```
In [1]:  import pandas as pd
         import matplotlib.pyplot as plt
         import numpy as np
```

```
In [2]:  # Load the Excel file
         excel_file = pd.ExcelFile('E:\Derivatives Trading\TAIEX derivatives trading record

         # Get the sheet you want to read
         sheet_name = 'ForPython' # Replace with the name of the sheet you want to read
         df = excel_file.parse(sheet_name)
```

```
In [3]:  # Output data information
         print(df.head())

              Date    PnL Index      TAIEX     VIX    Returns  Unnamed: 5  Unnamed: 6  \
         0 2022-07-01  100.000000  14343.08  27.01   0.000000         NaN         NaN
         1 2022-07-04   95.577858  14217.06  27.56  -0.044221         NaN         NaN
         2 2022-07-05   93.953178  14349.20  27.18  -0.016998         NaN         NaN
         3 2022-07-06   92.057052  13985.51  29.40  -0.020182         NaN         NaN
         4 2022-07-07   92.698962  14335.27  28.26   0.006973         NaN         NaN

              Base
         0  100.0
         1    NaN
         2    NaN
         3    NaN
         4    NaN
```

```
In [4]:  #******Plotting setup*****#
         # Generate some data
         Date = df["Date"]
         Date
         y1 =df["PnL Index"]
         y1
         y2 = df["TAIEX"]
         y2
```

```
Out[4]:  0        14343.08
         1        14217.06
         2        14349.20
         3        13985.51
         4        14335.27
                   ...
         283      16634.51
         284      16644.94
         285      16789.69
         286      16791.61
         287      16738.16
         Name: TAIEX, Length: 288, dtype: float64
```

```
In [5]:  # Get the maximum PnL value
         max_pnl = df['PnL Index'].max()
         max_pnl_date = df.loc[df['PnL Index']==max_pnl, 'Date'].values[0]
```

```
In [6]:  # Create the plot and set the first y-axis (left)
         fig, ax1 = plt.subplots()
         plt.xticks(rotation=90)
         ax1.plot(Date, y1, 'b-')
         ax1.scatter(max_pnl_date, max_pnl, color='red', marker='*')
         ax1.set_xlabel('Date')
         ax1.set_ylabel('PnL Index (Base = 100)', color='b')
         ax1.tick_params('y', colors='b')
```

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

# Add message box
msg = "Red * : Highest PnL"
props = dict(boxstyle='round', facecolor='white', alpha=0.5)
ax1.text(0.05, 0.95, msg, transform=ax1.transAxes, fontsize=8,
         verticalalignment='top', bbox=props)

# Show the plot
plt.title('PnL vs TAIEX')
plt.show()
```
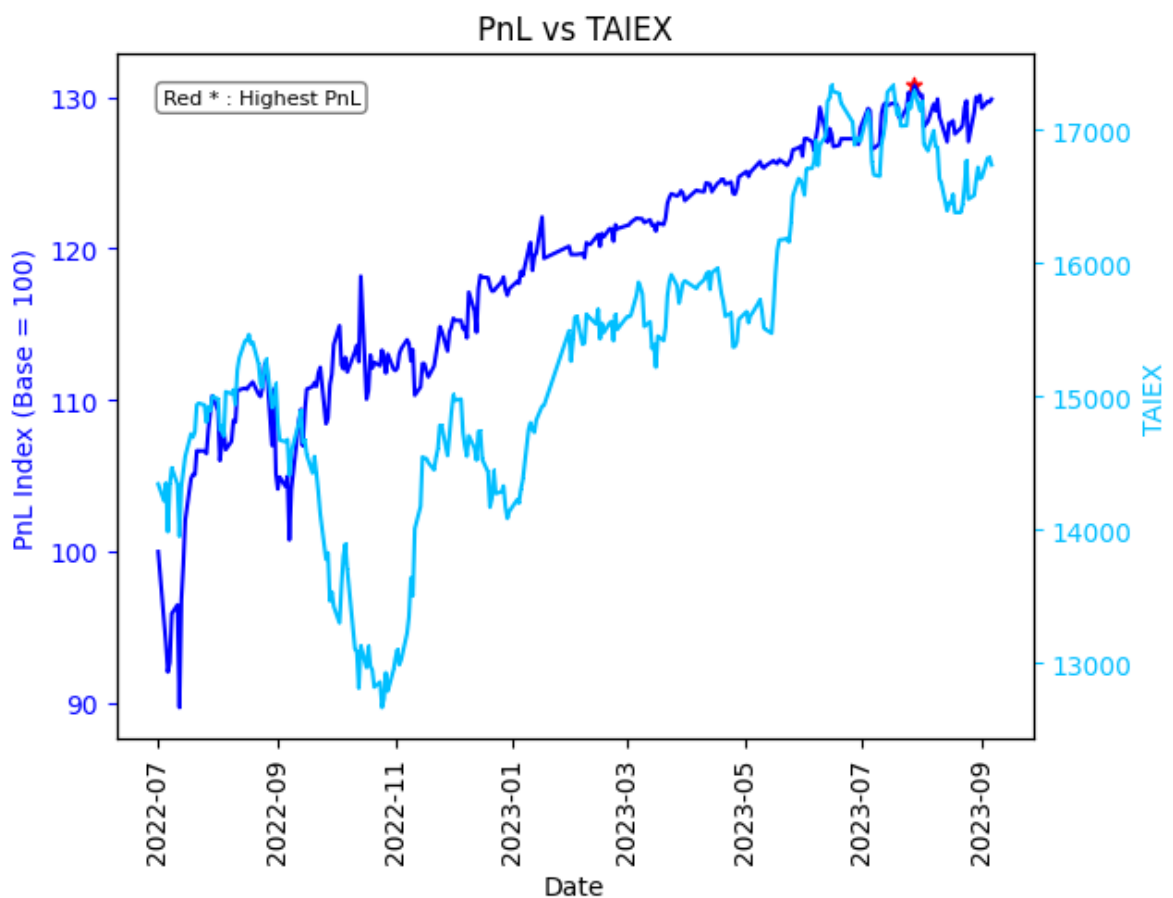
```
#Pnl vs VIX
y3 = df["VIX"]
y3

# Create the plot and set the first y-axis (left)
fig, ax1 = plt.subplots()
plt.xticks(rotation=90)
ax1.plot(Date, y1, 'b-')
ax1.scatter(max_pnl_date, max_pnl, color='red', marker='*')
ax1.set_xlabel('Date')
ax1.set_ylabel('PnL Index (Base = 100)', color='b')
ax1.tick_params('y', colors='b')


# Set the second y-axis (right)
ax3 = ax1.twinx()
ax3.plot(Date, y3, 'fuchsia', marker='1', linestyle='-.')
ax3.set_ylabel('VIX', color='fuchsia')
```
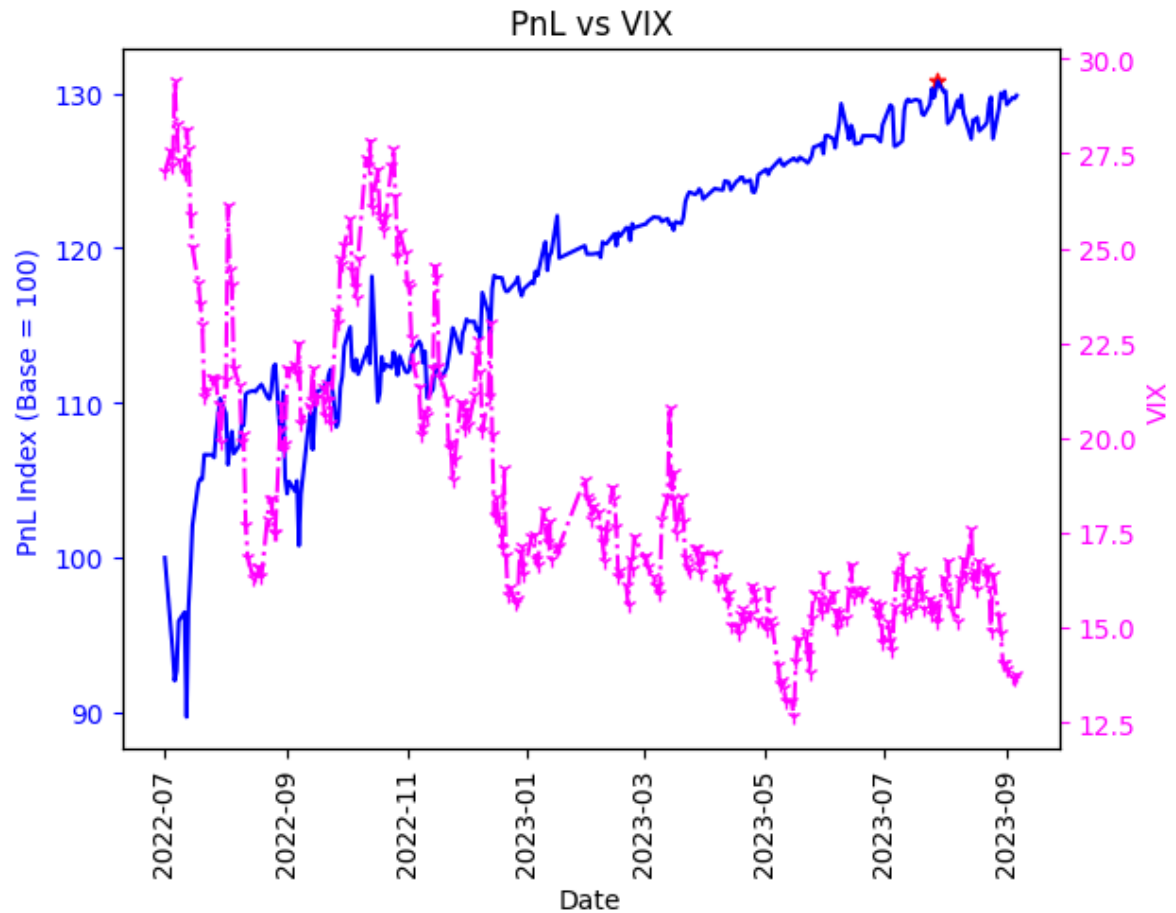
```
ax3.tick_params('y', colors='fuchsia')

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


PnL vs VIX

In [8]:
```
#Tracking error between PnL and TAIEX
PNL_returns = df['PnL Index'].pct_change()
TAIEX_returns = df['TAIEX'].pct_change()
diff_returns = PNL_returns  - TAIEX_returns
tracking_error = diff_returns.std()


roll_te = diff_returns.rolling(20).std()

plt.figure(figsize=(10, 6))
plt.title('Rolling 20-Day Tracking Error btw PnL and TAIEX')
plt.plot(df['Date'], roll_te, color='mediumslateblue')
plt.xlabel('Date')
plt.ylabel('Tracking Error')
plt.show()

#Comment
#Apparently, when market is in turmoil, tracking error  will be widen, and vice ver
#Due to the fact that my derivatives position is well hedged against the market sho
```

Rolling 20-Day Tracking Error btw PnL and TAIEX
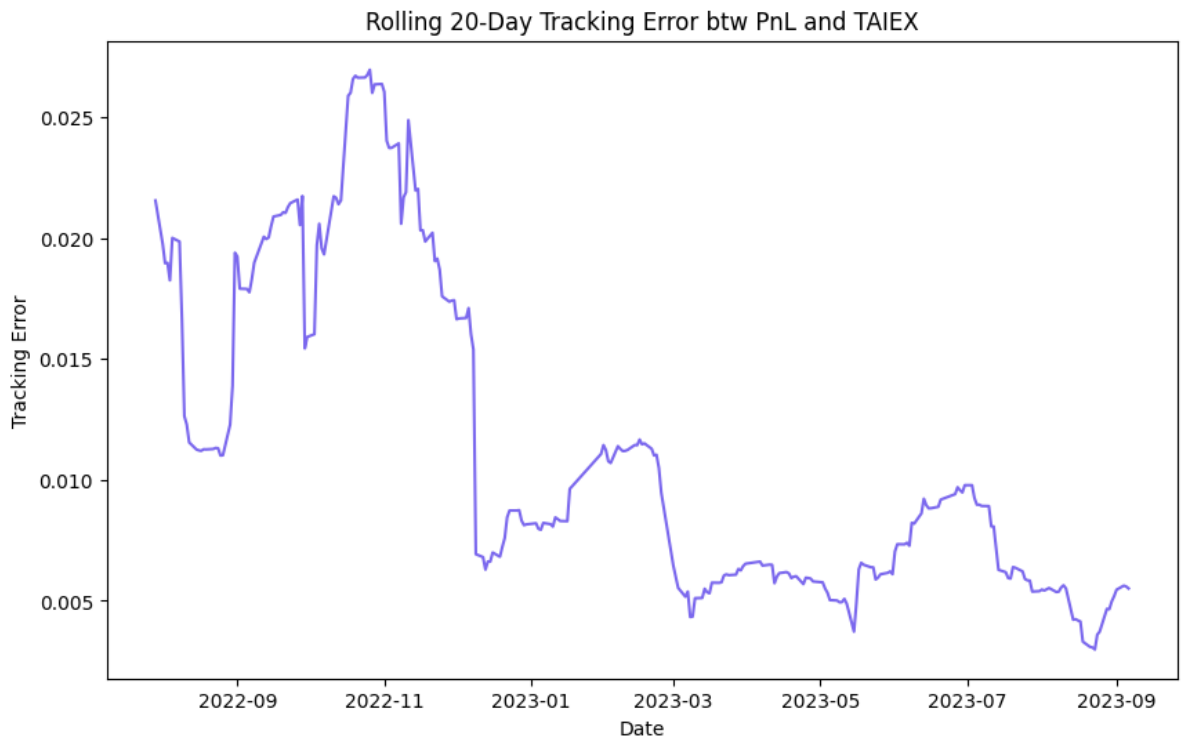
In [9]:
```python
#Historical volatility
#GARCH model volatility

from arch import arch_model
from scipy.stats import mstats

# Calculate log returns
log_returns = np.log(y2/y2.shift(1))

# Remove NaN values
log_returns = log_returns.dropna()

# Winsorize outliers
log_returns = mstats.winsorize(log_returns, limits=0.1)

# Fit GARCH model
garch = arch_model(log_returns, p=1, q=1, dist='StudentsT')
garch_fit = garch.fit(update_freq=10)

# Extract volatility
sigma = garch_fit.conditional_volatility
annual_vol = sigma.mean()*np.sqrt(250)*100

print(annual_vol)
```

```
Optimization terminated successfully    (Exit mode 0)
            Current function value: 24390.139592189757
            Iterations: 13
            Function evaluations: 122
            Gradient evaluations: 9
13631.31554973801
```

In [10]:
```python
#######################################Scientific experiment###############
#Least Squares algo
from scipy.optimize import least_squares

# Set lower and upper bounds
bounds =(10, 45)


# Objective function
def f(vix, PNL_returns , TAIEX_returns):
    diff =  (TAIEX_returns* annual_vol.std() )-(PNL_returns*vix.std())
    return diff.std()

# Set initial guess within bounds
x0 = [15.0]

# By using Trust Region Reflective (bounded)
result1 = least_squares(f, x0,  bounds=bounds, method='trf', args=(TAIEX_returns, 
optimal_vix = result1.x[0]

print("Optimal VIX:", optimal_vix)
print("Minimum Tracking Error:", f(optimal_vix, TAIEX_returns, PNL_returns))

# By using Levenberg-Marquardt algo (unbounded)
result2 = least_squares(f, x0, method='lm', args=(TAIEX_returns, PNL_returns))
optimal_vix = result2.x[0]

print("Optimal VIX:", optimal_vix)
print("Minimum Tracking Error:", f(optimal_vix, TAIEX_returns, PNL_returns))


#Source: https://github.com/scipy/scipy/blob/v1.9.1/scipy/optimize/_lsq/least_squa

#* 'lm' : Levenberg-Marquardt algorithm as implemented in MINPACK.
           #  Doesn't handle bounds and sparse Jacobians. Usually the most
         #   efficient method for small unconstrained problems.
#* 'trf' : Trust Region Reflective algorithm, particularly suitable
             #  for large sparse problems with bounds. Generally robust method.
```

```
Optimal VIX: 15.0
Minimum Tracking Error: 0.0
Optimal VIX: 15.0
Minimum Tracking Error: 0.0
```

In [11]:
```python
###################################### #Performance###########################
#Sharpe ratio
# Read in the portfolio returns data from a CSV file
R_first=df["PnL Index"].iloc[0,]
R_first
R_last = df["PnL Index"].iloc[-1]   #Always excel's actual row-2
R_last
```

```
Out[11]:   129.86285640179412

In [12]:   portfolio_returns=(R_last-R_first)/R_first
           portfolio_returns

Out[12]:   0.2986285640179412

In [13]:   daily_returns=df["Returns"]
           daily_returns

Out[13]:   0       0.000000
           1      -0.044221
           2      -0.016998
           3      -0.020182
           4       0.006973
                     ...
           283     0.002878
           284    -0.006605
           285     0.003504
           286    -0.000442
           287     0.001518
           Name: Returns, Length: 288, dtype: float64

In [14]:   # Max Drawdown Calculation for PnL Index
           cumulative_returns = (1 + df["Returns"]).cumprod()
           cumulative_max = cumulative_returns.cummax()
           drawdown = (cumulative_returns / cumulative_max) - 1
           max_drawdown = drawdown.min()

           print("Max Drawdown:", max_drawdown)

           Max Drawdown: -0.10420949154156467

In [15]:   # Calculate the Profit Factor
           positive_returns = daily_returns[daily_returns > 0].sum()
           negative_returns = daily_returns[daily_returns < 0].sum()

           # Avoid division by zero
           if negative_returns != 0:
               profit_factor = abs(positive_returns / negative_returns)
           else:
               profit_factor = float('inf')

           print("Profit Factor:", profit_factor)

           Profit Factor: 1.301418144921691

In [16]:   # Calculate the excess returns and standard deviation
           risk_free_rate = 0.0145  # Taiwan savings rate
           excess_returns = portfolio_returns - risk_free_rate
           std_dev = np.std(daily_returns)
           print("Standard Deviation of Daily Return:", std_dev)

           Standard Deviation of Daily Return: 0.013465024442633424

In [17]:   # Calculate the Sharpe ratio
           Sharpe_Ratio = excess_returns / std_dev
           print("Sharpe Ratio:", Sharpe_Ratio)

           Sharpe Ratio: 21.10122898242379

In [18]:   #Annualized Sharpe ratio
           risk_free_rate_daily = (1 + risk_free_rate) ** (1/250) - 1
           risk_free_rate_daily
```

```
average_daily_returns = daily_returns.sum()/250
average_daily_returns
excess_daily_return=average_daily_returns-risk_free_rate_daily
excess_daily_return

Annualized_Sharpe_Ratio=excess_daily_return/std_dev*np.sqrt(250)
print("Annualized Sharpe Ratio:", Annualized_Sharpe_Ratio)
```

Annualized Sharpe Ratio: 1.2835519162145208

In [19]:
```
#Portfolio ALpha
# Compute the mean returns
mean_PNL = PNL_returns.mean()
mean_TAIEX = TAIEX_returns.mean()

# Compute beta
covariance = PNL_returns.cov(TAIEX_returns)
variance = TAIEX_returns.var()
beta = covariance / variance
beta

# Compute alpha (assuming risk-free rate is 0)
alpha = (mean_PNL - (risk_free_rate_daily  +beta * mean_TAIEX))*np.sqrt(250)

# Print alpha
print("Alpha: ", alpha)
```

Alpha: 0.010299724190450651

In [ ]:

In [ ]: