In [1]:
```
pip install yfinance
```

```
Collecting yfinance
  Obtaining dependency information for yfinance from https://files.pythonhosted.or
g/packages/09/05/28664524fcc67c078313d482bf25fe403e9399130622cfc89e185ec0abf6/yfin
ance-0.2.54-py2.py3-none-any.whl.metadata
  Downloading yfinance-0.2.54-py2.py3-none-any.whl.metadata (5.8 kB)
Requirement already satisfied: pandas>=1.3.0 in c:\users\captr\anaconda3\lib\site-
packages (from yfinance) (2.0.3)
Requirement already satisfied: numpy>=1.16.5 in c:\users\captr\anaconda3\lib\site-
packages (from yfinance) (1.24.3)
Requirement already satisfied: requests>=2.31 in c:\users\captr\anaconda3\lib\site
-packages (from yfinance) (2.31.0)
Collecting multitasking>=0.0.7 (from yfinance)
  Obtaining dependency information for multitasking>=0.0.7 from https://files.pyth
onhosted.org/packages/3e/8a/bb3160e76e844db9e69a413f055818969c8acade64e1a9ac5ce9df
dcf6c1/multitasking-0.0.11-py3-none-any.whl.metadata
  Downloading multitasking-0.0.11-py3-none-any.whl.metadata (5.5 kB)
Requirement already satisfied: platformdirs>=2.0.0 in c:\users\captr\anaconda3\lib
\site-packages (from yfinance) (3.10.0)
Requirement already satisfied: pytz>=2022.5 in c:\users\captr\anaconda3\lib\site-p
ackages (from yfinance) (2023.3.post1)
Collecting frozendict>=2.3.4 (from yfinance)
  Obtaining dependency information for frozendict>=2.3.4 from https://files.python
hosted.org/packages/04/13/d9839089b900fa7b479cce495d62110cddc4bd5630a04d8469916c0e
79c5/frozendict-2.4.6-py311-none-any.whl.metadata
  Downloading frozendict-2.4.6-py311-none-any.whl.metadata (23 kB)
Collecting peewee>=3.16.2 (from yfinance)
  Downloading peewee-3.17.9.tar.gz (3.0 MB)
     ---------------------------------------- 0.0/3.0 MB ? eta -:--:--
     - -------------------------------------- 0.1/3.0 MB 2.3 MB/s eta 0:00:02
     --------- ------------------------------ 0.7/3.0 MB 8.9 MB/s eta 0:00:01
     --------------------- ------------------ 1.7/3.0 MB 12.1 MB/s eta 0:00:01
     ---------------------------------- ----- 2.6/3.0 MB 15.1 MB/s eta 0:00:01
     ---------------------------------------- 3.0/3.0 MB 13.8 MB/s eta 0:00:00
  Installing build dependencies: started
  Installing build dependencies: finished with status 'done'
  Getting requirements to build wheel: started
  Getting requirements to build wheel: finished with status 'done'
  Preparing metadata (pyproject.toml): started
  Preparing metadata (pyproject.toml): finished with status 'done'
Requirement already satisfied: beautifulsoup4>=4.11.1 in c:\users\captr\anaconda3
\lib\site-packages (from yfinance) (4.12.2)
Requirement already satisfied: soupsieve>1.2 in c:\users\captr\anaconda3\lib\site-
packages (from beautifulsoup4>=4.11.1->yfinance) (2.4)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\captr\anaconda3
\lib\site-packages (from pandas>=1.3.0->yfinance) (2.8.2)
Requirement already satisfied: tzdata>=2022.1 in c:\users\captr\anaconda3\lib\site
-packages (from pandas>=1.3.0->yfinance) (2023.3)
Requirement already satisfied: charset-normalizer<4,>=2 in c:\users\captr\anaconda
3\lib\site-packages (from requests>=2.31->yfinance) (2.0.4)
Requirement already satisfied: idna<4,>=2.5 in c:\users\captr\anaconda3\lib\site-p
ackages (from requests>=2.31->yfinance) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in c:\users\captr\anaconda3\lib
\site-packages (from requests>=2.31->yfinance) (1.26.16)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\captr\anaconda3\lib
\site-packages (from requests>=2.31->yfinance) (2023.7.22)
Requirement already satisfied: six>=1.5 in c:\users\captr\anaconda3\lib\site-packa
ges (from python-dateutil>=2.8.2->pandas>=1.3.0->yfinance) (1.16.0)
Downloading yfinance-0.2.54-py2.py3-none-any.whl (108 kB)
   ---------------------------------------- 0.0/108.7 kB ? eta -:--:--
   ---------------------------------------- 108.7/108.7 kB 6.6 MB/s eta 0:00:00
Downloading frozendict-2.4.6-py311-none-any.whl (16 kB)
Downloading multitasking-0.0.11-py3-none-any.whl (8.5 kB)
Building wheels for collected packages: peewee
  Building wheel for peewee (pyproject.toml): started
```

```
  Building wheel for peewee (pyproject.toml): finished with status 'done'
  Created wheel for peewee: filename=peewee-3.17.9-py3-none-any.whl size=139096 sh
a256=ac1fe5aeda96dd317273137908309b412daf66eab89a715ba1ab9933c7495af2
  Stored in directory: c:\users\captr\appdata\local\pip\cache\wheels\f4\14\e4\50c8
8c865833085aeb91e2bd40e3a683ff434806386b8ee7bc
Successfully built peewee
Installing collected packages: peewee, multitasking, frozendict, yfinance
Successfully installed frozendict-2.4.6 multitasking-0.0.11 peewee-3.17.9 yfinance
-0.2.54
Note: you may need to restart the kernel to use updated packages.
```

