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
         ...
344    17370.56
345    17433.85
346    17438.35
347    17421.48
348    17328.01
Name: TAIEX, Length: 349, dtype: float64
```

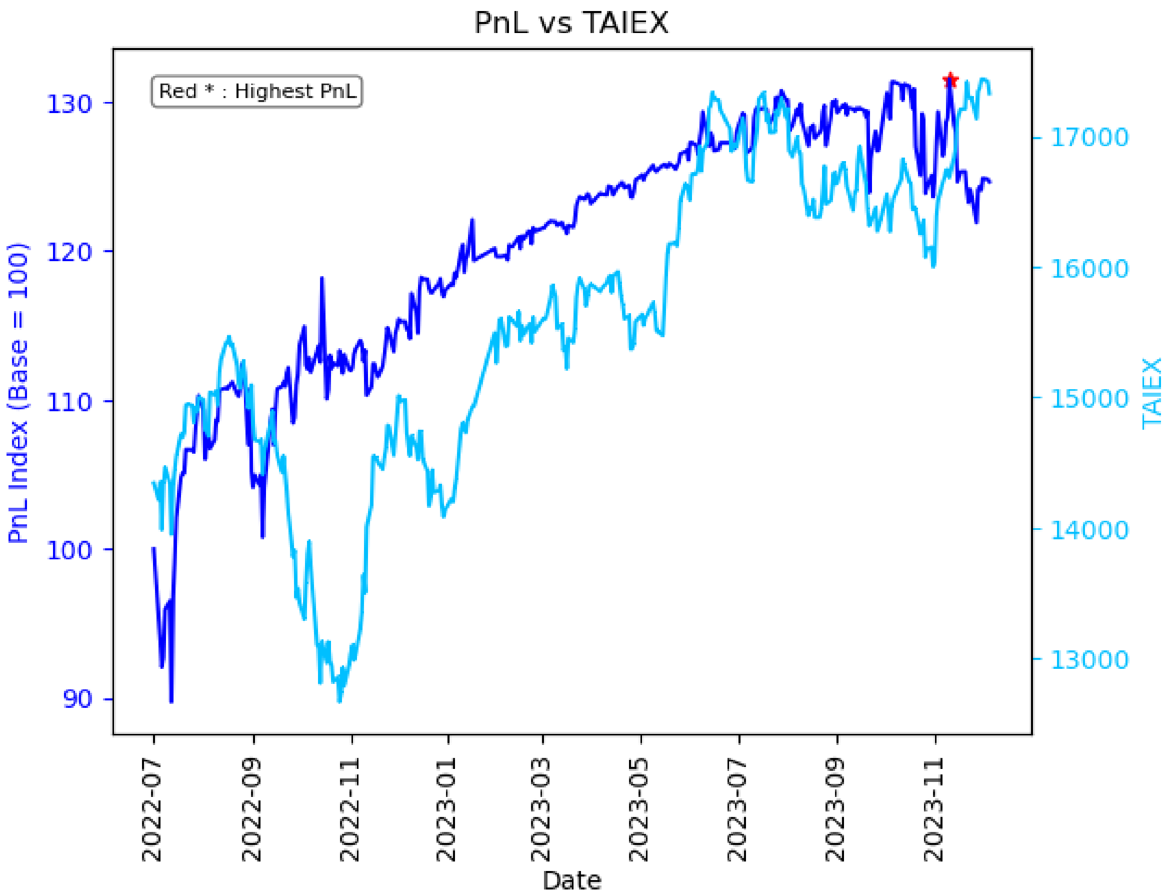
In [5]:
```
# Get the maximum PnL value
max_pnl = df['PnL Index'].max()
max_pnl_date = df.loc[df['PnL Index']==max_pnl, 'Date'].values[0]
```

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()
```



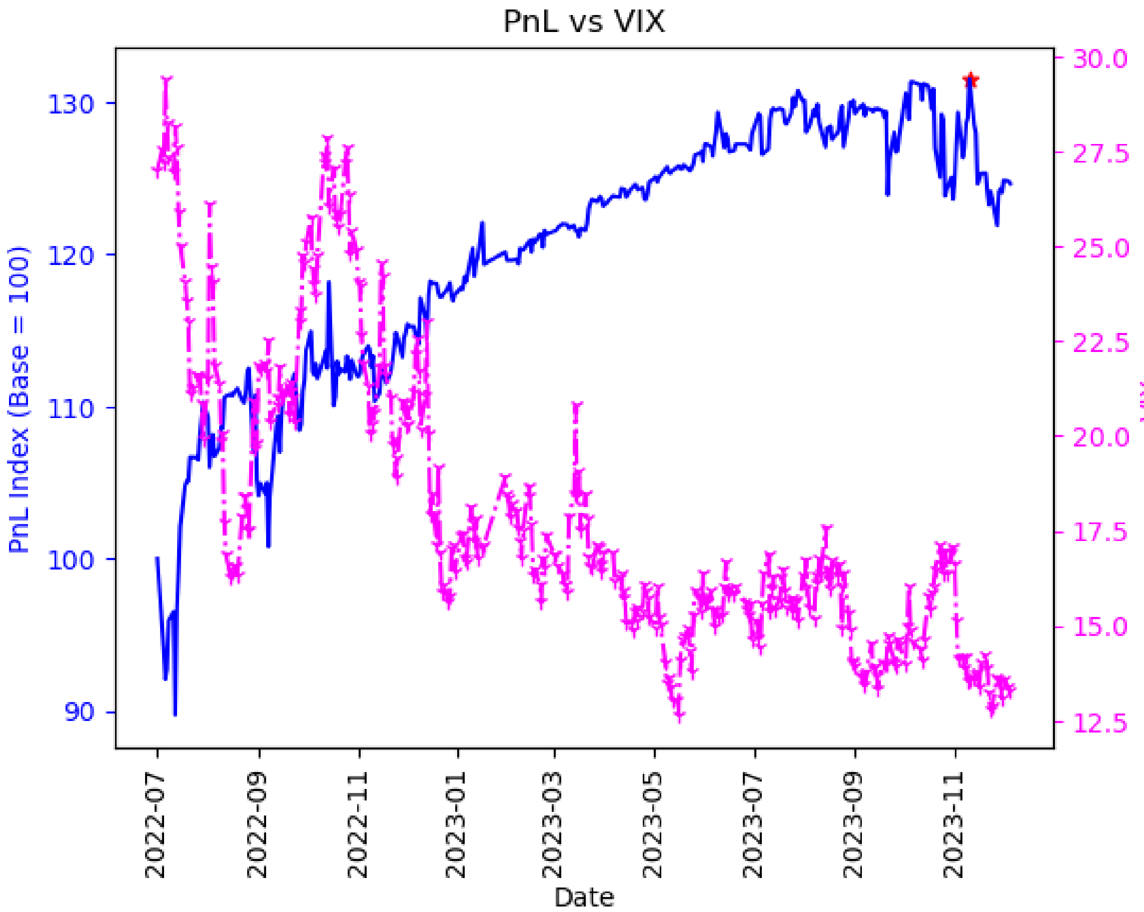
In [7]:
```
#Pnl vs VIX
y3 = df["VIX"]
y3

# 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')
```

```python
ax3.tick_params('y', colors='fuchsia')

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

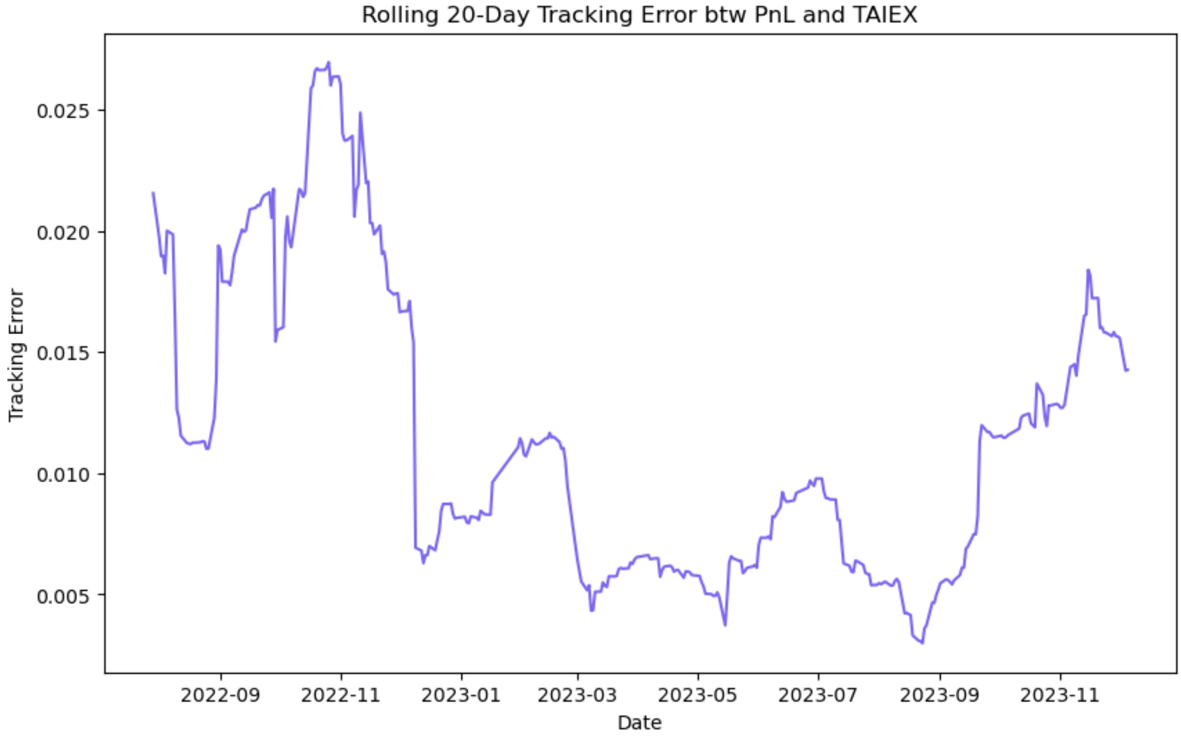


In [8]:
```python
#Tracking error between PnL and TAIEX
PNL_returns = df['PnL Index'].pct_change()
TAIEX_returns = df['TAIEX'].pct_change()
diff_returns = PNL_returns  - TAIEX_returns
tracking_error = diff_returns.std()


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
```



In [9]:

```
#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()
log_returns = mstats.winsorize(log_returns, limits=0.1)

#Volatility estimation by GARCH (p, q)
from arch import arch_model
import warnings
warnings.filterwarnings("ignore")

# Define the maximum p and q
max_p = 5
max_q = 5

# Initialize variables to store best values of p and q
best_p = 0
best_q = 0
best_bic = np.inf

# Optimal leg selections for p and q of GARCH
for p in range(max_p + 1):
    for q in range(max_q + 1):
        try:
            # Define the GARCH model
            model = arch_model(log_returns, vol="Garch", p=p, q=q)
            # Fit the GARCH model
            model_fit = model.fit(disp='off')
            # If the current model's BIC is lower than our best_bic, update the bes
            if model_fit.bic < best_bic:
                best_p = p
                best_q = q
                best_bic = model_fit.bic
        except:
```

```
            pass
print(f"The best model is GARCH({best_p}, {best_q}) with BIC of {best_bic}")
# Use the suggested best fitted GARCH model parameter (p=1, q=1) by the Bayesian Ir
# 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)
```

```
The best model is GARCH(1, 1) with BIC of -2392.5001179105566
Iteration:     10,   Func. Count:     94,   Neg. LLF: 3457.672697471252
Iteration:     20,   Func. Count:    164,   Neg. LLF: -123.13276128511656
Iteration:     30,   Func. Count:    261,   Neg. LLF: 1183.2139455092401
Iteration:     40,   Func. Count:    337,   Neg. LLF: 1284.9113181841112
Iteration:     50,   Func. Count:    406,   Neg. LLF: 6729.566847444594
Optimization terminated successfully    (Exit mode 0)
            Current function value: -986.5808051421802
            Iterations: 60
            Function evaluations: 454
            Gradient evaluations: 56
35.53727504643416
```

In [10]:
```
##########################################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[10]:
```
124.65354281484647
```

In [11]:
```
portfolio_returns=(R_last-R_first)/R_first
portfolio_returns
```

Out[11]:
```
0.24653542814846474
```

In [12]:
```
daily_returns=df["Returns"]
daily_returns
```

Out[12]:
```
0       0.000000
1      -0.044221
2      -0.016998
3      -0.020182
4       0.006973
          ...
344     0.001732
345    -0.001968
346     0.006404
347    -0.000540
348    -0.001172
Name: Returns, Length: 349, dtype: float64
```

In [13]:
```
# 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 [14]:
```
# 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: 1.1993896589972588
```

In [15]:
```
# Calculate the excess returns and standard deviation
risk_free_rate = 0.01148  # TAIBOR rate source: https://www.ba.org.tw/Taibor/Detail
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: 0.013533569416998124
```

In [16]:
```
# Calculate the Sharpe ratio
Sharpe_Ratio = excess_returns / std_dev
print("Sharpe Ratio:", Sharpe_Ratio)
```

```
Sharpe Ratio: 17.36832471212183
```

In [17]:
```
#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: 1.1269567294626222
```

In [18]:
```
#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:  0.005966624105411075
```

In [ ]:
```

```

In [ ]: