**Scenario Analyse:**

import numpy as np

import pandas as pd

import yfinance as yf

import matplotlib.pyplot as plt

from collections import namedtuple, defaultdict

from scipy.stats import norm

\_riskfree = 0.01

def \_d1(S, K, T, sigma, r=.0, q=.0):

with np.errstate(divide='ignore'):

return (np.log(S/K) + (r - q + 0.5\*sigma\*\*2)\*T) / (sigma\*np.sqrt(T))

def \_d2(S, K, T, sigma, r=.0, q=.0):

return \_d1(S, K, T, sigma, r, q) - sigma\*np.sqrt(T)

def call(S, K, T, sigma, r=.0, q=.0):

if T <= 0: return max(0, S-K)

d1, d2 = \_d1(S, K, T, sigma, r, q), \_d2(S, K, T, sigma, r, q)

call = S \* np.exp(-q\*T) \* norm.cdf(d1, 0.0, 1.0) - K \* np.exp(-r\*T) \* norm.cdf(d2, 0.0, 1.0)

return call

def put(S, K, T, sigma, r=.0, q=.0):

if T<=0: return max(0, K-S)

d1, d2 = \_d1(S, K, T, sigma, r, q), \_d2(S, K, T, sigma, r, q)

put = K \* np.exp(-r\*T) \* norm.cdf(-d2, 0.0, 1.0) - S \* np.exp(-q\*T) \* norm.cdf(-d1, 0.0, 1.0)

return put

def Stock(ticker=None):

Class = namedtuple('Stock', ('ticker', ))

return Class(ticker.upper())

def Option(right=None, underlying=None, strike=None, lastTradeDate=None, multiplier=100):

Class = namedtuple('Option', ('right', 'underlying', 'strike', 'lastTradeDate', 'multiplier', ))

return Class(

right.upper(),

underlying.upper(),

float(round(strike, 2)),

pd.to\_datetime(lastTradeDate),

int(multiplier),

)

def get\_close(ticker):

stock = yf.Ticker(ticker)

df = stock.history(period="max")

df.index = df.index.tz\_localize(None)

close = df['Close']

return close

def run\_strategy(call\_offset, put\_offset):

return Strategy(

stock\_ticker='JPM',

ini\_fund=1e6,

days\_to\_maturity=30,

rebal\_freq=21,

call\_offset=call\_offset,

put\_offset=put\_offset,

).run(

start='2007-01-01',

end='2023-06-15',

).evaluate()

class Account:

def \_\_init\_\_(self, name, ini\_fund, stock\_prices, vix):

self.\_name = name

self.\_ini\_fund = ini\_fund

self.\_cash = 0

self.\_stock\_pos = defaultdict(int)

self.\_option\_pos = defaultdict(int)

self.\_stock\_prices = stock\_prices

self.\_vix = vix

self.\_dashboard = pd.DataFrame(columns=('Cash', 'Stock', 'Option', 'NAV'), dtype=float)

def stock\_price\_at(self, at, contract):

return self.\_stock\_prices[contract.ticker].loc[at]

def option\_price\_at(self, at, contract):

pricing = call if contract.right.upper()=='CALL' else put

price = pricing(

S=self.\_stock\_prices[contract.underlying].loc[at],

K=contract.strike,

T=(contract.lastTradeDate - at).days / 365,

sigma=self.\_vix.loc[at]/100,

r=\_riskfree,

)

return price

def deposit(self, amount):

self.\_cash += amount

def trade\_stock(self, at, contract, share):

price = self.\_stock\_prices[contract.ticker].loc[at]

self.\_cash -= price\*share

self.\_stock\_pos[contract] += share

def trade\_option(self, at, contract, share):

price = self.option\_price\_at(at, contract)

self.\_cash -= price\*share\*contract.multiplier

self.\_option\_pos[contract] += share

def close\_all\_option\_positions(self, at):

for contract,share in self.\_option\_pos.items():

amount = self.\_option\_pos[contract]

self.trade\_option(at, contract, -amount)

def net\_asset\_value(self, at):

cash\_val = self.\_cash

stock\_val = sum(self.stock\_price\_at(at, contract)\*share

for contract,share in self.\_stock\_pos.items())

option\_val = sum(self.option\_price\_at(at, contract)\*share\*contract.multiplier

for contract,share in self.\_option\_pos.items())

nav = cash\_val + stock\_val + option\_val

return cash\_val, stock\_val, option\_val, nav

def settlement(self, at):

for o in tuple(self.\_option\_pos.keys()):

if self.\_option\_pos[o] == 0:

del self.\_option\_pos[o]

vals = self.net\_asset\_value(at)

self.\_dashboard.loc[at] = vals

return vals

class Strategy:

def \_set\_args(self, kwargs):

if not hasattr(self, '\_args'): self.\_args = {}

for key,val in kwargs.items():

setattr(self, f'\_{key}', val)

self.\_args = {\*\*self.\_args, \*\*kwargs}

def get\_performance\_metric(self):

final\_nav = self.\_acc.\_dashboard['NAV'].iloc[-1]

return final\_nav

def \_\_init\_\_(self, \*\*kwargs):

self.\_set\_args(kwargs)

self.\_stock\_prices = {self.\_stock\_ticker:get\_close(self.\_stock\_ticker), }

self.\_vix = get\_close('^VIX')

self.\_acc = Account('Active', self.\_ini\_fund, self.\_stock\_prices, self.\_vix)

self.\_acc\_bm = Account('Benchmark', self.\_ini\_fund, self.\_stock\_prices, self.\_vix)

def run(self, \*\*kwargs):

self.\_set\_args(kwargs)

timeline = self.\_stock\_prices[self.\_stock\_ticker].loc[self.\_start:self.\_end].index

for i,today in enumerate(timeline):

last = self.\_stock\_prices[self.\_stock\_ticker].loc[today]

if i==0:

for acc in (self.\_acc, self.\_acc\_bm):

acc.deposit(self.\_ini\_fund)

self.\_ini\_stock\_position = round(self.\_ini\_fund/last)

acc.trade\_stock(today, Stock(self.\_stock\_ticker), +self.\_ini\_stock\_position)

if i%self.\_rebal\_freq==0:

self.\_acc.close\_all\_option\_positions(today)

put\_contract = Option(

right='put',

underlying=self.\_stock\_ticker,

strike=round(last\*(1-self.\_put\_offset), 2),

lastTradeDate=today+pd.Timedelta(days=self.\_days\_to\_maturity),

multiplier=100,

)

call\_contract = Option(

right='call',

underlying=self.\_stock\_ticker,

strike=round(last\*(1+self.\_call\_offset), 2),

lastTradeDate=today+pd.Timedelta(days=self.\_days\_to\_maturity),

multiplier=100,

)

amount = round(self.\_ini\_stock\_position/call\_contract.multiplier)

self.\_acc.trade\_option(today, put\_contract, +amount)

self.\_acc.trade\_option(today, call\_contract, -amount)

print(f'{i:5d} | {today.date()} ', end='')

for acc in (self.\_acc, self.\_acc\_bm):

nav = acc.settlement(today)[-1]

print(f' | {acc.\_name}: {nav:12,.2f}', end='')

print(end='\t\t\r')

return self

def evaluate(self, \*\*kwargs):

self.\_set\_args(kwargs)

df = pd.concat((self.\_acc.\_dashboard['NAV'], self.\_acc\_bm.\_dashboard['NAV']), axis=1)

df.columns=('Strategy', 'Benchmark', )

title = ', '.join((f'{k}={v}' for k, v in self.\_args.items()))

for name, ts in df.iteritems():

def metrics(name, ts):

def cal\_sharpe(ts, rf=0.025):

lndiffs = np.log(ts).diff()

mu = lndiffs.mean() \* 255

sigma = lndiffs.std() \* 252 \*\* .5

sharpe = (mu - rf) / sigma

return mu, sigma, sharpe

def cal\_drawdown(ts):

ts = np.log(ts)

run\_max = np.maximum.accumulate(ts)

end = (run\_max - ts).idxmax()

start = (ts.loc[:end]).idxmax()

low = ts.at[end]

high = ts.at[start]

dd = np.exp(low) / np.exp(high) - 1

pts = {'high': start, 'low': end}

duration = len(ts.loc[start:end])

return dd, pts, duration

mu, sigma, sharpe = cal\_sharpe(ts)

dd, pts, duration = cal\_drawdown(ts)

text = (f'\n{name} |mu:{mu:.2%} | sigma:{sigma:.2%} | sharpe:{sharpe:.2%} | '

f'drawdown:{dd:.2%} ({pts["high"].date()}-{pts["low"].date()}, {duration}d)')

return text

return self

if \_\_name\_\_ == "\_\_main\_\_":

call\_offsets = np.arange(0.10, 0.11, 0.01)

put\_offsets = np.arange(0.01, 0.11, 0.01)

results = []

for call\_offset in call\_offsets:

for put\_offset in put\_offsets:

result = run\_strategy(call\_offset, put\_offset)

performance\_metric = result.get\_performance\_metric()

results.append((call\_offset, put\_offset, performance\_metric))

results\_JPM = pd.DataFrame(results, columns=['call\_offset', 'put\_offset', 'performance\_metric'])