In [2]:

```
#importing libraries for data manipulation and visualization
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import yfinance as yf
import datetime
import warnings
warnings.filterwarnings('ignore')
```

In [5]:

```
#downloading data from yfinance api
end = '2024-12-31'
start = '2019-1-1'
#downloading icicibank and axisbank last 7 years close prices from yfinance
x = yf.download('KOTAKBANK.NS',start,end)['Close']
y = yf.download('HDFCBANK.NS',start,end)['Close']
```

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

In [6]:

```
df = pd.concat([x,y],axis=1)
df.columns = ['KOTAKBANK','HDFCBANK']
# df.index = pd.to_datetime(df.index)
df
```

Out[6]:

	KOTAKBANK	HDFCBANK
Date		
2019-01-01	1246.011719	1022.420776
2019-01-02	1236.196533	1013.091736
2019-01-03	1230.865723	1005.143188
2019-01-04	1243.520508	1007.832336
2019-01-07	1242.175293	1009.355408
...	...	...
2024-12-23	1745.349976	1801.000000
2024-12-24	1749.050049	1798.099976
2024-12-26	1752.800049	1790.750000
2024-12-27	1759.900024	1798.250000
2024-12-30	1740.699951	1777.900024

1480 rows × 2 columns

In [7]:

```
df.plot(figsize=(10,7))
plt.ylabel("Price")
```

```
plt.show()
```

Upon initial observation of the charts, it appears that there is a correlation in the prices of both stocks. However, before proceeding, it is imperative to conduct further investigation to determine whether these stocks are indeed suitable candidates for the pairs trading strategy.

We're employing linear regression to determine the relationship between the prices of HDFC Bank ('HDFCBANK') and Kotak Mahindra Bank ('KOTAKBANK'). This analysis yields the hedge ratio, indicating the degree of co-movement between the two stocks. A higher hedge ratio implies a stronger correlation, which is essential for constructing a viable pairs trading strategy.

In [8]:

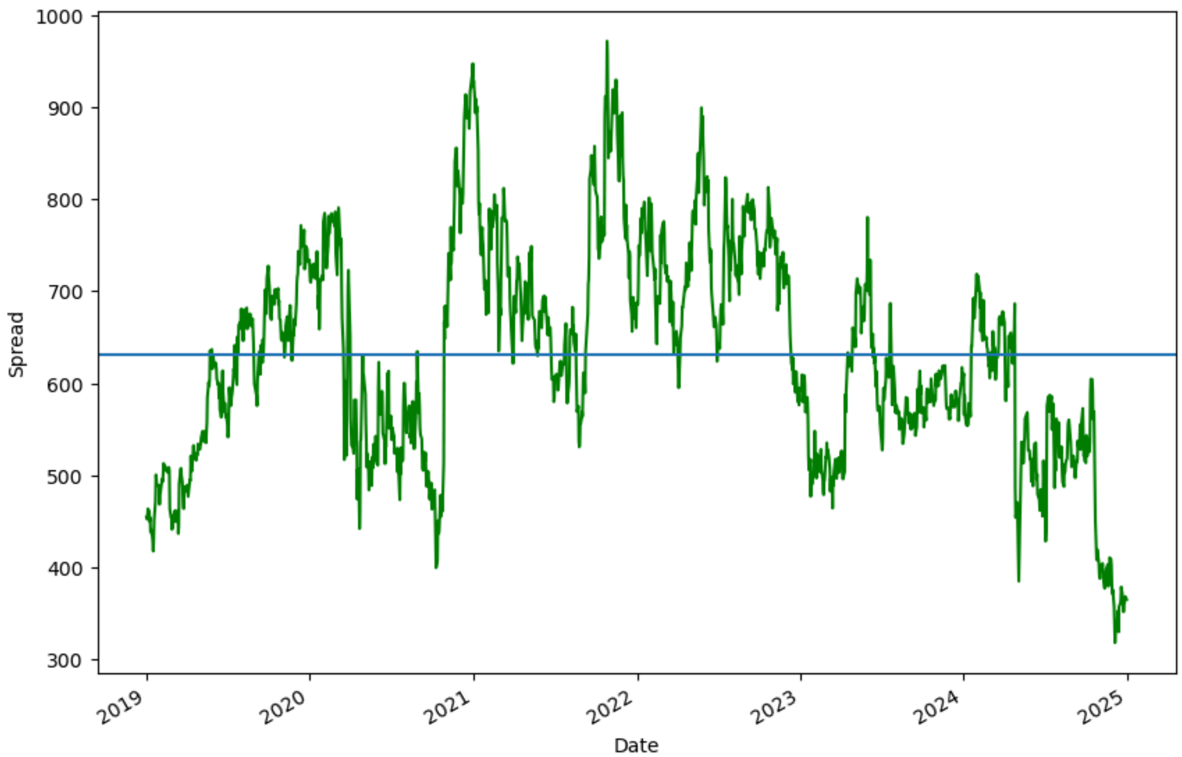
```
'''
Performing linear regression between HDFC Bank ('HDFCBANK') and Kotak Mahindra Bank
prices using np.polyfit(). The resulting slope represents the hedge ratio,
indicating how much of Kotak Mahindra Bank we need to long/short against one unit c

'''


X = df['HDFCBANK']
y = df['KOTAKBANK']

model = np.polyfit(X,y,deg=1)
hedge_ratio = model[0]
hr = round(hedge_ratio,3)
print(f'The hedge ratio is {hr}')
```

```
The hedge ratio is 0.774
```

Hedge ratio=0.77: This represents the ratio between the two assets in the pair trading strategy. It shows that for each unit of HDFCBANK, you should take a position of 0.774 units of KOTAKBANK.NS. This seems like a reasonable hedge ratio,

which suggests that the two stocks are correlated in a way that would allow a market-neutral position.

In [9]:
```
'''
Calculating and plotting the spread between 'KOTAKBANK' and 'HDFCBANK'
prices, adjusted by the hedge ratio. This spread visualization aids in
identifying trading opportunities based on price differentials.

'''

df['spread'] = df.KOTAKBANK - hr * df.HDFCBANK

# Plot the spread
df.spread.plot(figsize=(10,7), color='g')
plt.axhline(df.spread.mean())
plt.ylabel("Spread")

plt.show()
```



Cointegration

Cointegration is a unique form of correlation between two time series, where their ratio fluctuates around a mean value. In pairs trading, cointegration is essential as it ensures the ratio between assets converges to a stable mean over time, validating the strategy's effectiveness.

Testing for Cointegration

In the statsmodels.tsa.stattools library, there's a convenient test for cointegration. Given that we've artificially constructed two highly cointegrated series, we expect to observe an extremely low p-value from this test.

In [10]:
```python
"""
Using the Augmented Dickey-Fuller (ADF) test from statsmodels.tsa.stattools
to assess the stationarity of the 'spread' series. A low p-value indicates strong e
against non-stationarity, validating its suitability for pairs trading strategies.

"""
# Importing the ADF test function from statsmodels.tsa.stattools
from statsmodels.tsa.stattools import adfuller

# Applying the ADF test to the 'spread' series with a maximum lag of 1
adf_result = adfuller(df.spread, maxlag=1)

# Printing the ADF test results
print("ADF Statistic:", adf_result[0])
print("p-value:", adf_result[1])
print("Critical Values:", adf_result[4])

# Check if the spread series is stationary based on the p-value
if adf_result[1] < 0.05:
    print("The spread series is likely stationary.")
else:
    print("The spread series is likely not stationary.")
```

```
ADF Statistic: -3.649822523072047
p-value: 0.0048769337260563565
Critical Values: {'1%': -3.434779131760461, '5%': -2.863496173799589, '10%': -2.56
78114464207265}
The spread series is likely stationary.
```

The ADF test (Augmented Dickey-Fuller test) is used to check if the spread between the two stocks is stationary.

p-value < 0.05 indicates statistical significance, meaning we can reject the null hypothesis that the spread is non-stationary. The ADF statistic is also more negative than the critical values at all significance levels (1%, 5%, and 10%), further supporting that the spread is stationary.

Conclusion: The spread between the two stocks is likely stationary, which is a good sign for a pair trading strategy, as the assumption is that the spread will mean revert over time.

Spread-based Mean Reversion Strategy Function

This function calculates trading positions based on the spread, incorporating parameters such as the lookback period and standard deviation. It enables the implementation of a mean reversion trading strategy.

In [11]:
```python
def mean_reversion_strategy(df, period, std_dev):
    """
    Computes trading positions based on the spread, using a mean reversion strategy

    Parameters:
```

```
    - df: DataFrame containing spread data
    - period: Lookback period for calculating moving average and standard deviatior
    - std_dev: Number of standard deviations to use for defining upper and lower ba

    Returns:
    - DataFrame with additional columns for positions based on the mean reversion s
    """

    # Moving Average
    df['moving_average'] = df.spread.rolling(period).mean()
    # Moving Standard Deviation
    df['moving_std_dev'] = df.spread.rolling(period).std()

    # Upper band and Lower band
    df['upper_band'] = df.moving_average + std_dev * df.moving_std_dev
    df['lower_band'] = df.moving_average - std_dev * df.moving_std_dev

    # Long positions
    df['long_entry'] = df.spread < df.lower_band
    df['long_exit'] = df.spread >= df.moving_average
    df['positions_long'] = np.nan
    df.loc[df.long_entry, 'positions_long'] = 1
    df.loc[df.long_exit, 'positions_long'] = 0
    df.positions_long = df.positions_long.fillna(method='ffill')

    # Short positions
    df['short_entry'] = df.spread > df.upper_band
    df['short_exit'] = df.spread <= df.moving_average
    df['positions_short'] = np.nan
    df.loc[df.short_entry, 'positions_short'] = -1
    df.loc[df.short_exit, 'positions_short'] = 0
    df.positions_short = df.positions_short.fillna(method='ffill')

    # Combined positions
    df['positions'] = df.positions_long + df.positions_short

    return df
```

In [12]:
```
df = mean_reversion_strategy(df,30,1)
df.dropna(inplace=True)
df.tail(3)
```

Out[12]:

Date	KOTAKBANK	HDFCBANK	spread	moving_average	moving_std_dev	upper_band	lowe
2024-12-26	1752.800049	1790.750000	366.759549	367.693598	24.779512	392.473110	342.
2024-12-27	1759.900024	1798.250000	368.054524	366.911640	24.368658	391.280298	342.
2024-12-30	1740.699951	1777.900024	364.605332	366.182315	24.088453	390.270768	342.

Cumulative returns

In [13]:
```
df['percentage_change'] = (df.spread - df.spread.shift(1))/(hr*df.HDFCBANK + df.KOT
df['strategy_returns'] = df.positions.shift(1) * df.percentage_change
```

```
df['cumulative_returns'] = (df.strategy_returns+1).cumprod()
"The total strategy returns are %.2f" % ((df['cumulative_returns'].iloc[-1]-1)*100)
```

Out[13]: 'The total strategy returns are 118.50'

A return of 118.50% over the period of 5 years is excellent, indicating that the strategy has performed well overall.

In [14]:
```
# Calculating the Sharpe ratio of the strategy
sharpe_ratio = (df['strategy_returns'].mean() * 252) / (df['strategy_returns'].std(

# Printing the Sharpe ratio with 3 decimal places
print(f'Sharpe Ratio: {np.round(sharpe_ratio, 3)}')
```

Sharpe Ratio: 1.163

The Sharpe Ratio measures the risk-adjusted return. A Sharpe ratio of 1.163 is reasonable, meaning we're earning over 1% return per unit of risk. Generally, a Sharpe ratio above 1 is considered good, with values above 2 being excellent. The ratio is decent but could be higher to consider this a very robust strategy.

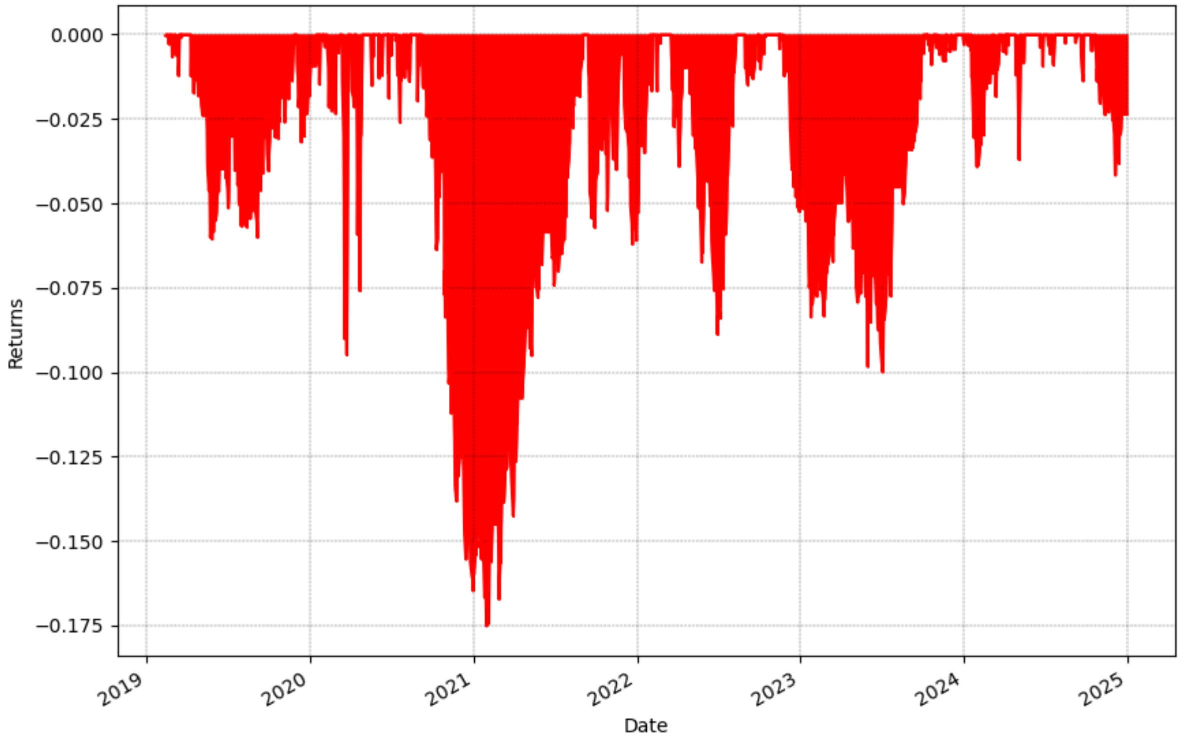
In [15]:
```
#plotting the cumulative returns of the strategy
df.cumulative_returns.plot(label='Returns', figsize=(10,7),color='magenta')
plt.xlabel('Date')
plt.ylabel('Cumulative Returns')
plt.show()
```

DRWADOWN FUNCTION

In [16]:
```
def calc_drawdown(cum_rets):
    # Calculate the running maximum
```

```
    running_max = np.maximum.accumulate(cum_rets.dropna())
    # Ensure the value never drops below 1
    running_max[running_max < 1] = 1
    # Calculate the percentage drawdown
    drawdown = (cum_rets)/running_max - 1
    return drawdown

def plot_drawdown(drawdown):
    fig = plt.figure(figsize=(10, 7))
    # Plot
    drawdown.plot(color='r')
    plt.ylabel('Returns')
    plt.fill_between(drawdown.index, drawdown, color='red')
    plt.grid(which="major", color='k', linestyle='-.', linewidth=0.2)
    plt.show()

drawdown_strategy = calc_drawdown(df.cumulative_returns)
print("The maximum drawdown is %.2f" % (drawdown_strategy.min()*100))
plot_drawdown(drawdown_strategy)
```

```
The maximum drawdown is -17.52
```



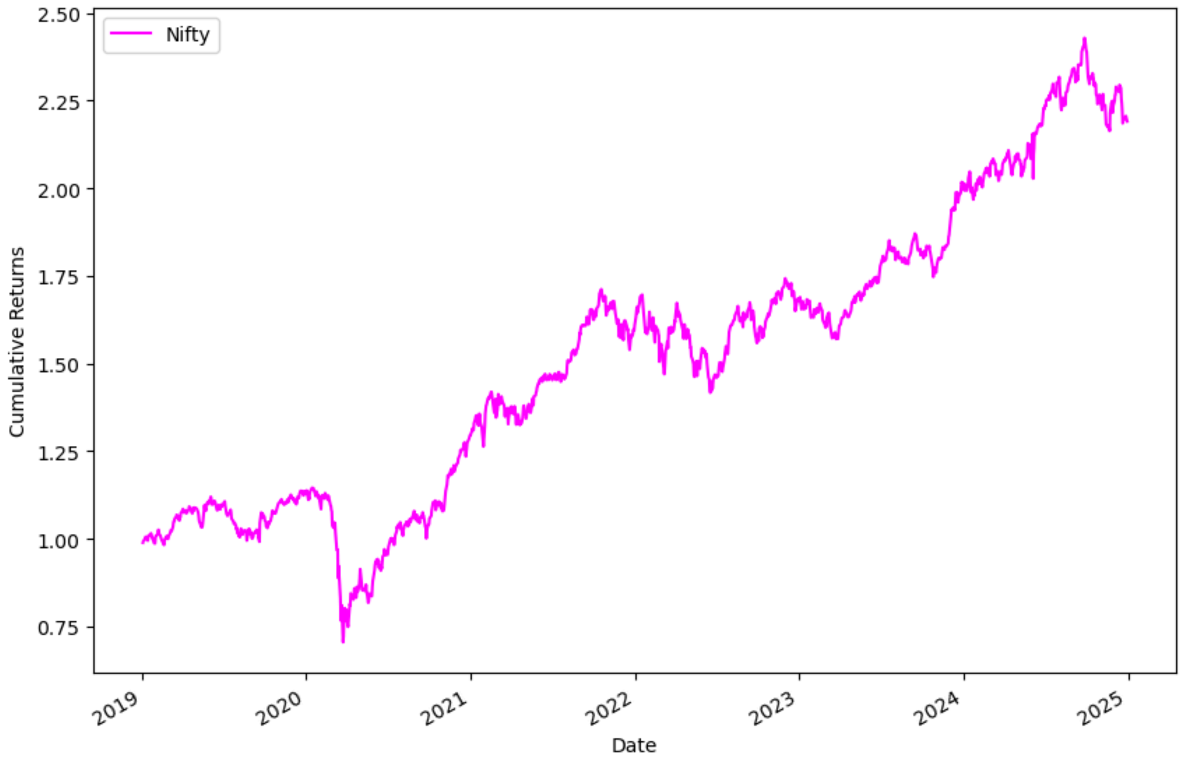
Maximum Drawdown (MDD) refers to the largest peak-to-trough loss in the portfolio. A drawdown of -17.52% is not negligible, but it's also not excessive for a strategy that seeks to capture mean reversion in pair trading. Typically, We want to keep MDD under 20% to minimize the risk of large losses during unfavorable market conditions.

In [18]:

```
#to compare how our strategy performed , we will create an instance and compare the
start_date = '2019-01-01'
end_date = '2024-12-31'

nifty = yf.download('^NSEI', start=start_date, end=end_date)['Close']
nifty.columns = ['Nifty']
nifty_cum_rets = (nifty.pct_change().dropna()+1).cumprod()
nifty_cum_rets.plot(label='Nifty', figsize=(10,7),color='magenta')
plt.xlabel('Date')
```

```
plt.ylabel('Cumulative Returns')
plt.show()
nifty
```

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



Out[18]:

Date	Nifty
2019-01-02	10792.500000
2019-01-03	10672.250000
2019-01-04	10727.349609
2019-01-07	10771.799805
2019-01-08	10802.150391
...	...
2024-12-23	23753.449219
2024-12-24	23727.650391
2024-12-26	23750.199219
2024-12-27	23813.400391
2024-12-30	23644.900391

1477 rows × 1 columns

In [22]:

```
#drawdown_nifty
drawdown_nifty = calc_drawdown(nifty_cum_rets)
print("The maximum drawdown is %.2f" % (drawdown_nifty.min()*100))
```

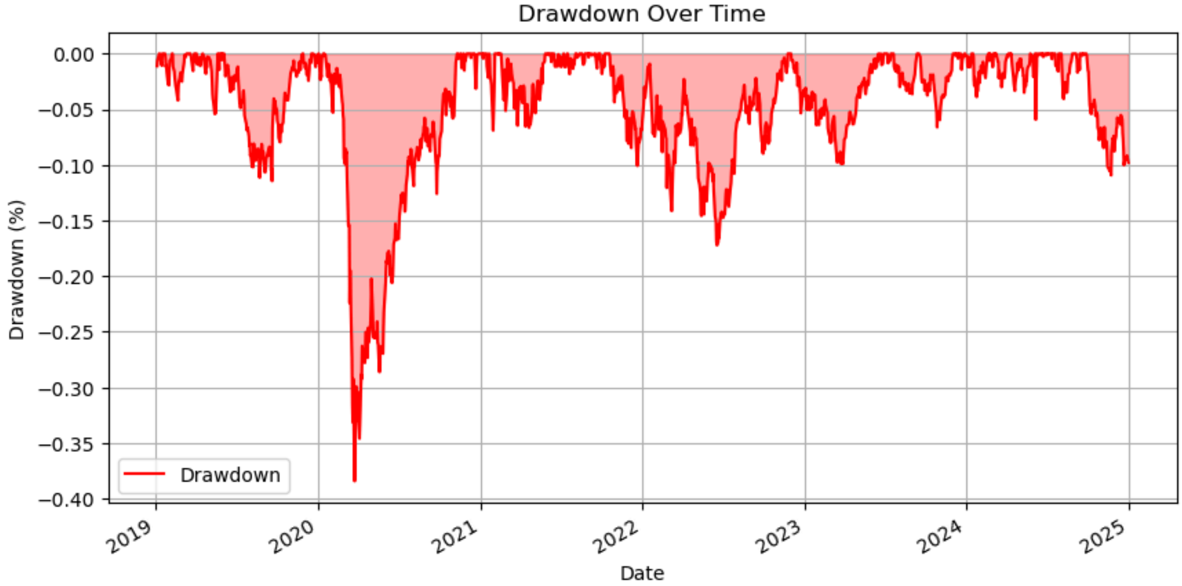
```
The maximum drawdown is -38.44
```

The Nifty's drawdown of -38.44% is more significant, which suggests that while our strategy does experience some

drawdowns, it is performing better than the market overall during periods of downturns. This is a positive sign as the strategy is less volatile compared to the broader market.

In [23]:
```
import matplotlib.pyplot as plt

def plot_drawdown(drawdown):
    plt.figure(figsize=(10, 5))

    # Ensure drawdown is a Pandas Series
    if isinstance(drawdown, pd.DataFrame):
        drawdown = drawdown.squeeze()  # Convert to Series

    if drawdown.ndim != 1:
        raise ValueError("Drawdown data must be 1-dimensional")

    drawdown.plot(color='r', label="Drawdown")
    plt.fill_between(drawdown.index, drawdown, color='red', alpha=0.3)
    plt.ylabel('Drawdown (%)')
    plt.xlabel('Date')
    plt.title('Drawdown Over Time')
    plt.grid(True)
    plt.legend()
    plt.show()
```

In [24]:
```
drawdown_nifty = calc_drawdown(nifty_cum_rets)

# Ensure it's a Series
if isinstance(drawdown_nifty, pd.DataFrame):
    drawdown_nifty = drawdown_nifty.squeeze()

plot_drawdown(drawdown_nifty)
```



In [ ]: