In [1]:

```
# Import required libraries
import pandas as pd
import numpy as np
from scipy.stats import norm
#ip install fredapi
from fredapi import Fred
#pip install fredpy
import fredpy as fp
#pip install QuantLib
import QuantLib as ql
import statistics
import math
import matplotlib.pyplot as plt
from tabulate import tabulate # conda install tabulate
import yfinance as yf
```

In [2]:

```
# Data Manipulation
import pandas as pd
from numpy import *
from datetime import timedelta
import yfinance as yf
from tabulate import tabulate

# Math & Optimization
from scipy.stats import norm
from scipy.optimize import fsolve

# Plotting
import matplotlib.pyplot as plt
import cufflinks as cf
cf.set_config_file(offline=True)
```

In [13]:

```
class BS:

    """
    This is a class for Options contract for pricing European options on stocks/inc

    Attributes:
        spot          : int or float
        strike        : int or float
        rate          : float
        dte           : int or float [days to expiration in number of years]
        volatility    : float
        callprice     : int or float [default None]
        putprice      : int or float [default None]
    """

    def __init__(self, spot, strike, rate, dte, volatility, callprice=None, putpric

        # Spot Price
        self.spot = spot

        # Option Strike
        self.strike = strike

        # Interest Rate
        self.rate = rate

        # Days To Expiration
        self.dte = dte
```

```python
        # Volatlity
        self.volatility = volatility

        # Callprice # mkt price
        self.callprice = callprice

        # Putprice # mkt price
        self.putprice = putprice

        # Utility
        self._a_ = self.volatility * self.dte**0.5

        if self.strike == 0:
            raise ZeroDivisionError('The strike price cannot be zero')
        else:
            self._d1_ = (log(self.spot / self.strike) + \
                     (self.rate + (self.volatility**2) / 2) * self.dte) / self._a_

        self._d2_ = self._d1_ - self._a_

        self._b_ = e**-(self.rate * self.dte)


        # The __dict__ attribute
        '''
        Contains all the attributes defined for the object itself. It maps the attr
        '''
        for i in ['callPrice', 'putPrice', 'callDelta', 'putDelta', 'callTheta', '
                  'callRho', 'putRho', 'vega', 'gamma', 'impvol']:
            self.__dict__[i] = None

        [self.callPrice, self.putPrice] = self._price()
        [self.callDelta, self.putDelta] = self._delta()
        [self.callTheta, self.putTheta] = self._theta()
        [self.callRho, self.putRho] = self._rho()
        self.vega = self._vega()
        self.gamma = self._gamma()
        self.impvol = self._impvol()

    # Option Price
    def _price(self):
        '''Returns the option price: [Call price, Put price]'''

        if self.volatility == 0 or self.dte == 0:
            call = maximum(0.0, self.spot - self.strike)
            put = maximum(0.0, self.strike - self.spot)
        else:
            call = self.spot * norm.cdf(self._d1_) - self.strike * e**(-self.rate 
                                                                  self.dte) *

            put = self.strike * e**(-self.rate * self.dte) * norm.cdf(-self._d2_)
                                                                   self.spot 
        return [call, put]

    # Option Delta
    def _delta(self):
        '''Returns the option delta: [Call delta, Put delta]'''

        if self.volatility == 0 or self.dte == 0:
            call = 1.0 if self.spot > self.strike else 0.0
            put = -1.0 if self.spot < self.strike else 0.0
        else:
            call = norm.cdf(self._d1_)
```

```
            put = -norm.cdf(-self._d1_)
        return [call, put]

    # Option Gamma
    def _gamma(self):
        '''Returns the option gamma'''
        return norm.pdf(self._d1_) / (self.spot * self._a_)

    # Option Vega
    def _vega(self):
        '''Returns the option vega'''
        if self.volatility == 0 or self.dte == 0:
            return 0.0
        else:
            return self.spot * norm.pdf(self._d1_) * self.dte**0.5 / 100

    # Option Theta
    def _theta(self):
        '''Returns the option theta: [Call theta, Put theta]'''
        call = -self.spot * norm.pdf(self._d1_) * self.volatility / (2 * self.dte*

        put = -self.spot * norm.pdf(self._d1_) * self.volatility / (2 * self.dte**
        return [call / 365, put / 365]

    # Option Rho
    def _rho(self):
        '''Returns the option rho: [Call rho, Put rho]'''
        call = self.strike * self.dte * self._b_ * norm.cdf(self._d2_) / 100
        put = -self.strike * self.dte * self._b_ * norm.cdf(-self._d2_) / 100

        return [call, put]

    # Option Implied Volatility
    def _impvol(self):
        '''Returns the option implied volatility'''
        if (self.callprice or self.putprice) is None:
            return self.volatility
        else:
            def f(sigma):
                option = BS(self.spot,self.strike,self.rate,self.dte,sigma)
                if self.callprice:
                    return option.callPrice - self.callprice # f(x) = BS_Call - Ma
                if self.putprice and not self.callprice:
                    return option.putPrice - self.putprice

            return maximum(1e-5, fsolve(f, 0.2)[0])

# Initialize option
option = BS(345,335,0.02,79/250,0.22,25)

header = ['Option Price', 'Delta', 'Gamma', 'Theta', 'Vega', 'Rho', 'IV']
table = [[option.callPrice, option.callDelta, option.gamma, option.callTheta, opti

print(tabulate(table,header))
```

```
  Option Price     Delta       Gamma       Theta      Vega       Rho        IV
--------------  --------  ----------  ----------  --------  --------  --------
       23.4591  0.637123  0.00879236  -0.0801439  0.727534  0.620461  0.241106
```

In [6]:

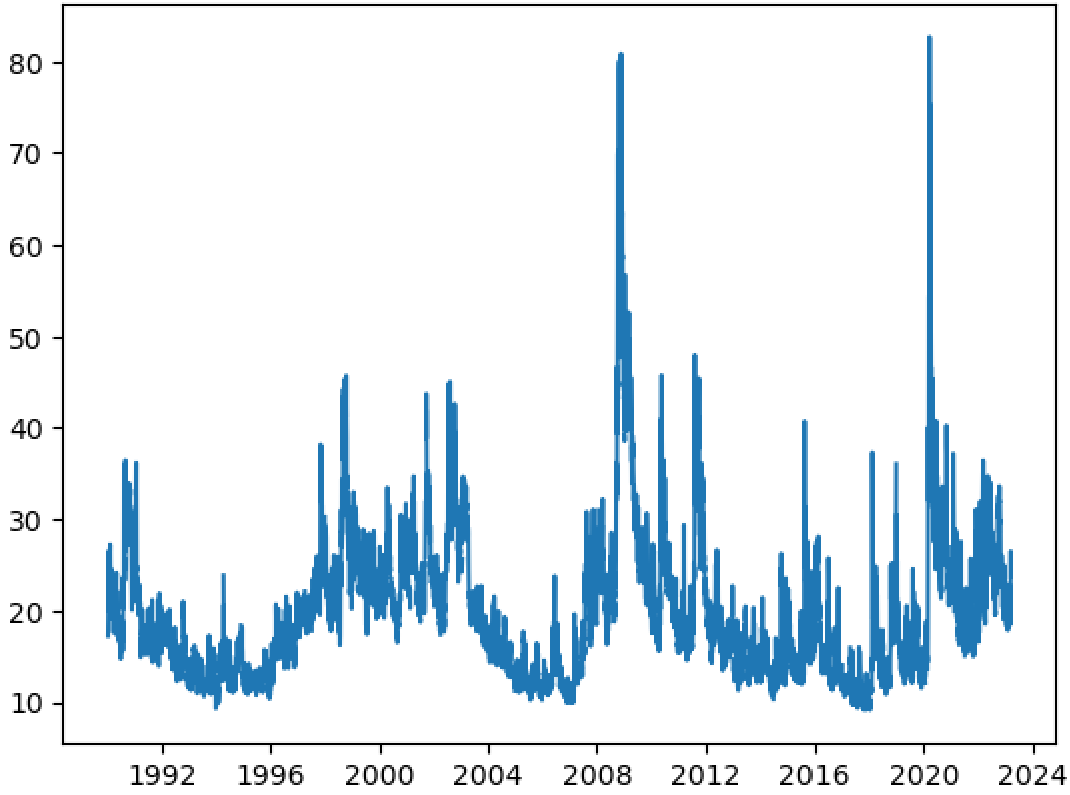
```
#Fred database to get SP500 historical prices
fred=Fred(api_key='8ac7604258e93947305e988327a4f7df')

vix=fred.get_series('VIXCLS', start_date='2018-03-01', end_date='2023-03-01')
```

```
vix
plt.plot(vix)
```

Out[6]:

```
[<matplotlib.lines.Line2D at 0x27e008cba60>]
```



In [14]:

```
median_vix=np.median(vix)
print(median_vix)


average_vix=np.mean(vix)
print(average_vix)
```

```
nan
19.67165013741184
```

In [15]:

```
# For ITM Call
c_spot = 340
c_strike = 330
c_rate = 0.03
c_dte =20/365
Initial_c_dte =30/365
End_c_dte=1/365
c_vol = 0.1966
c_spy_itm_opt = BS(c_spot,c_strike,c_rate,c_dte,c_vol)
print(f'Option Price with BS Model is {c_spy_itm_opt.callPrice:0.4f}')

Initial_Day_ITM_Call_price= BS(c_spot,c_strike, c_rate, Initial_c_dte, c_vol)
print(f'Option Price with BS Model is {Initial_Day_ITM_Call_price.callPrice:0.4f}'
End_Day_ITM_Call_price= BS(c_spot,c_strike,c_rate,End_c_dte,c_vol)
print(f'Option Price with BS Model is {End_Day_ITM_Call_price.callPrice:0.4f}')

Net_ITM_Short_Call=Initial_Day_ITM_Call_price.callPrice-End_Day_ITM_Call_price.cal
Net_ITM_Short_Call

ITM_Call_Theta=abs(c_spy_itm_opt.callTheta)*100

print(ITM_Call_Theta)
```

```
ITM_Call_Delta=-(c_spy_itm_opt.callDelta)
print(ITM_Call_Delta)
```

```
Option Price with BS Model is 12.8011
Option Price with BS Model is 14.1324
Option Price with BS Model is 10.0289
14.170592029715815
-0.7603463502280214
```

In [16]:

```
#For OTM Put
p_spot = c_spot
p_strike = 295
p_rate = c_rate
p_dte = c_dte
Initial_p_dte =30/365
End_p_dte=1/365
p_vol = c_vol
p_spy_otm_opt = BS(p_spot,p_strike,p_rate,p_dte,p_vol)
print(f'Option Price with BS Model is {p_spy_otm_opt.putPrice:0.4f}')

Initial_Day_OTM_Put_price= BS(p_spot,p_strike,p_rate,Initial_p_dte,p_vol)
print(f'Option Price with BS Model is {Initial_Day_OTM_Put_price.putPrice:0.4f}')

End_Day_OTM_Put_price= BS(p_spot,p_strike,p_rate,End_p_dte,p_vol)
print(f'Option Price with BS Model is {End_Day_OTM_Put_price.putPrice:0.4f}')

Net_OTM_Long_Put=End_Day_OTM_Put_price.putPrice-Initial_Day_OTM_Put_price.putPrice
Net_OTM_Long_Put
```

```
Option Price with BS Model is 0.0036
Option Price with BS Model is 0.0294
Option Price with BS Model is 0.0000
```

Out[16]:

```
-0.02937353106044127
```

In [17]:

```
#Sum up the above SC and BP
Time_premium=(Net_ITM_Short_Call+Net_OTM_Long_Put)*100
print(Time_premium)

OTM_Put_Theta=p_spy_otm_opt.putTheta*100
print(OTM_Put_Theta)

OTM_Put_Delta=p_spy_otm_opt.putDelta
print(OTM_Put_Delta)

SCBP_netDelta=ITM_Call_Delta+OTM_Put_Delta
print(SCBP_netDelta)
```

```
407.4054073043088
-0.10914608682152246
-0.0008342464857269179
-0.7611805967137484
```

In [18]:

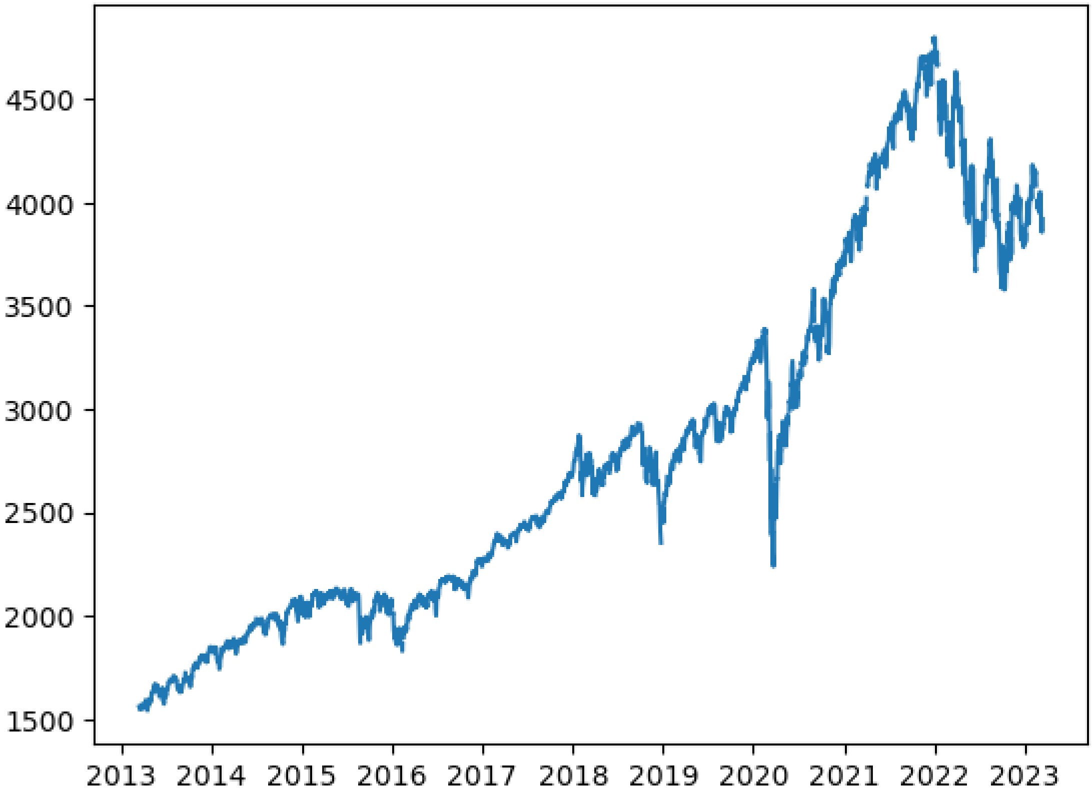
```
#Import SP500 data from the Fed's FRED database
sp500=fred.get_series('SP500', start_date='2013-03-01', end_date='2023-03-01')
sp500

monthly_sp500=sp500.resample('M').last()
print(monthly_sp500)
plt.plot(sp500)
```

```
2013-03-31    1569.19
2013-04-30    1597.57
2013-05-31    1630.74
2013-06-30    1606.28
2013-07-31    1685.73
               ...
2022-11-30    4080.11
2022-12-31    3839.50
2023-01-31    4076.60
2023-02-28    3970.15
2023-03-31    3919.29
Freq: M, Length: 121, dtype: float64
```

Out[18]:

```
[<matplotlib.lines.Line2D at 0x27e00b3e1f0>]
```



In [19]:

```
#For example, my other leg could be just represent 0.7 of the futures's Delta
mirror_partial_hedge_monthly_sp500=-monthly_sp500*0.70
print(mirror_partial_hedge_monthly_sp500)
number_to_add= Time_premium
```

```
2013-03-31   -1098.433
2013-04-30   -1118.299
2013-05-31   -1141.518
2013-06-30   -1124.396
2013-07-31   -1180.011
                ...
2022-11-30   -2856.077
2022-12-31   -2687.650
2023-01-31   -2853.620
2023-02-28   -2779.105
2023-03-31   -2743.503
Freq: M, Length: 121, dtype: float64
```

In [20]:

```
# use a loop to add the time premium
for i in range(len(mirror_partial_hedge_monthly_sp500)):
   mirror_partial_hedge_monthly_sp500[i] += number_to_add

# print the modified list
print(mirror_partial_hedge_monthly_sp500)
```

```
2013-03-31    -691.027593
2013-04-30    -710.893593
2013-05-31    -734.112593
2013-06-30    -716.990593
2013-07-31    -772.605593
                  ...
2022-11-30   -2448.671593
2022-12-31   -2280.244593
2023-01-31   -2446.214593
2023-02-28   -2371.699593
2023-03-31   -2336.097593
Freq: M, Length: 121, dtype: float64
```

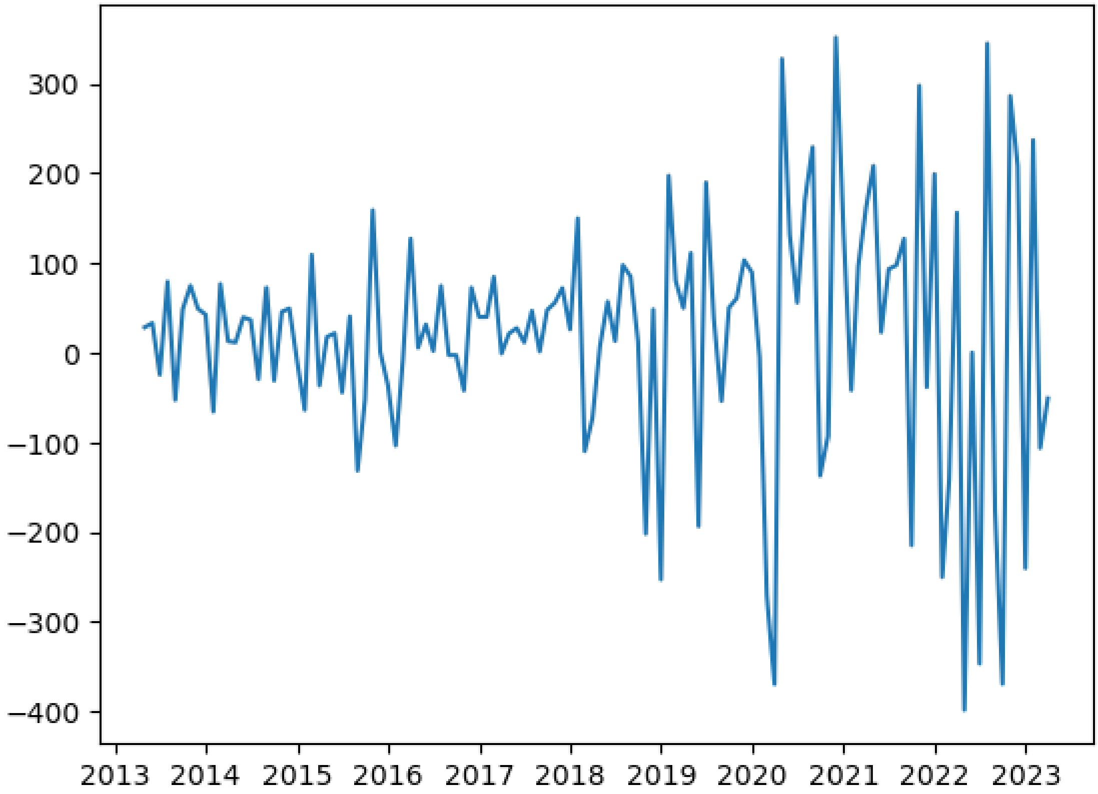
In [21]:

```
# print the modified list
print(mirror_partial_hedge_monthly_sp500)

monthly_sp500_fd =monthly_sp500.diff()
monthly_sp500_fd
plt.plot(monthly_sp500_fd)

cumul_PnL_sp500=(monthly_sp500_fd.cumsum())*100
cumul_PnL_sp500=cumul_PnL_sp500.dropna()
```

```
2013-03-31    -691.027593
2013-04-30    -710.893593
2013-05-31    -734.112593
2013-06-30    -716.990593
2013-07-31    -772.605593
                  ...
2022-11-30   -2448.671593
2022-12-31   -2280.244593
2023-01-31   -2446.214593
2023-02-28   -2371.699593
2023-03-31   -2336.097593
Freq: M, Length: 121, dtype: float64
```



In [22]:

```
mirror_time_monthly_leg_fd=mirror_partial_hedge_monthly_sp500.diff()
cumul_PnL_mirror_time_leg_fd=mirror_time_monthly_leg_fd.cumsum()*100
```

```
cumul_PnL_mirror_time_leg_fd=cumul_PnL_mirror_time_leg_fd.dropna()
```

In [23]:
```
#Capital and transaction cost set up
initial_margin=13200
Mirror_leg_using_options_margin=13200
Total_capital = initial_margin+Mirror_leg_using_options_margin
PnL_Index_Base=100
#For equity curve setup
PnL_month_net=monthly_sp500_fd+mirror_time_monthly_leg_fd
print(PnL_month_net)
```

```
2013-03-31       NaN
2013-04-30     8.514
2013-05-31     9.951
2013-06-30    -7.338
2013-07-31    23.835
               ...
2022-11-30    62.439
2022-12-31   -72.183
2023-01-31    71.130
2023-02-28   -31.935
2023-03-31   -15.258
Freq: M, Length: 121, dtype: float64
```

In [25]:
```
#plt.plot(PnL_month_net)
PnL_month_net_trend=PnL_month_net.cumsum()
print(PnL_month_net_trend)
#plt.plot(PnL_month_net_trend)
```

```
2013-03-31        NaN
2013-04-30      8.514
2013-05-31     18.465
2013-06-30     11.127
2013-07-31     34.962
               ...
2022-11-30    753.276
2022-12-31    681.093
2023-01-31    752.223
2023-02-28    720.288
2023-03-31    705.030
Freq: M, Length: 121, dtype: float64
```

In [27]:
```
PnL_month_net_trend_return=PnL_month_net_trend.pct_change()

equity_curve=PnL_Index_Base*(1+np.cumsum(PnL_month_net_trend_return))
print(equity_curve)
PnL_Index=pd.DataFrame(equity_curve, columns=['PnL Index'])
#PnL_Index.plot()
```

```
2013-03-31           NaN
2013-04-30           NaN
2013-05-31    216.878083
2013-06-30    177.138034
2013-07-31    391.346716
                 ...
2022-11-30    830.836251
2022-12-31    821.253708
2023-01-31    831.697215
2023-02-28    827.451799
2023-03-31    825.333479
Freq: M, Length: 121, dtype: float64
```

In [28]:
```
std_dev = np.std(PnL_month_net_trend_return)
print("Standard Deviation of Monthly Return:", std_dev)
```

```
Standard Deviation of Monthly Return: 0.2699015174673088
```

In [29]:
```
#Total Transaction cost including tax and slippage cost (TTC)
#Set at 10% of capital used
TTC=0.10
```

In [30]:
```
#Check for correct location
equity_curve[120]
equity_curve[2]
```

Out[30]:
```
216.87808315715463
```

In [31]:
```
raw_return=(equity_curve[120]-equity_curve[2])/equity_curve[2]
raw_return
```

Out[31]:
```
2.805518138621236
```

In [32]:
```
#number of years
noy=2023-2013
#annualzied power
ap=1/noy
```

In [33]:
```
Final_value=Total_capital*(1+raw_return)
Final_value
Initial_value=Total_capital
Initial_value
annualized_return=((Final_value/Initial_value)**ap-1)*(1-TTC)
annualized_return
```

Out[33]:
```
0.12868851020626698
```

In [34]:
```
portfolio_return=((Final_value-Initial_value)/Initial_value)*(1-TTC)
portfolio_return
```

Out[34]:
```
2.524966324759113
```

In [35]:
```
#risk free rate for Sharpe ratio
# https://fred.stlouisfed.org/series/FEDFUNDS
#My rfr is a weighted average from the last ten years
risk_free_rate=0.4*0.004+0.2*0.015+0.2*0.001+0.2*0.025
risk_free_rate
```

Out[35]:
```
0.0098
```

In [36]:
```
#Sharpe ratio
excess_return =portfolio_return -risk_free_rate
excess_return
Annualized_sharpe_ratio = excess_return /std_dev*math.sqrt(12)
print("Annualized Sharpe Ratio:", Annualized_sharpe_ratio)
```

```
Annualized Sharpe Ratio: 32.2813736273026
```

In [37]:
```
#******Plotting setup*****#
# Generate some data
PnL_Index.index=pd.to_datetime(PnL_Index.index)
PnL_Index.index
Date = PnL_Index.index
Date
y1 =PnL_Index
y1
```

```
y2 = monthly_sp500
y2
```

Out[37]:

```
2013-03-31    1569.19
2013-04-30    1597.57
2013-05-31    1630.74
2013-06-30    1606.28
2013-07-31    1685.73
               ...
2022-11-30    4080.11
2022-12-31    3839.50
2023-01-31    4076.60
2023-02-28    3970.15
2023-03-31    3919.29
Freq: M, Length: 121, dtype: float64
```

In [38]:

```
# Create the plot and set the first y-axis (left)
fig, ax1 = plt.subplots()
plt.xticks(rotation=90)
ax1.plot(Date, y1, 'b-')
ax1.set_xlabel('Date')
ax1.set_ylabel('PnL', color='b')
ax1.tick_params('y', colors='b')

# Set the second y-axis (right)
ax2 = ax1.twinx()
ax2.plot(Date, y2, 'k.')
ax2.set_ylabel('SP500', color='k')
ax2.tick_params('y', colors='k')

# Show the plot
plt.title('PnL vs SP500')
plt.show()
```