In [43]:
```
import numpy as np
import pandas as pd
import datetime
import yfinance as yf
#import mibian
from py_vollib.black_scholes import implied_volatility
import matplotlib.pyplot as plt

import warnings
warnings.simplefilter("ignore")
```

1. Read and Process the data

In [625…
```
top5 = pd.read_csv("~/Downloads/2022_02_28_2023_02_28_top5.csv") #xq4gifz
ticker_symbol = ['AAPL','MSFT','UNH','GS','HD']
start_date = '2022-02-28'
end_date='2023-03-01'

def stock_option(top5, ticker_symbol, start_date = start_date, end_date =
    ticker_data = [yf.Ticker(i) for i in ticker_symbol]
    ticker_df = []
    for i,x in zip(ticker_symbol,ticker_data):
        data = x.history(start=start_date, end=end_date)[['Close','Divide
        data['symbol'] = i[:2]
        ticker_df.append(data)
    ticker_df = pd.concat(ticker_df,axis=0).reset_index()
    ticker_df['Date'] = ticker_df['Date'].dt.date
    ticker_df['Date'] = pd.to_datetime(ticker_df['Date'])

    top5['exdate'] = pd.to_datetime(top5['exdate'])
    top5['date'] = pd.to_datetime(top5['date'])
    top5['Maturity'] = (top5['exdate']-top5['date']).dt.days/360
    top5['symbol_2'] = top5['symbol'].apply(lambda x : x[:2])
    top5['cp_flag'] = top5['cp_flag'].replace(['C', 'P'], ['call','put'])
    top5['strike_price'] = top5['strike_price']/1000
    top5['price'] = (top5['best_offer'] + top5['best_bid']) / 2

    top5 = pd.merge(top5,ticker_df,left_on=['date','symbol_2'],right_on=[
    return top5

top5 = stock_option(top5, ticker_symbol)
```

In [627…
```
top5
```

Out[627…

	secid	date	symbol_x	symbol_flag	exdate	last_date	cp_flag
0	101594	2022-02-28	AAPL 220311P135000	1	2022-03-11	2022-02-28	put
1	101594	2022-02-28	AAPL 220311P139000	1	2022-03-11	2022-02-28	put
2	101594	2022-02-28	AAPL 220311P140000	1	2022-03-11	2022-02-28	put
3	101594	2022-02-28	AAPL 220311P141000	1	2022-03-11	2022-02-28	put
4	101594	2022-02-28	AAPL 220311P142000	1	2022-03-11	2022-02-28	put
...	...	...	...	...	...	...	...
2230243	111469	2023-02-28	UNH 250117P740000	1	2025-01-17	NaN	put
2230244	111469	2023-02-28	UNH 250117P760000	1	2025-01-17	NaN	put
2230245	111469	2023-02-28	UNH 250117P780000	1	2025-01-17	NaN	put
2230246	111469	2023-02-28	UNH 250117P800000	1	2025-01-17	NaN	put
2230247	111469	2023-02-28	UNH 250117P820000	1	2025-01-17	2022-11-22	put

2230248 rows × 45 columns

In [324…

```
# try to calculate the implied volatility which is NA
is_na = top5['impl_volatility'].isna()
# just try to use the ffill and bfill volatility to use as guess volatili
top5['impl_volatility'] = top5.groupby(['cp_flag','symbol_y'])['impl_vola
top5 = top5[is_na] # only calculate the NA
top5.to_csv("~/Downloads/option_before.csv")
# RUN the R Code
```

In [325…

```
# use the data from R Code
data = pd.read_csv("~/Downloads/option.csv")
```

```
data[~data['ImpliedVolatility'].isna()]
# not too much but has culculated some
# and maybe we can just ffill it if they are really close
```

Out[325…

	X	secid	date	symbol_x	symbol_flag	exdate	last_date	c
0	0	101594	2023-02-01	AAPL 230203C100000	1	2023-02-03	2023-02-01	
1	1	101594	2023-02-01	AAPL 230203C105000	1	2023-02-03	2023-02-01	
2	2	101594	2023-02-01	AAPL 230203C109000	1	2023-02-03	NaN	
3	3	101594	2023-02-01	AAPL 230203C110000	1	2023-02-03	2023-02-01	
4	4	101594	2023-02-01	AAPL 230203C111000	1	2023-02-03	2023-01-25	
…	…	…	…	…	…	…	…	
23291	172142	111469	2023-02-28	UNH 230406C610000	1	2023-04-06	NaN	
23292	172143	111469	2023-02-28	UNH 230406C620000	1	2023-04-06	NaN	
23293	172144	111469	2023-02-28	UNH 230406C630000	1	2023-04-06	NaN	
23313	172200	111469	2023-02-28	UNH 230421C250000	1	2023-04-21	NaN	
23314	172201	111469	2023-02-28	UNH 230421C260000	1	2023-04-21	NaN	

6226 rows × 47 columns

In [557…

```
# build staddle of the top 5
top5_copy = top5.groupby(['symbol_y','Date','exdate','strike_price']).app
top5_copy['avg_price'] = top5_copy['price'].groupby(['symbol_y','exdate',
top5_copy
```

Out[557…

			price	IV	avg_price
symbol_y	Date	exdate	strike_price		
AA	2023-02-01	2023-02-03	50.0	95.380	NaN	NaN
			55.0	90.380	NaN	NaN
			60.0	85.380	NaN	NaN
			65.0	80.430	NaN	NaN
			70.0	75.430	NaN	NaN
…	…	…	…	…	…	…
UN	2023-02-28	2025-01-17	740.0	271.525	0.213272	0.019812
			760.0	290.775	0.214786	0.019369
			780.0	309.550	0.214283	0.017671
			800.0	327.775	0.210988	0.015963
			820.0	346.620	0.204909	0.017122

86425 rows × 3 columns

In [576…

```
# read the index option data
options_data = pd.read_csv("~/Downloads/shril26ejyyyoum8.csv")
futures_data = yf.download("YM=F", start="2022-02-28", end="2023-02-28")

options_data['exdate'] = pd.to_datetime(options_data['exdate'])
options_data['date'] = pd.to_datetime(options_data['date'])
options_data['time_to_expiry'] = (options_data['exdate'] - options_data['
options_data = options_data.rename(columns={'date':'Date'})
options_data = options_data[['Date','exdate','cp_flag','strike_price','ti

indx_data = pd.merge(options_data,futures_data,on = 'Date')
indx_data['strike_price'] = indx_data['strike_price']/10
indx_data['strike_distance'] = np.abs(indx_data['Adj Close'] - indx_data[
indx_data_copy = indx_data
indx_data['option_close'] = (indx_data['best_bid'] + indx_data['best_offe
indx_data
```

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

Out[576…

	Date	exdate	cp_flag	strike_price	time_to_expiry	best_bid	best_offer
0	2023-02-01	2023-02-03	C	32900.0	2	9.35	14.30
1	2023-02-01	2023-02-03	C	33000.0	2	8.40	13.35
2	2023-02-01	2023-02-03	C	33100.0	2	7.40	12.35
3	2023-02-01	2023-02-03	C	33200.0	2	8.70	9.25
4	2023-02-01	2023-02-03	C	33300.0	2	7.75	8.30
...	...	...	...	...	...	...	...
22595	2023-02-27	2025-12-19	P	52000.0	1026	137.00	153.00
22596	2023-02-27	2025-12-19	P	54000.0	1026	154.00	170.00
22597	2023-02-27	2025-12-19	P	56000.0	1026	171.00	187.00
22598	2023-02-27	2025-12-19	P	58000.0	1026	189.00	205.00
22599	2023-02-27	2025-12-19	P	60000.0	1026	202.00	226.00

22600 rows × 16 columns

In [585…

```
# build staddle of the index
indx_data_copy = indx_data.groupby(['Date','exdate','strike_price']).appl
indx_data_copy['avg_price'] = indx_data_copy['price'].groupby(['exdate','
indx_data_copy
```

Out[585…

Date	exdate	strike_price	price	IV	avg_price
2023-02-01	**2023-02-03**	**32900.0**	11.895	0.358665	NaN
		33000.0	10.960	0.344616	NaN
		33100.0	9.980	0.331070	NaN
		33200.0	9.120	0.293400	NaN
		33300.0	8.220	0.293689	NaN
...	**...**	**...**	...	...	...
2023-02-27	**2025-12-19**	**52000.0**	149.800	NaN	0.0
		54000.0	166.800	NaN	0.0
		56000.0	183.800	NaN	0.0
		58000.0	201.800	NaN	0.0
		60000.0	218.800	NaN	0.0

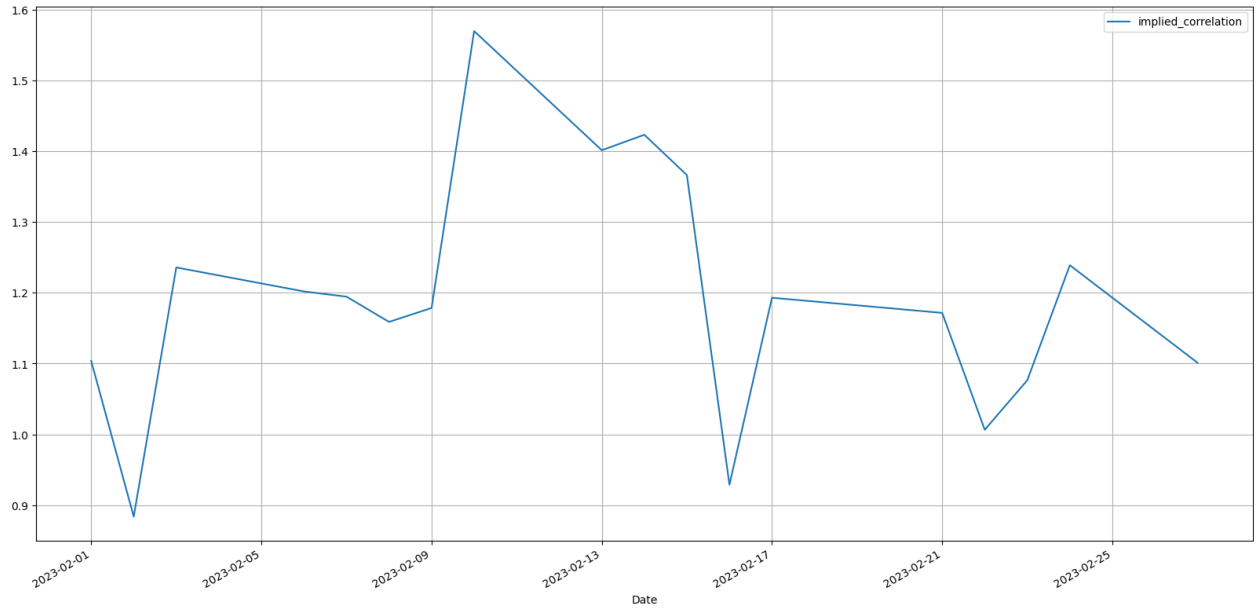
11300 rows × 3 columns

2. Get the dirty correlation and the position

In [563…

```
def implied_dirty_correlation():
    dowjones_IV = indx_data_copy.groupby(['Date'])['IV'].mean().to_frame(
    top5_IV = top5_copy.groupby(['Date'])['IV'].mean().to_frame()
    return (dowjones_IV/top5_IV)**2

df = implied_dirty_correlation()
df = df.rename(columns={'IV': 'implied_correlation'})
df.plot( grid = True , figsize = (20,10))
plt.show()
```



In [564…

```
#-1 short the index and long the stocks
#1 long the index and short the stocks
df['position'] = df.apply(lambda x: -1 if x['implied_correlation']>1 else
df
```

Out[564…

Date	implied_correlation	position
2023-02-01	1.103835	-1
2023-02-02	0.883795	0
2023-02-03	1.235879	-1
2023-02-06	1.202000	-1
2023-02-07	1.194604	-1
2023-02-08	1.158948	-1
2023-02-09	1.178621	-1
2023-02-10	1.569823	-1
2023-02-13	1.401499	-1
2023-02-14	1.423327	-1
2023-02-15	1.366365	-1
2023-02-16	0.928912	0
2023-02-17	1.193092	-1
2023-02-21	1.171675	-1
2023-02-22	1.006552	-1
2023-02-23	1.076520	-1
2023-02-24	1.239030	-1
2023-02-27	1.101245	-1
2023-02-28	NaN	0

In [587…

```
top5_return = pd.merge(top5_copy,df,on="Date")
indx_return = pd.merge(indx_data_copy,df,on="Date")
```

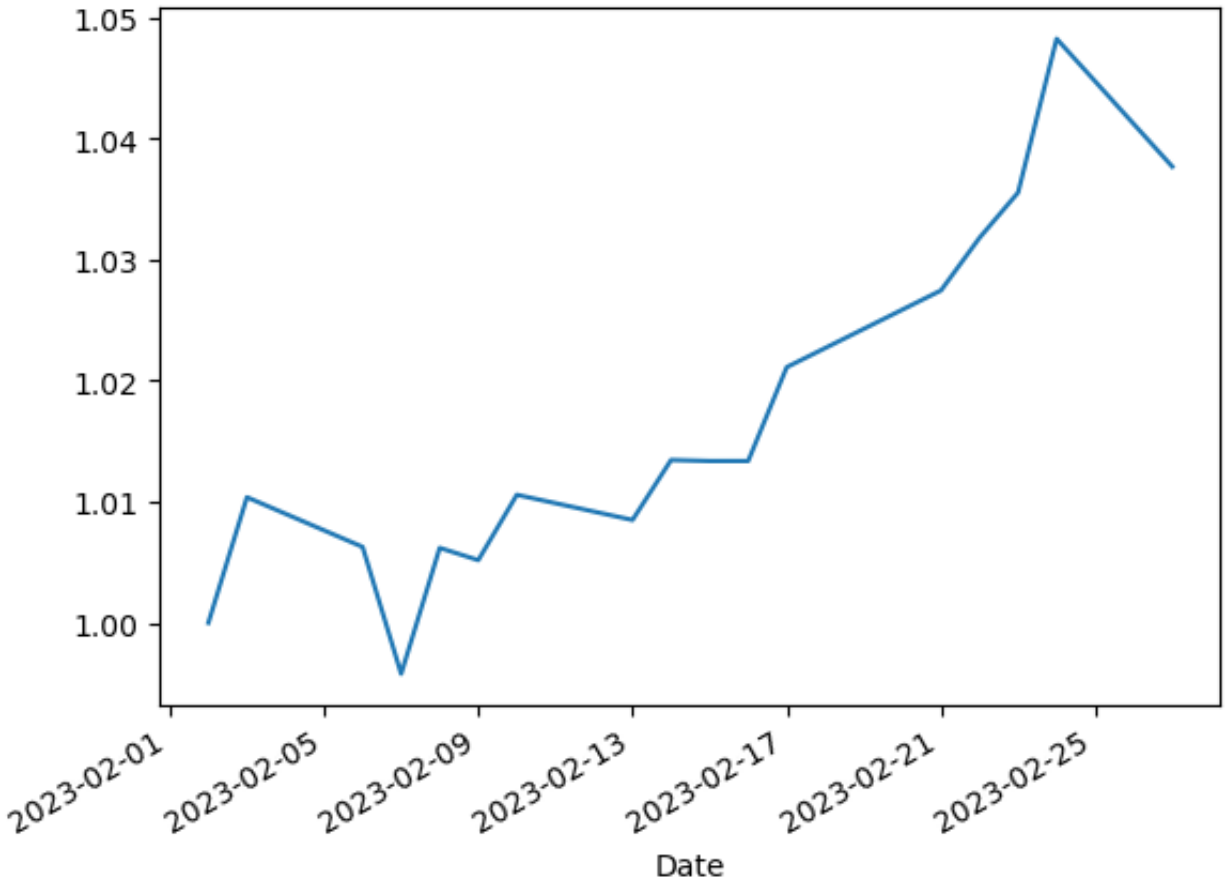
3. Get the Return

In [607…

```
strategy_return = (top5_return['avg_price'] * top5_return['position']).gr
np.cumprod(1+strategy_return).plot()
```

Out[607…

```
<Axes: xlabel='Date'>
```



4.

In [604…

```
def performance(strategy_returns, benchmark_returns, window = 5):
    # Annualized return
    annual_return = strategy_returns['returns'].mean() * 252

    # Annualized volatility
    annual_volatility = strategy_returns['returns'].std() * np.sqrt(252)

    # Sharpe ratio, assuming risk-free rate is 0 for simplicity
    sharpe_ratio = annual_return / annual_volatility

    # Max drawdown in one day
    max_drawdown = strategy_returns['returns'].min()

    # Cumulative returns with a benchmark plot
    strategy_returns['cumulative'] = (1 + strategy_returns['returns']).cu
    benchmark_returns['cumulative'] = (1 + benchmark_returns['dow_jones_r

    # Rolling volatility and rolling sharpe ratio plot
    strategy_returns['rolling_vol'] = strategy_returns['returns'].rolling
    strategy_returns['rolling_sharpe'] = strategy_returns['returns'].roll

    # Plot cumulative returns
    plt.figure(figsize=(12, 6))
```

```
    plt.plot(strategy_returns['cumulative'], label='Strategy')
    plt.plot(benchmark_returns['cumulative'], label='Benchmark')
    plt.legend()
    plt.title('Cumulative Returns')
    plt.show()

    # Plot rolling volatility
    plt.figure(figsize=(12, 6))
    plt.plot(strategy_returns['rolling_vol'], label='Rolling Volatility')
    plt.legend()
    plt.title('Rolling Volatility (1 Year)')
    plt.show()

    # Plot rolling Sharpe ratio
    plt.figure(figsize=(12, 6))
    plt.plot(strategy_returns['rolling_sharpe'], label='Rolling Sharpe Ra
    plt.legend()
    plt.title('Rolling Sharpe Ratio (1 Year)')
    plt.show()

    # Regression daily return on daily return of Dow Jones, portfolio bet
    import statsmodels.api as sm

    # Assume 'dow_jones_returns' is a column in your DataFrame
    X = sm.add_constant(benchmark_returns['dow_jones_returns'])  # Adds a
    est = sm.OLS(strategy_returns['returns'], X).fit()
    strategy_beta = est.params['dow_jones_returns']

    # Performance alpha (return - beta * Dow Jones return)
    strategy_alpha = annual_return - (strategy_beta * benchmark_returns['

    print(f'Annual Return: {annual_return}')
    print(f'Annual Volatility: {annual_volatility}')
    print(f'Sharpe Ratio: {sharpe_ratio}')
    print(f'Max Drawdown: {max_drawdown}')
    print(f'Portfolio Beta: {strategy_beta}')
    print(f'Alpha: {strategy_alpha}')
```

In [621…

```
pd.read_csv("~/Downloads/strategy_return_marchtojuly_2020.csv").rename(co
```

Out[621…

	Date	returns
0	2020-01-02	NaN
1	2020-01-03	-0.016376
2	2020-01-06	0.010865
3	2020-01-07	0.004926
4	2020-01-08	0.011634
...	...	...
120	2020-06-24	-0.036338
121	2020-06-25	0.038167
122	2020-06-26	-0.016436
123	2020-06-29	0.015947
124	2020-06-30	0.022729

125 rows × 2 columns

In [615…

```
strategy_return
dow_jones = yf.Ticker("^DJI").history(start=start_date, end=end_date)
benchmark_returns = pd.DataFrame(dow_jones['Close'].pct_change().rename('
strategy_returns = pd.DataFrame(strategy_return.rename('returns'))
is_na = (benchmark_returns['dow_jones_returns'].isna())|(strategy_returns
performance(strategy_returns[~is_na], benchmark_returns[~is_na])
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[615], line 4
      2 benchmark_returns = pd.DataFrame(dow_jones['Close'].pct_change().r
ename('dow_jones_returns'))
      3 strategy_returns = pd.DataFrame(strategy_return.rename('returns'))
----> 4 is_na = (benchmark_returns['dow_jones_returns'].isna())|(strategy_
returns['returns'].isna())
      5 performance(strategy_returns[~is_na], benchmark_returns[~is_na])

File ~/anaconda3/lib/python3.11/site-packages/pandas/core/ops/common.py:76
, in _unpack_zerodim_and_defer.<locals>.new_method(self, other)
     72             return NotImplemented
     74 other = item_from_zerodim(other)
---> 76 return method(self, other)

File ~/anaconda3/lib/python3.11/site-packages/pandas/core/arraylike.py:78,
in OpsMixin.__or__(self, other)
     76 @unpack_zerodim_and_defer("__or__")
     77 def __or__(self, other):
```

```
---> 78     return self._logical_method(other, operator.or_)

File ~/anaconda3/lib/python3.11/site-packages/pandas/core/series.py:6105,
in Series._logical_method(self, other, op)
   6103 def _logical_method(self, other, op):
   6104     res_name = ops.get_op_result_name(self, other)
-> 6105     self, other = self._align_for_op(other, align_asobject=True)
   6107     lvalues = self._values
   6108     rvalues = extract_array(other, extract_numpy=True, extract_ran
ge=True)

File ~/anaconda3/lib/python3.11/site-packages/pandas/core/series.py:6144,
in Series._align_for_op(self, right, align_asobject)
   6141             left = left.astype(object)
   6142             right = right.astype(object)
-> 6144         left, right = left.align(right, copy=False)
   6146 return left, right

File ~/anaconda3/lib/python3.11/site-packages/pandas/core/generic.py:10441
, in NDFrame.align(self, other, join, axis, level, copy, fill_value, metho
d, limit, fill_axis, broadcast_axis)
  10428     left, _right, join_index = self._align_frame(
  10429         other,
  10430         join=join,
   (...)
  10437         fill_axis=fill_axis,
  10438     )
  10440 elif isinstance(other, ABCSeries):
> 10441     left, _right, join_index = self._align_series(
  10442         other,
  10443         join=join,
  10444         axis=axis,
  10445         level=level,
  10446         copy=copy,
  10447         fill_value=fill_value,
  10448         method=method,
  10449         limit=limit,
  10450         fill_axis=fill_axis,
  10451     )
  10452 else:  # pragma: no cover
  10453     raise TypeError(f"unsupported type: {type(other)}")

File ~/anaconda3/lib/python3.11/site-packages/pandas/core/generic.py:10558
, in NDFrame._align_series(self, other, join, axis, level, copy, fill_valu
e, method, limit, fill_axis)
  10556     join_index, lidx, ridx = None, None, None
  10557 else:
> 10558     join_index, lidx, ridx = self.index.join(
  10559         other.index, how=join, level=level, return_indexers=True
  10560     )
  10562 if is_series:
  10563     left = self._reindex_indexer(join_index, lidx, copy)
```

```
File ~/anaconda3/lib/python3.11/site-packages/pandas/core/indexes/base.py:
279, in _maybe_return_indexers.<locals>.join(self, other, how, level, retu
rn_indexers, sort)
    269 @functools.wraps(meth)
    270 def join(
    271     self,
   (...)
    277     sort: bool = False,
    278 ):
--> 279     join_index, lidx, ridx = meth(self, other, how=how, level=leve
l, sort=sort)
    280     if not return_indexers:
    281         return join_index

File ~/anaconda3/lib/python3.11/site-packages/pandas/core/indexes/base.py:
4595, in Index.join(self, other, how, level, return_indexers, sort)
   4592 if isinstance(self, ABCDatetimeIndex) and isinstance(other, ABCDat
etimeIndex):
   4593     if (self.tz is None) ^ (other.tz is None):
   4594         # Raise instead of casting to object below.
-> 4595         raise TypeError("Cannot join tz-naive with tz-aware Dateti
meIndex")
   4597 if not self._is_multi and not other._is_multi:
   4598     # We have specific handling for MultiIndex below
   4599     pself, pother = self._maybe_downcast_for_indexing(other)

TypeError: Cannot join tz-naive with tz-aware DatetimeIndex
```

In [ ]: