562	66653.703125	31414428647
1759	2024-10-25 00:00:00+00:00	66642.414062	68722.156250	65521.792969	68165.296875	41469984306
1760	2024-10-26 00:00:00+00:00	67014.695312	67317.921875	66360.593750	66628.734375	19588098156
1761	2024-10-27 00:00:00+00:00	67513.171875	67865.539062	66854.335938	67016.796875	14976246784

[1762 rows x 6 columns]

[28]:
```
linear_plot(df, 'BTC/USD')
```

[29]:
```
bt_df_sma = df.copy()
bt_df_sma.columns = bt_df_sma.columns.str.capitalize()
bt_df_sma.rename(columns={'Date': 'Datetime'}, inplace=True)
bt_df_sma["Datetime"] = pd.to_datetime(bt_df_sma["Datetime"])
bt_df_sma.set_index('Datetime', inplace=True)
bt_df_sma
```

[29]:

Datetime	Close	High	Low	\
2020-01-01 00:00:00+00:00	7200.174316	7254.330566	7174.944336	

```
2020-01-02 00:00:00+00:00   6985.470215   7212.155273   6935.270020
2020-01-03 00:00:00+00:00   7344.884277   7413.715332   6914.996094
2020-01-04 00:00:00+00:00   7410.656738   7427.385742   7309.514160
2020-01-05 00:00:00+00:00   7411.317383   7544.497070   7400.535645
…                                    …             …             …
2024-10-23 00:00:00+00:00  66432.195312  67402.742188  65188.035156
2024-10-24 00:00:00+00:00  68161.054688  68798.960938  66454.101562
2024-10-25 00:00:00+00:00  66642.414062  68722.156250  65521.792969
2024-10-26 00:00:00+00:00  67014.695312  67317.921875  66360.593750
2024-10-27 00:00:00+00:00  67513.171875  67865.539062  66854.335938

                                     Open       Volume
Datetime
2020-01-01 00:00:00+00:00   7194.892090  18565664997
2020-01-02 00:00:00+00:00   7202.551270  20802083465
2020-01-03 00:00:00+00:00   6984.428711  28111481032
2020-01-04 00:00:00+00:00   7345.375488  18444271275
2020-01-05 00:00:00+00:00   7410.451660  19725074095
…                                    …            …
2024-10-23 00:00:00+00:00  67362.375000  32263980353
2024-10-24 00:00:00+00:00  66653.703125  31414428647
2024-10-25 00:00:00+00:00  68165.296875  41469984306
2024-10-26 00:00:00+00:00  66628.734375  19588098156
2024-10-27 00:00:00+00:00  67016.796875  14976246784

[1762 rows x 5 columns]
```

[30]:
```
bt_sma = Backtest(bt_df_sma, SmaCross, cash=1_000_000, commission=.002,␣
 ↪exclusive_orders=True)
stats = bt_sma.run()
print(stats)
```

```
Start                     2020-01-01 00:00…
End                       2024-10-27 00:00…
Duration                   1761 days 00:00:00
Exposure Time [%]                  96.935301
Equity Final [$]               723419.882398
Equity Peak [$]               6130124.633664
Return [%]                       -27.658012
Buy & Hold Return [%]            837.660241
Return (Ann.) [%]                 -6.48687
Volatility (Ann.) [%]             59.927696
Sharpe Ratio                            0.0
Sortino Ratio                           0.0
Calmar Ratio                            0.0
Max. Drawdown [%]                -91.929825
Avg. Drawdown [%]                 -8.820616
Max. Drawdown Duration     1388 days 00:00:00
```

```
Avg. Drawdown Duration       77 days 00:00:00
# Trades                                   92
Win Rate [%]                        33.695652
Best Trade [%]                     181.445437
Worst Trade [%]                    -22.985365
Avg. Trade [%]                      -0.362041
Max. Trade Duration         105 days 00:00:00
Avg. Trade Duration          19 days 00:00:00
Profit Factor                        1.259071
Expectancy [%]                       1.288973
SQN                                 -0.071528
_strategy                            SmaCross
_equity_curve                               …
_trades                       Size  EntryB…
dtype: object
```

[31]:
```
bt_sma.plot()
```

```
BokehDeprecationWarning: Passing lists of formats for DatetimeTickFormatter
scales was deprecated in Bokeh 3.0. Configure a single string format for each
scale
BokehDeprecationWarning: Passing lists of formats for DatetimeTickFormatter
scales was deprecated in Bokeh 3.0. Configure a single string format for each
scale
```

[31]: GridPlot(id='p5061', …)

.

[32]:
```
stats = bt_sma.optimize(n1=range(5, 30, 5),
                        n2=range(10, 70, 5),
                        maximize='Equity Final [$]',
                        constraint=lambda param: param.n1 < param.n2)
print(stats)
```

```
Start                     2020-01-01 00:00…
End                       2024-10-27 00:00…
Duration                   1761 days 00:00:00
Exposure Time [%]                   96.594779
Equity Final [$]                 8244386.848148
Equity Peak [$]                 16240972.673984
Return [%]                        724.438685
Buy & Hold Return [%]             837.660241
Return (Ann.) [%]                  54.804324
Volatility (Ann.) [%]              96.857104
Sharpe Ratio                        0.565827
Sortino Ratio                       1.471784
Calmar Ratio                        0.982149
Max. Drawdown [%]                 -55.800403
```

```
Avg. Drawdown [%]                  -9.934571
Max. Drawdown Duration      681 days 00:00:00
Avg. Drawdown Duration       34 days 00:00:00
# Trades                                   42
Win Rate [%]                       42.857143
Best Trade [%]                     348.01875
Worst Trade [%]                   -20.442894
Avg. Trade [%]                     5.168614
Max. Trade Duration         194 days 00:00:00
Avg. Trade Duration          41 days 00:00:00
Profit Factor                       3.18224
Expectancy [%]                     11.164129
SQN                                 0.755683
_strategy                 SmaCross(n1=10,n…
_equity_curve                             …
_trades                      Size  EntryB…
dtype: object
```

[33]:
```
stats._strategy
```

[33]:
```
<Strategy SmaCross(n1=10,n2=45)>
```

[34]:
```
bt_sma.plot()
```

```
BokehDeprecationWarning: Passing lists of formats for DatetimeTickFormatter
scales was deprecated in Bokeh 3.0. Configure a single string format for each
scale
BokehDeprecationWarning: Passing lists of formats for DatetimeTickFormatter
scales was deprecated in Bokeh 3.0. Configure a single string format for each
scale
```

[34]:
```
GridPlot(id='p6187', …)
```