In [3]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import yfinance as yf
```

In [4]:
```
start_date = '2010-01-01'
end_date = '2023-05-01'

nasdaq_data= yf.download('^IXIC',start=start_date, end=end_date)['Adj Close']
nse_data= yf.download('^NSEI',start=start_date, end=end_date)['Adj Close']
```

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

In [5]:
```
nasdaq_data
```

Out[5]:
```
Date
2010-01-04     2308.419922
2010-01-05     2308.709961
2010-01-06     2301.090088
2010-01-07     2300.050049
2010-01-08     2317.169922
                  ...
2023-04-24    12037.200195
2023-04-25    11799.160156
2023-04-26    11854.349609
2023-04-27    12142.240234
2023-04-28    12226.580078
Name: Adj Close, Length: 3353, dtype: float64
```

In [6]:
```
nse_data
```

Out[6]:
```
Date
2010-01-04     5232.200195
2010-01-05     5277.899902
2010-01-06     5281.799805
2010-01-07     5263.100098
2010-01-08     5244.750000
                  ...
2023-04-24    17743.400391
2023-04-25    17769.250000
2023-04-26    17813.599609
2023-04-27    17915.050781
2023-04-28    18065.000000
Name: Adj Close, Length: 3268, dtype: float64
```

In [ ]:

In [7]:
```
corr_coeficient=nasdaq_data.corr(nse_data)
print(corr_coeficient)
```

```
0.9513138758576782
```

In [ ]:

In [8]:
```
df=pd.DataFrame({'NSE': nse_data,'NASDAQ': nasdaq_data})
df.head()
```

Out[8]:

Date	NSE	NASDAQ
2010-01-04	5232.200195	2308.419922
2010-01-05	5277.899902	2308.709961
2010-01-06	5281.799805	2301.090088
2010-01-07	5263.100098	2300.050049
2010-01-08	5244.750000	2317.169922

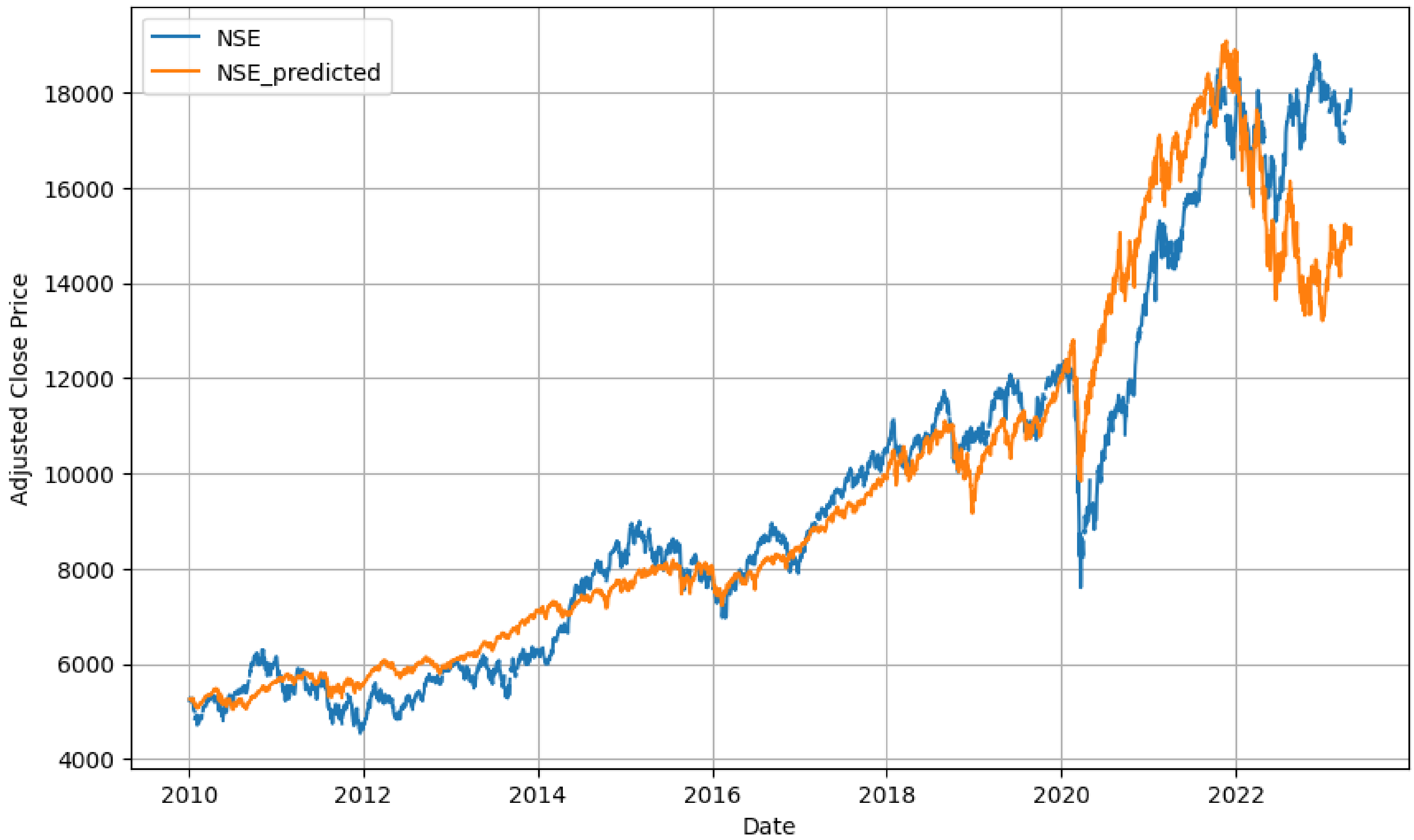
In [9]:
```
df['NASDAQ_Lag']=df['NASDAQ'].shift(1)
df.head()
```

Out[9]:

Date	NSE	NASDAQ	NASDAQ_Lag
2010-01-04	5232.200195	2308.419922	NaN
2010-01-05	5277.899902	2308.709961	2308.419922
2010-01-06	5281.799805	2301.090088	2308.709961
2010-01-07	5263.100098	2300.050049	2301.090088
2010-01-08	5244.750000	2317.169922	2300.050049

In [10]:
```
from sklearn.linear_model import LinearRegression
X=df.loc[:,['NASDAQ_Lag']]
X.dropna(inplace=True)
y=df.loc[:,'NSE']
y.dropna(inplace=True)
y,X=y.align(X, join='inner')
model = LinearRegression()
model.fit(X, y)
y_pred = pd.Series(model.predict(X), index=X.index)
```

In [11]:
```
plt.figure(figsize=(10, 6))
plt.plot(df['NSE'], label='NSE')
plt.plot(y_pred, label='NSE_predicted')
plt.xlabel('Date')
plt.ylabel('Adjusted Close Price')
#plt.title('NSE vs NASDAQ')
plt.legend()
plt.grid(True)
plt.show()
```



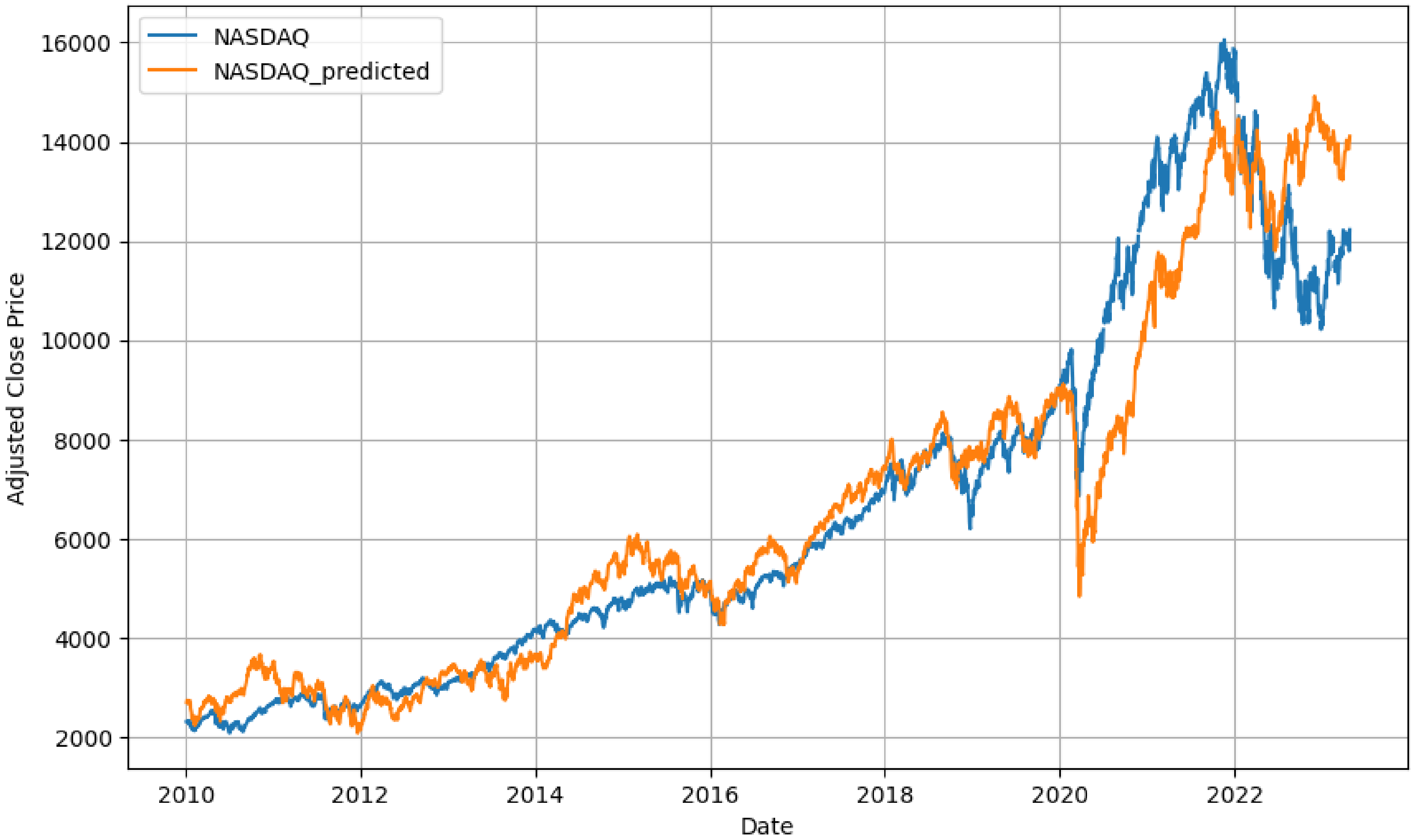
In [12]:
```
df['NSE_lag']=df['NSE'].shift(1)
df.head()
```

Out[12]:

Date	NSE	NASDAQ	NASDAQ_Lag	NSE_lag
2010-01-04	5232.200195	2308.419922	NaN	NaN
2010-01-05	5277.899902	2308.709961	2308.419922	5232.200195
2010-01-06	5281.799805	2301.090088	2308.709961	5277.899902
2010-01-07	5263.100098	2300.050049	2301.090088	5281.799805
2010-01-08	5244.750000	2317.169922	2300.050049	5263.100098

In [13]:
```
from sklearn.linear_model import LinearRegression
X=df.loc[:,['NSE_lag']]
X.dropna(inplace=True)
y=df.loc[:,'NASDAQ']
y.dropna(inplace=True)
y,X=y.align(X, join='inner')
model = LinearRegression()
model.fit(X, y)
y_pred = pd.Series(model.predict(X), index=X.index)
```

In [14]:
```
plt.figure(figsize=(10, 6))
plt.plot(df['NASDAQ'], label='NASDAQ')
plt.plot(y_pred, label='NASDAQ_predicted')
plt.xlabel('Date')
plt.ylabel('Adjusted Close Price')
#plt.title('NSE vs NASDAQ')
plt.legend()
plt.grid(True)
plt.show()
```



In [15]:
```
correlation1=df['NSE'].corr(df['NASDAQ_Lag'])
correlation1
```

Out[15]: 0.9516400456476988

In [16]:
```
correlation2=df['NASDAQ'].corr(df['NSE_lag'])
correlation2
```

Out[16]: 0.9508038875682409

In [17]:
```
from statsmodels.tsa.stattools import grangercausalitytests
```

In [18]:
```
data = pd.concat([df['NASDAQ'], df['NSE']], axis=1).dropna()
data.head()
```

Out[18]:

Date	NASDAQ	NSE
2010-01-04	2308.419922	5232.200195
2010-01-05	2308.709961	5277.899902
2010-01-06	2301.090088	5281.799805
2010-01-07	2300.050049	5263.100098
2010-01-08	2317.169922	5244.750000

In [19]:
```
result = grangercausalitytests(data, maxlag=10, verbose=False)
result
```

```
C:\lib\site-packages\statsmodels\tsa\stattools.py:1488: FutureWarning: verbose is deprecated since functions should not prin
t results
  warnings.warn(
```

```
Out[19]: {1: ({'ssr_ftest': (0.7183832312542275, 0.3967385774178641, 3178.0, 1),
   'ssr_chi2test': (0.7190613777909685, 0.39645196692055884, 1),
   'lrtest': (0.7189801185377291, 0.3964786526335031, 1),
   'params_ftest': (0.7183832312564801, 0.39673857741716834, 3178.0, 1.0)},
  [<statsmodels.regression.linear_model.RegressionResultsWrapper at 0x238294d0a60>,
   <statsmodels.regression.linear_model.RegressionResultsWrapper at 0x238294d04f0>,
   array([[0., 1., 0.]])]),
 2: ({'ssr_ftest': (6.136003608400947, 0.0021892951439746633, 3175.0, 2),
   'ssr_chi2test': (12.291333212418905, 0.002142747047240743, 2),
   'lrtest': (12.267640020218096, 0.002168282260093713, 2),
   'params_ftest': (6.136003608401168, 0.002189295143974222, 3175.0, 2.0)},
  [<statsmodels.regression.linear_model.RegressionResultsWrapper at 0x238294d0280>,
   <statsmodels.regression.linear_model.RegressionResultsWrapper at 0x238294d05e0>,
   array([[0., 0., 1., 0., 0.],
          [0., 0., 0., 1., 0.]])]),
 3: ({'ssr_ftest': (5.718986075556011, 0.0006697780573567369, 3172.0, 3),
   'ssr_chi2test': (17.19482036651251, 0.0006444399374741416, 3),
   'lrtest': (17.14848503757821, 0.0006587440166701158, 3),
   'params_ftest': (5.718986075555411, 0.0006697780573573091, 3172.0, 3.0)},
  [<statsmodels.regression.linear_model.RegressionResultsWrapper at 0x238294d0580>,
   <statsmodels.regression.linear_model.RegressionResultsWrapper at 0x238294d0940>,
   array([[0., 0., 0., 1., 0., 0., 0.],
          [0., 0., 0., 0., 1., 0., 0.],
          [0., 0., 0., 0., 0., 1., 0.]])]),
 4: ({'ssr_ftest': (4.484264447792296, 0.0012954642674950874, 3169.0, 4),
   'ssr_chi2test': (17.98799926170264, 0.0012407803846401894, 4),
   'lrtest': (17.937283038983878, 0.0012694174283071297, 4),
   'params_ftest': (4.484264447792612, 0.0012954642674941818, 3169.0, 4.0)},
  [<statsmodels.regression.linear_model.RegressionResultsWrapper at 0x238294d0310>,
   <statsmodels.regression.linear_model.RegressionResultsWrapper at 0x238294d0130>,
   array([[0., 0., 0., 0., 1., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 1., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 1., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 1., 0.]])]),
 5: ({'ssr_ftest': (5.2611343820819885, 8.126679837288849e-05, 3166.0, 5),
   'ssr_chi2test': (26.397068749012124, 7.472544829051807e-05, 5),
   'lrtest': (26.288008411116607, 7.846030804755128e-05, 5),
   'params_ftest': (5.261134382082496, 8.126679837279842e-05, 3166.0, 5.0)},
  [<statsmodels.regression.linear_model.RegressionResultsWrapper at 0x238294d0ca0>,
   <statsmodels.regression.linear_model.RegressionResultsWrapper at 0x238294d0070>,
   array([[0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0.]])]),
 6: ({'ssr_ftest': (5.345527211963915, 1.6816206070758035e-05, 3163.0, 6),
   'ssr_chi2test': (32.20498468263811, 1.4905532783283295e-05, 6),
   'lrtest': (32.042799127353646, 1.6012218453580527e-05, 6),
   'params_ftest': (5.345527211964043, 1.6816206070749433e-05, 3163.0, 6.0)},
  [<statsmodels.regression.linear_model.RegressionResultsWrapper at 0x238294d0e80>,
   <statsmodels.regression.linear_model.RegressionResultsWrapper at 0x238294d1000>,
   array([[0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0.]])]),
 7: ({'ssr_ftest': (4.653739496107375, 3.37259187342079e-05, 3160.0, 7),
   'ssr_chi2test': (32.73081022183114, 2.9714804978966623e-05, 7),
   'lrtest': (32.56325120937254, 3.19252161406897 4e-05, 7),
   'params_ftest': (4.653739496108044, 3.3725918734140746e-05, 3160.0, 7.0)},
  [<statsmodels.regression.linear_model.RegressionResultsWrapper at 0x238294d10c0>,
   <statsmodels.regression.linear_model.RegressionResultsWrapper at 0x238294d1240>,
   array([[0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0.]])]),
 8: ({'ssr_ftest': (4.94665968700549, 4.205898091531638e-06, 3157.0, 8),
   'ssr_chi2test': (39.78637401724531, 3.510997450740077e-06, 8),
   'lrtest': (39.53907559876825, 3.903320340460809e-06, 8),
   'params_ftest': (4.946659687005783, 4.205898091527865e-06, 3157.0, 8.0)},
  [<statsmodels.regression.linear_model.RegressionResultsWrapper at 0x238294d1300>,
   <statsmodels.regression.linear_model.RegressionResultsWrapper at 0x238294d1480>,
   array([[0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0.,
           0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0.,
           0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0.,
           0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0.,
           0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0.,
           0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0.,
           0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0.,
           0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1.,
```

```
             0.]])]),
 9: ({'ssr_ftest': (4.667267941681058, 3.6137577071032737e-06, 3154.0, 9),
   'ssr_chi2test': (42.25845612257007, 2.9476189702003046e-06, 9),
   'lrtest': (41.97952792630531, 3.3149901277903104e-06, 9),
   'params_ftest': (4.667267941680623, 3.613757707109802e-06, 3154.0, 9.0)},
  [<statsmodels.regression.linear_model.RegressionResultsWrapper at 0x23829423a90>,
   <statsmodels.regression.linear_model.RegressionResultsWrapper at 0x238294d12d0>,
   array([[0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0.,
           0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0.,
           0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0.,
           0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0.,
           0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0.,
           0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0.,
           0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1.,
           0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
           1., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
           0., 1., 0.]])]),
 10: ({'ssr_ftest': (4.495536108073155, 2.4690113203638417e-06, 3151.0, 10),
   'ssr_chi2test': (45.25496837450983, 1.956063081287496e-06, 10),
   'lrtest': (44.93517977108422, 2.2340706550784267e-06, 10),
   'params_ftest': (4.495536108073049, 2.469011320364929e-06, 3151.0, 10.0)},
  [<statsmodels.regression.linear_model.RegressionResultsWrapper at 0x238294d15d0>,
   <statsmodels.regression.linear_model.RegressionResultsWrapper at 0x238294d1750>,
   array([[0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0.,
           0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0.,
           0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0.,
           0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0.,
           0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0.,
           0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1.,
           0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
           1., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
           0., 1., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
           0., 0., 1., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
           0., 0., 0., 1., 0.]])])}
```

In [20]:

```python
for lag in result.keys():
    p_value = result[lag][0]['ssr_chi2test'][1]
    if p_value < 0.05:
        print(f"Lag {lag}: NSE leads NASDAQ (p-value: {p_value})")
    else:
        print(f"Lag {lag}: NASDAQ leads NSE (p-value: {p_value})")
```

```
Lag 1: NASDAQ leads NSE (p-value: 0.39645196692055884)
Lag 2: NSE leads NASDAQ (p-value: 0.002142747047240743)
Lag 3: NSE leads NASDAQ (p-value: 0.0006444399374741416)
Lag 4: NSE leads NASDAQ (p-value: 0.0012407803846401894)
Lag 5: NSE leads NASDAQ (p-value: 7.472544829051807e-05)
Lag 6: NSE leads NASDAQ (p-value: 1.4905532783283295e-05)
Lag 7: NSE leads NASDAQ (p-value: 2.9714804978966623e-05)
Lag 8: NSE leads NASDAQ (p-value: 3.510997450740077e-06)
Lag 9: NSE leads NASDAQ (p-value: 2.9476189702003046e-06)
Lag 10: NSE leads NASDAQ (p-value: 1.956063081287496e-06)
```

In [21]:

```python
df.head()
```

Out[21]:

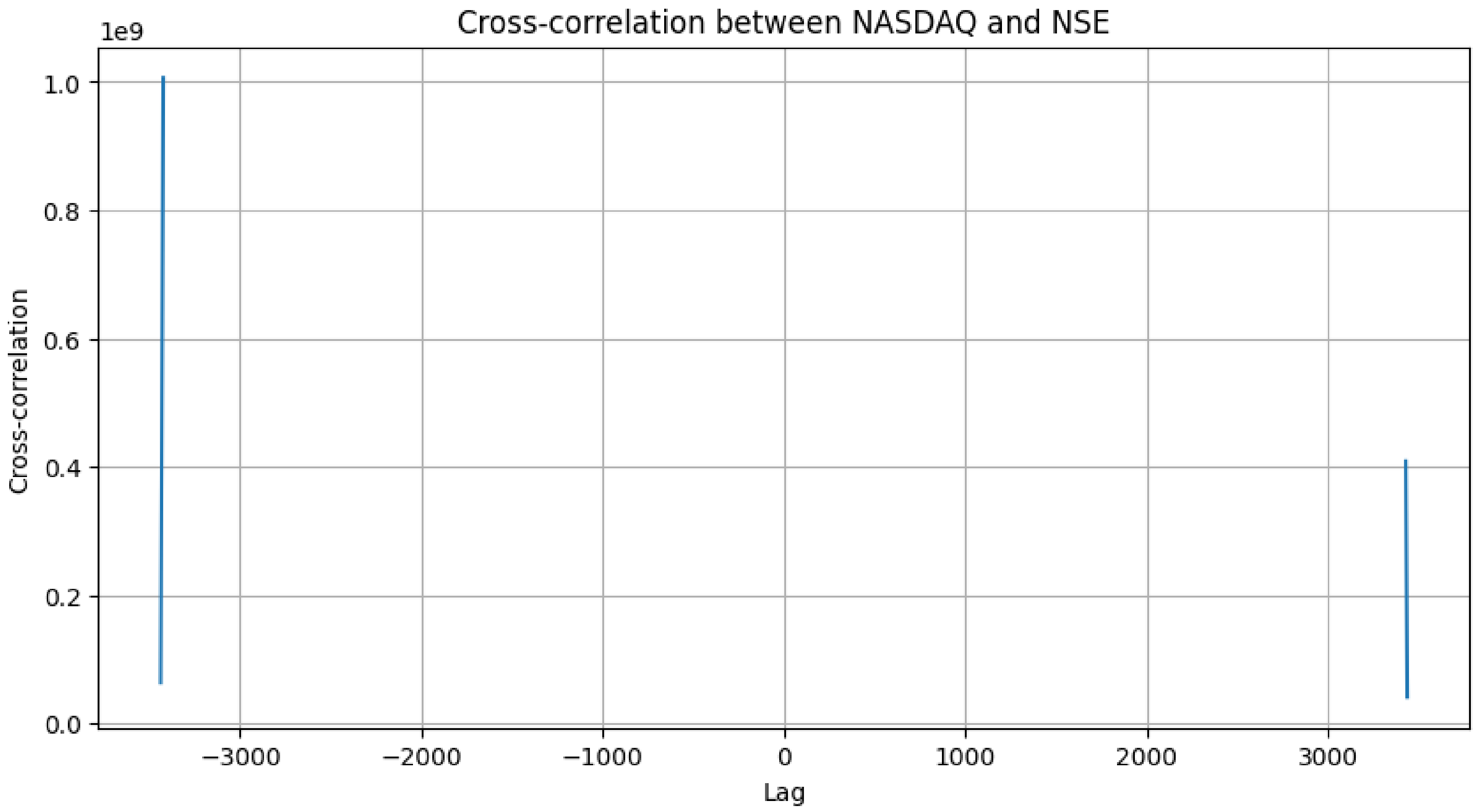
Date	NSE	NASDAQ	NASDAQ_Lag	NSE_lag
2010-01-04	5232.200195	2308.419922	NaN	NaN
2010-01-05	5277.899902	2308.709961	2308.419922	5232.200195
2010-01-06	5281.799805	2301.090088	2308.709961	5277.899902
2010-01-07	5263.100098	2300.050049	2301.090088	5281.799805
2010-01-08	5244.750000	2317.169922	2300.050049	5263.100098

In [24]:
```
nasdaq_prices = df['NSE'].values
nse_prices = df['NASDAQ'].values
```

In [25]:
```
cross_corr = np.correlate(nasdaq_prices, nse_prices, mode='full')
```

In [26]:
```
lags = np.arange(-len(nasdaq_prices) + 1, len(nasdaq_prices))
```

In [27]:
```
import matplotlib.pyplot as plt
plt.figure(figsize=(10, 5))
plt.plot(lags, cross_corr)
plt.xlabel('Lag')
plt.ylabel('Cross-correlation')
plt.title('Cross-correlation between NASDAQ and NSE')
plt.grid(True)
plt.show()
```



In [ ]:
```
#Therefore, based on the graph, we can conclude that NSE leads NASDAQ. The positive cross-correlation values further support t
```

In [ ]:
```
# Keltner Channel
```

In [ ]:

In [ ]:

In [29]:
```
start_date = '2010-01-01'
end_date = '2023-05-01'

nasdaq_data= yf.download('^IXIC',start=start_date, end=end_date)
nse_data= yf.download('^NSEI',start=start_date, end=end_date)
nasdaq_data
```

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

Out[29]:

Date	Open	High	Low	Close	Adj Close	Volume
2010-01-04	2294.409912	2311.149902	2294.409912	2308.419922	2308.419922	1931380000
2010-01-05	2307.270020	2313.729980	2295.620117	2308.709961	2308.709961	2367860000
2010-01-06	2307.709961	2314.070068	2295.679932	2301.090088	2301.090088	2253340000
2010-01-07	2298.090088	2301.300049	2285.219971	2300.050049	2300.050049	2270050000
2010-01-08	2292.239990	2317.600098	2290.610107	2317.169922	2317.169922	2145390000
...	...	...	...	...	...	...
2023-04-24	12053.469727	12103.580078	11960.299805	12037.200195	12037.200195	4854050000
2023-04-25	11968.809570	11990.459961	11798.769531	11799.160156	11799.160156	4806020000
2023-04-26	11913.230469	11967.990234	11833.070312	11854.349609	11854.349609	5281970000
2023-04-27	11972.150391	12154.009766	11950.919922	12142.240234	12142.240234	5253710000
2023-04-28	12117.540039	12227.719727	12082.570312	12226.580078	12226.580078	5331380000

3353 rows × 6 columns

In [30]:
```
nasdaq_ema=nasdaq_data['Close'].ewm(span=20,adjust=False).mean()
nasdaq_ema
```

Out[30]:
```
Date
2010-01-04     2308.419922
2010-01-05     2308.447545
2010-01-06     2307.746834
2010-01-07     2307.013807
2010-01-08     2307.981056
                  ...
2023-04-24    12014.002361
2023-04-25    11993.541199
2023-04-26    11980.284857
2023-04-27    11995.709179
2023-04-28    12017.696883
Name: Close, Length: 3353, dtype: float64
```

In [32]:
```
nasdaq_df1=abs(nasdaq_data['High']-nasdaq_data['Low'].shift(1))
nasdaq_df1
```

Out[32]:
```
Date
2010-01-04           NaN
2010-01-05     19.320068
2010-01-06     18.449951
2010-01-07      5.620117
2010-01-08     32.380127
                 ...
2023-04-24    116.759766
2023-04-25     30.160156
2023-04-26    169.220703
2023-04-27    320.939453
2023-04-28    276.799805
Length: 3353, dtype: float64
```

In [33]:
```
nasdaq_df2=abs(nasdaq_data['High']-nasdaq_data['Close'].shift(1))
nasdaq_df2
```

Out[33]:
```
Date
2010-01-04           NaN
2010-01-05      5.310059
2010-01-06      5.360107
2010-01-07      0.209961
2010-01-08     17.550049
                 ...
2023-04-24     31.120117
2023-04-25     46.740234
2023-04-26    168.830078
2023-04-27    299.660156
2023-04-28     85.479492
Length: 3353, dtype: float64
```

In [34]:
```
nasdaq_df3=abs(nasdaq_data['Low']-nasdaq_data['Close'].shift(1))
nasdaq_df3
```

Out[34]:
```
Date
2010-01-04           NaN
2010-01-05     12.799805
2010-01-06     13.030029
2010-01-07     15.870117
2010-01-08      9.439941
                 ...
2023-04-24    112.160156
2023-04-25    238.430664
2023-04-26     33.910156
2023-04-27     96.570312
2023-04-28     59.669922
Length: 3353, dtype: float64
```

In [35]:
```
nasdaq_df=pd.concat([nasdaq_df1,nasdaq_df2,nasdaq_df3],axis=1)
nasdaq_df.columns=['H_L','H_PC','L_PC']
nasdaq_df
```

Out[35]:

	H_L	H_PC	L_PC
Date			
2010-01-04	NaN	NaN	NaN
2010-01-05	19.320068	5.310059	12.799805
2010-01-06	18.449951	5.360107	13.030029
2010-01-07	5.620117	0.209961	15.870117
2010-01-08	32.380127	17.550049	9.439941
...	...	...	...
2023-04-24	116.759766	31.120117	112.160156
2023-04-25	30.160156	46.740234	238.430664
2023-04-26	169.220703	168.830078	33.910156
2023-04-27	320.939453	299.660156	96.570312
2023-04-28	276.799805	85.479492	59.669922

3353 rows × 3 columns

In [36]:
```
nasdaq_tr=nasdaq_df.max(axis=1)
nasdaq_tr
```

Out[36]:
```
Date
2010-01-04           NaN
2010-01-05     19.320068
2010-01-06     18.449951
2010-01-07     15.870117
2010-01-08     32.380127
                 ...
2023-04-24    116.759766
2023-04-25    238.430664
2023-04-26    169.220703
2023-04-27    320.939453
2023-04-28    276.799805
Length: 3353, dtype: float64
```

In [37]:
```
nasdaq_KCL=nasdaq_ema-(2*nasdaq_tr)
nasdaq_KCL
```

Out[37]:
```
Date
2010-01-04             NaN
2010-01-05     2269.807408
2010-01-06     2270.846932
2010-01-07     2275.273573
2010-01-08     2243.220802
                  ...
2023-04-24    11780.482830
2023-04-25    11516.679871
2023-04-26    11641.843451
2023-04-27    11353.830273
2023-04-28    11464.097274
Length: 3353, dtype: float64
```

In [38]:
```
nasdaq_KCU=nasdaq_ema+(2*nasdaq_tr)
nasdaq_KCU
```

Out[38]:
```
Date
2010-01-04             NaN
2010-01-05     2347.087681
2010-01-06     2344.646737
2010-01-07     2338.754042
2010-01-08     2372.741310
                  ...
2023-04-24    12247.521893
2023-04-25    12470.402527
2023-04-26    12318.726263
2023-04-27    12637.588085
2023-04-28    12571.296493
Length: 3353, dtype: float64
```

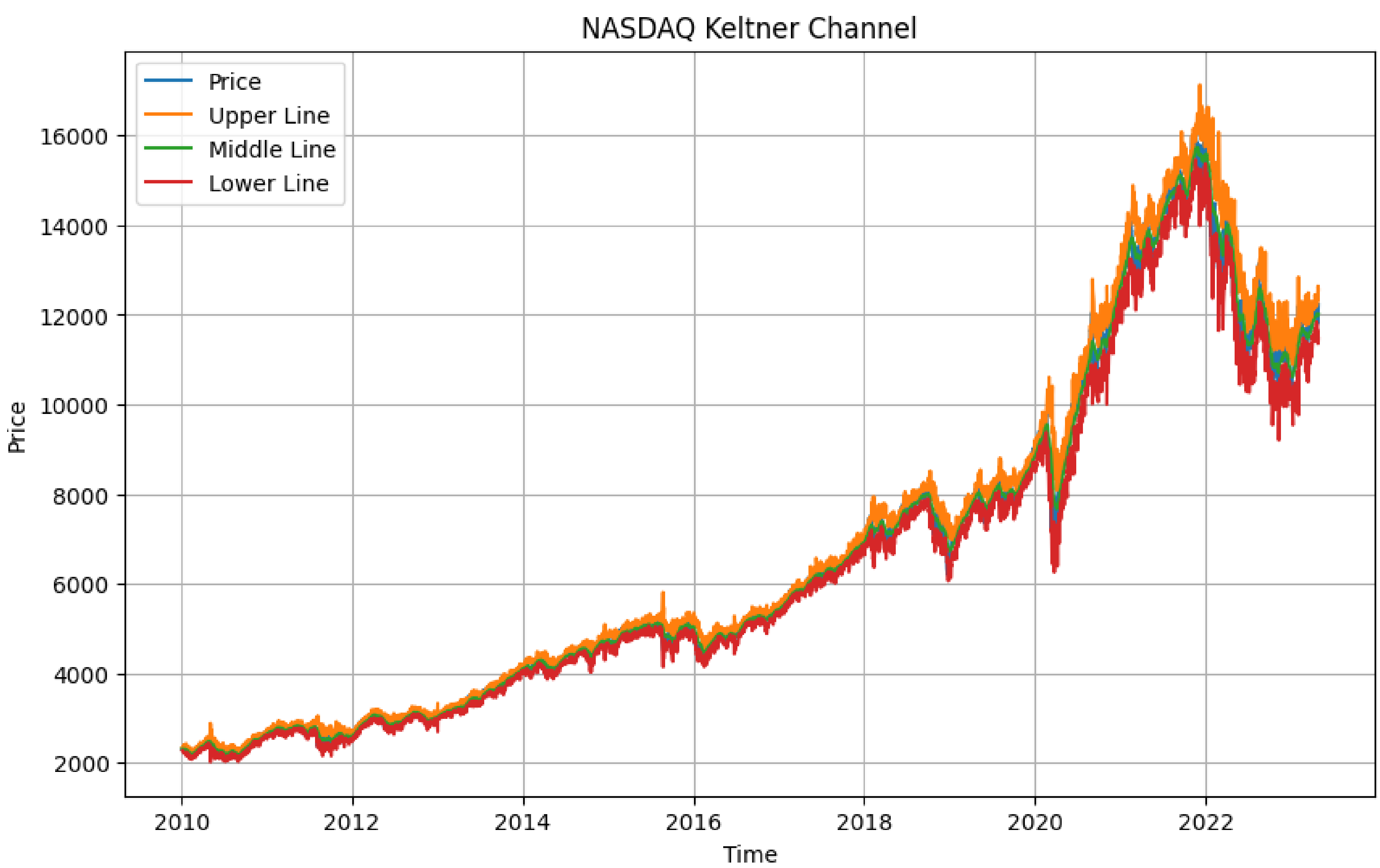
In [39]:
```
nasdaq_KC=pd.concat([nasdaq_KCL,nasdaq_ema,nasdaq_KCU],axis=1)
nasdaq_KC.columns=['nasdaq_KCL','nasdaq_ema','nasdaq_KCU']
nasdaq_KC
```

Out[39]:

	nasdaq_KCL	nasdaq_ema	nasdaq_KCU
Date			
2010-01-04	NaN	2308.419922	NaN
2010-01-05	2269.807408	2308.447545	2347.087681
2010-01-06	2270.846932	2307.746834	2344.646737
2010-01-07	2275.273573	2307.013807	2338.754042
2010-01-08	2243.220802	2307.981056	2372.741310
...	...	...	...
2023-04-24	11780.482830	12014.002361	12247.521893
2023-04-25	11516.679871	11993.541199	12470.402527
2023-04-26	11641.843451	11980.284857	12318.726263
2023-04-27	11353.830273	11995.709179	12637.588085
2023-04-28	11464.097274	12017.696883	12571.296493

3353 rows × 3 columns

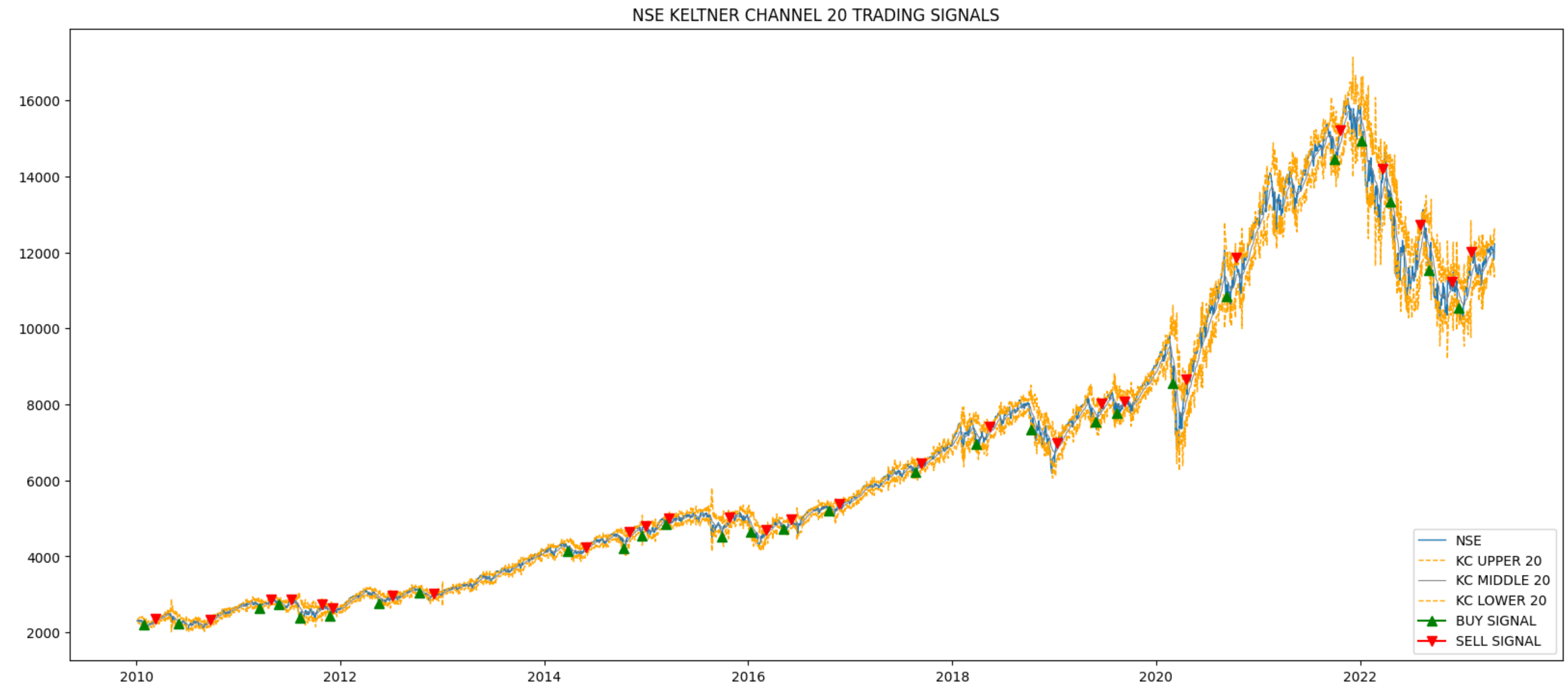
In [41]:
```
# Plotting the Keltner Channel for NASDAQ
plt.figure(figsize=(10, 6))
plt.plot(nasdaq_data['Close'], label='Price')
plt.plot(nasdaq_KCU, label='Upper Line')
plt.plot(nasdaq_ema, label='Middle Line')
plt.plot(nasdaq_KCL, label='Lower Line')
plt.title('NASDAQ Keltner Channel')
plt.xlabel('Time')
plt.ylabel('Price')
plt.legend()
plt.grid(True)
plt.show()
```



In [ ]:
```
#From the above graph of Keltner Channel, we can interpret the upper line, middle line, and lower line as follows:
#Upper Line: The upper line of the Keltner Channel represents the potential resistance level. It indicates the price level abo
#Middle Line: The middle line of the Keltner Channel is typically a moving average and represents the mean or average price ov
#Lower Line: The lower line of the Keltner Channel represents the potential support level. It indicates the price level below
```

In [43]:
```
def implement_kc_strategy(prices, kc_upper, kc_lower):
    buy_price = []
    sell_price = []
    kc_signal = []
    signal = 0

    for i in range(len(prices)-1):
        if prices[i] < kc_lower[i] and prices[i+1] > prices[i]:
            if signal != 1:
                buy_price.append(prices[i])
                sell_price.append(np.nan)
                signal = 1
                kc_signal.append(signal)
            else:
                buy_price.append(np.nan)
                sell_price.append(np.nan)
                kc_signal.append(0)
        elif prices[i] > kc_upper[i] and prices[i+1] < prices[i]:
            if signal != -1:
                buy_price.append(np.nan)
                sell_price.append(prices[i])
                signal = -1
                kc_signal.append(signal)
            else:
                buy_price.append(np.nan)
                sell_price.append(np.nan)
                kc_signal.append(0)
        else:
            buy_price.append(np.nan)
            sell_price.append(np.nan)
            kc_signal.append(0)

    return buy_price, sell_price, kc_signal

buy_price, sell_price, kc_signal = implement_kc_strategy(nasdaq_data['Close'], nasdaq_KC['nasdaq_KCU'],nasdaq_KC['nasdaq_KCL'
buy_price.append(np.nan)
sell_price.append(np.nan)
kc_signal.append(0)
```

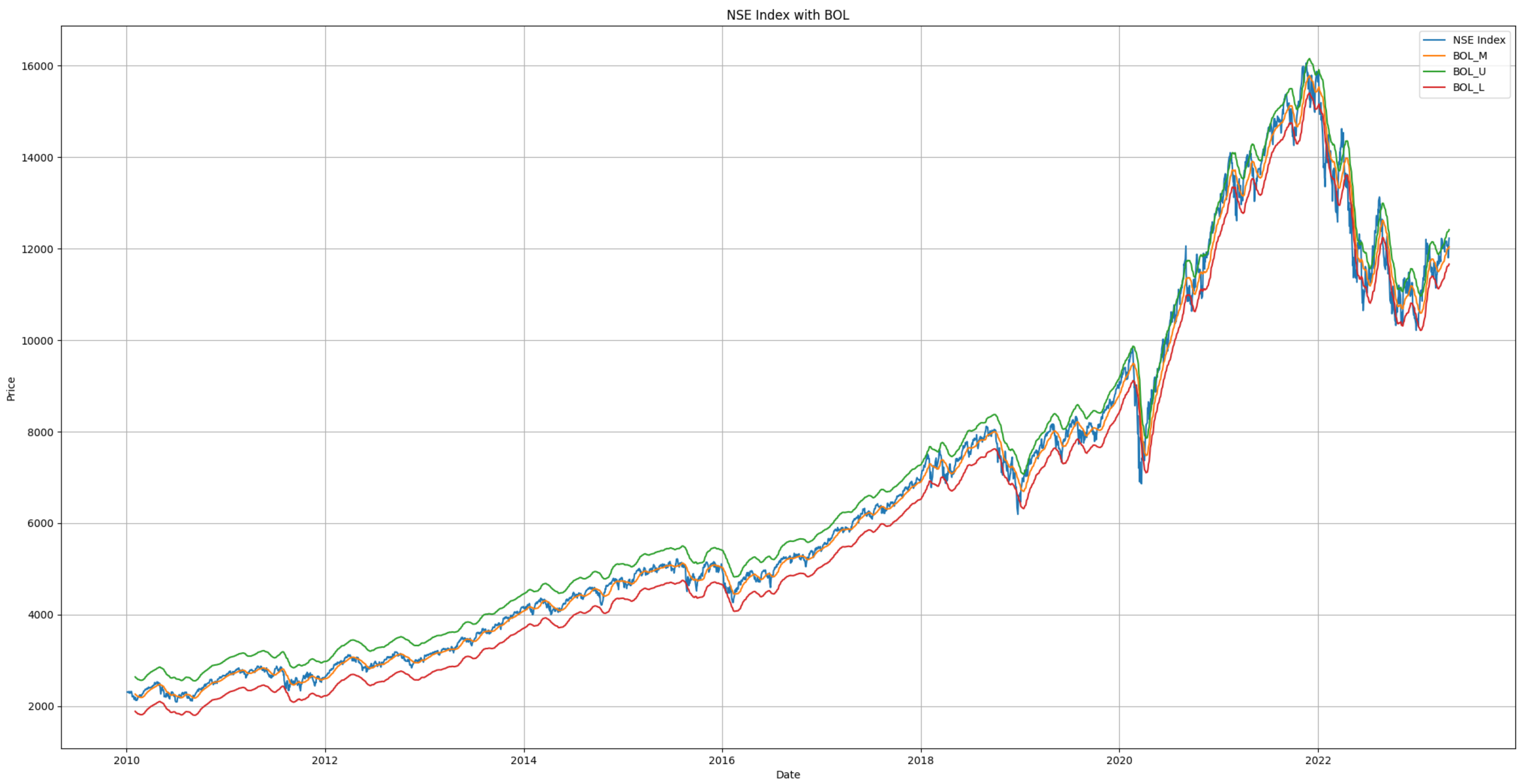
In [45]:
```
plt.figure(figsize=(20,8.5))
plt.plot(nasdaq_data['Close'], linewidth=1, label='NSE')
plt.plot(nasdaq_KC['nasdaq_KCU'], linewidth=1, color='orange', linestyle='--', label='KC UPPER 20')
plt.plot(nasdaq_KC['nasdaq_ema'], linewidth=0.75, color='grey', label='KC MIDDLE 20')
plt.plot(nasdaq_KC['nasdaq_KCL'], linewidth=1, color='orange', linestyle='--', label='KC LOWER 20')
plt.plot(nasdaq_data.index, buy_price, marker = '^', color = 'green', label = 'BUY SIGNAL', markersize=7)
plt.plot(nasdaq_data.index, sell_price, marker = 'v', color= 'r', label = 'SELL SIGNAL', markersize=7)
plt.legend(loc = 'lower right')
plt.title('NSE KELTNER CHANNEL 20 TRADING SIGNALS')
plt.show()
```



In [ ]:
```
#Bollinger Bands
```

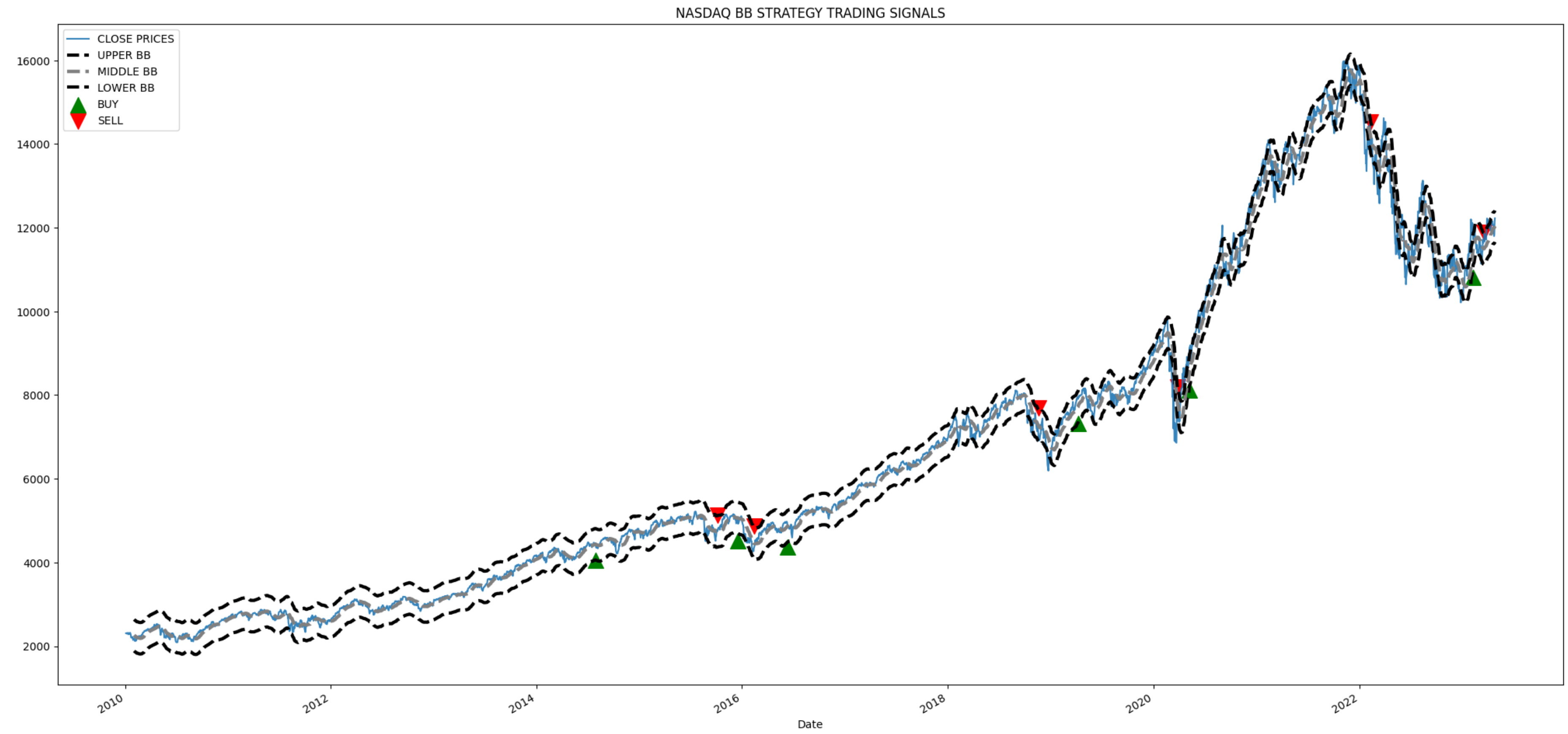
In [47]:
```
sma_period=20
data['TP']=(nasdaq_data['High']+nasdaq_data['Low']+nasdaq_data['Close'])/3
data['SMA']=data['TP'].rolling(window=sma_period).mean()
data['BOL_U']=data['SMA']+(0.1*data['TP'].std())
data['BOL_L']=data['SMA']-(0.1*data['TP'].std())
```

In [48]:
```
plt.figure(figsize=(25, 12.5))
plt.plot(nasdaq_data['Close'], label='NSE Index')
plt.plot(data['SMA'], label='BOL_M')
plt.plot(data['BOL_U'], label='BOL_U')
plt.plot(data['BOL_L'], label='BOL_L')
plt.xlabel('Date')
plt.ylabel('Price')
plt.title('NSE Index with BOL')
plt.legend()
plt.grid(True)
plt.show()
```



In [49]:
```
def implement_bb_strategy(data):
    buy_price = []
    sell_price = []
    bb_signal = []
    signal = 0

    for i in range(len(data)):
        if nasdaq_data['Close'][i-1] > data['BOL_L'][i-1] and nasdaq_data['Close'][i] < data['BOL_L'][i]:
            if signal != 1:
                buy_price.append(nasdaq_data['Close'][i])
                sell_price.append(np.nan)
                signal = 1
                bb_signal.append(signal)
            else:
                buy_price.append(np.nan)
                sell_price.append(np.nan)
                bb_signal.append(0)
        elif nasdaq_data['Close'][i-1] < data['BOL_U'][i-1] and nasdaq_data['Close'][i] > data['BOL_U'][i]:
            if signal != -1:
                buy_price.append(np.nan)
                sell_price.append(nasdaq_data['Close'][i])
                signal = -1
                bb_signal.append(signal)
            else:
                buy_price.append(np.nan)
                sell_price.append(np.nan)
                bb_signal.append(0)
        else:
            buy_price.append(np.nan)
            sell_price.append(np.nan)
            bb_signal.append(0)

    return buy_price, sell_price, bb_signal

buy_price, sell_price, bb_signal = implement_bb_strategy(data)
```

In [58]:
```
plt.figure(figsize=(25, 12.5))
nasdaq_data['Close'].plot(label = 'CLOSE PRICES', alpha = 0.9)
data['BOL_L'].plot(label = 'UPPER BB', linestyle = '--', linewidth = 3, color = 'black')
data['SMA'].plot(label = 'MIDDLE BB', linestyle = '--', linewidth = 3.2, color = 'grey')
data['BOL_U'].plot(label = 'LOWER BB', linestyle = '--', linewidth = 3, color = 'black')
plt.scatter(data.index, buy_price, marker = '^', color = 'green', label = 'BUY', s = 200)
plt.scatter(data.index, sell_price, marker = 'v', color = 'red', label = 'SELL', s = 200)
plt.title('NASDAQ BB STRATEGY TRADING SIGNALS')
plt.legend(loc = 'upper left')
plt.show()
```



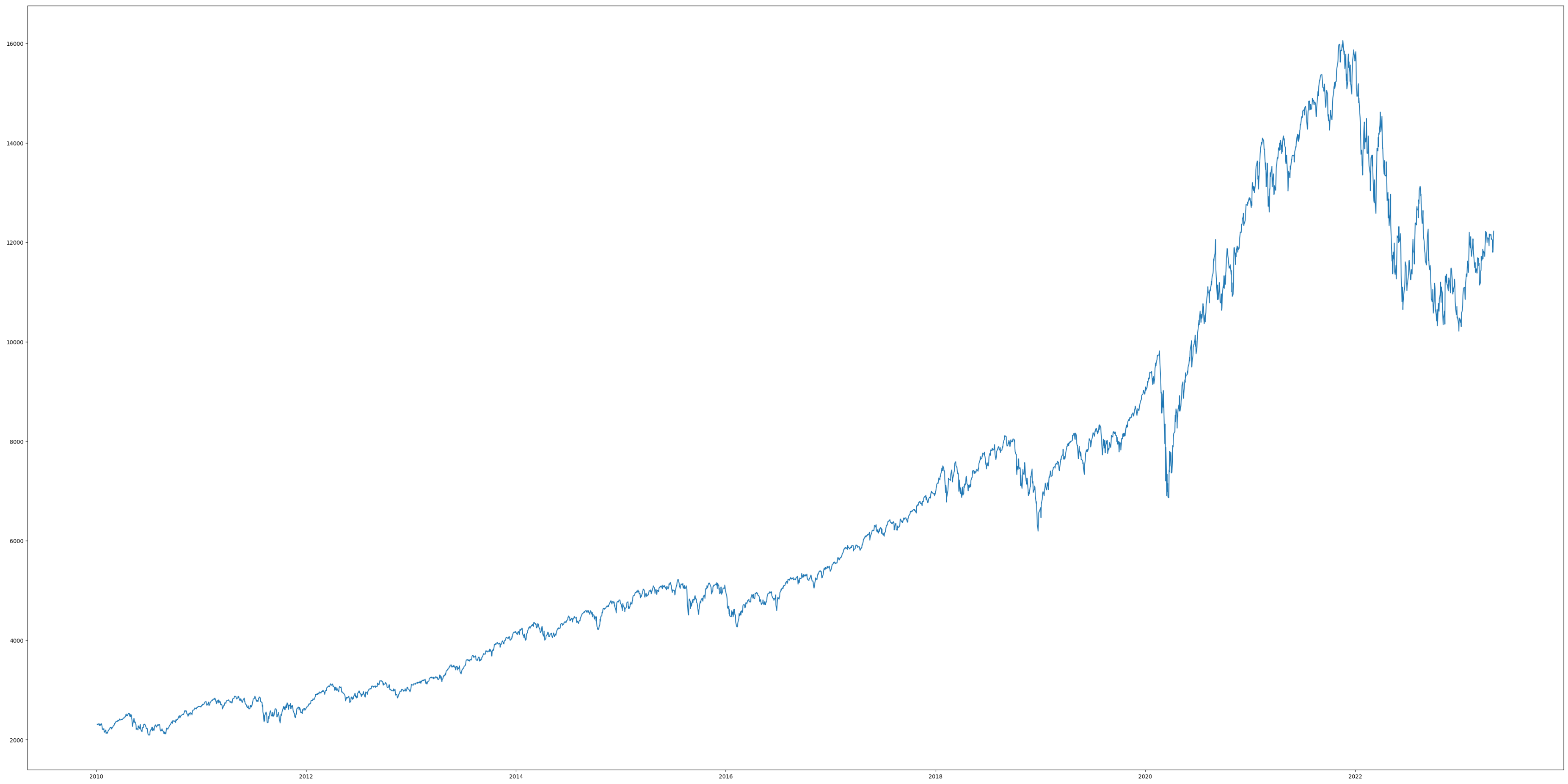
In [ ]:

In [54]:
```
#MACD
```

In [55]:
```
data['MACD']=nasdaq_data['Close'].ewm(span=15).mean()-nasdaq_data['Close'].ewm(span=50).mean()
data['Signal']=data['MACD'].ewm(span=10).mean()
data['Histogram']=data['MACD']-data['Signal']
```

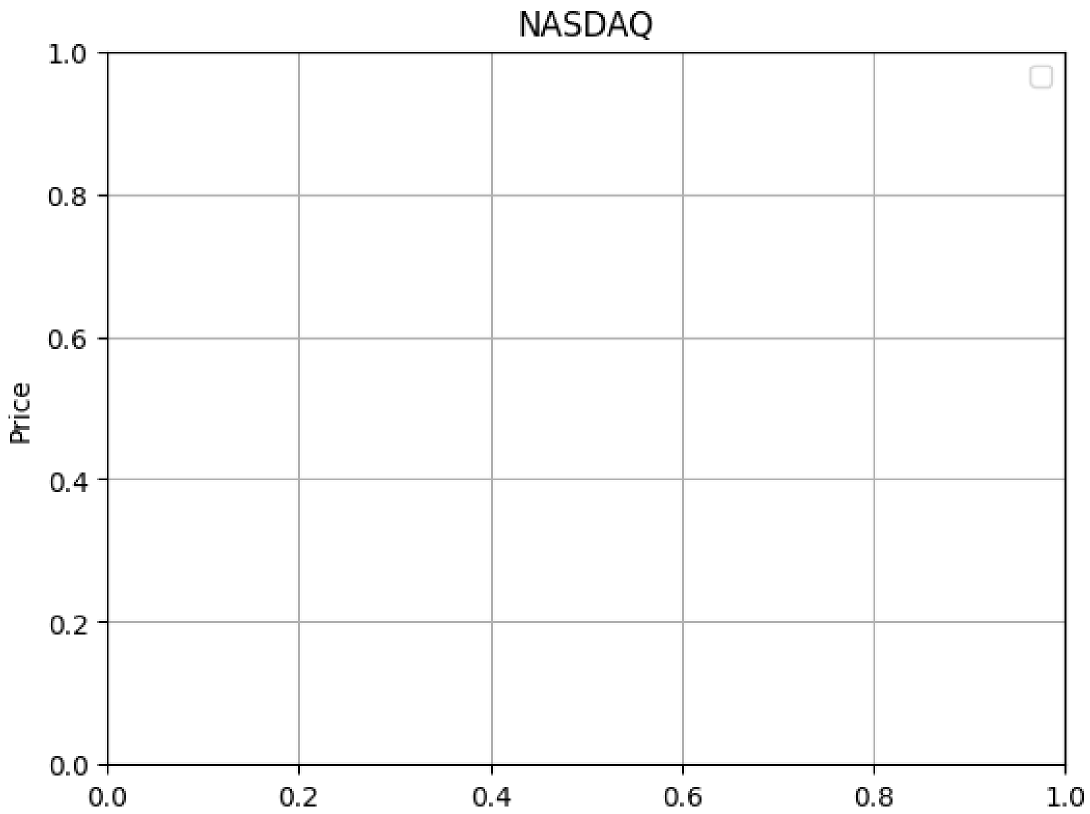
In [57]:
```
plt.figure(figsize=(50, 25))
plt.plot(nasdaq_data['Close'])
```

Out[57]: [<matplotlib.lines.Line2D at 0x2382dd48af0>]

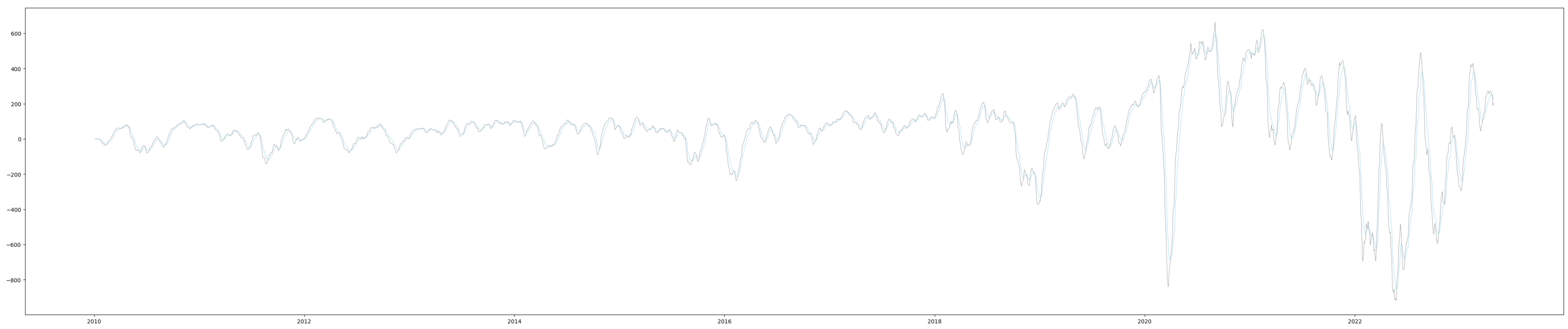


In [59]:
```python
plt.legend(loc = 'lower right')
plt.ylabel('Price')
plt.title('NASDAQ')
plt.legend()
plt.grid(True)
plt.show()

plt.figure(figsize=(50, 10))
plt.plot(data['MACD'], color = 'grey', linewidth = 0.5, label = 'MACD')
plt.plot(data['Signal'], color = 'skyblue', linewidth = 0.5, label = 'SIGNAL')
```

```
No artists with labels found to put in legend.  Note that artists whose label start with an underscore are ignored when lege
nd() is called with no argument.
No artists with labels found to put in legend.  Note that artists whose label start with an underscore are ignored when lege
nd() is called with no argument.
```

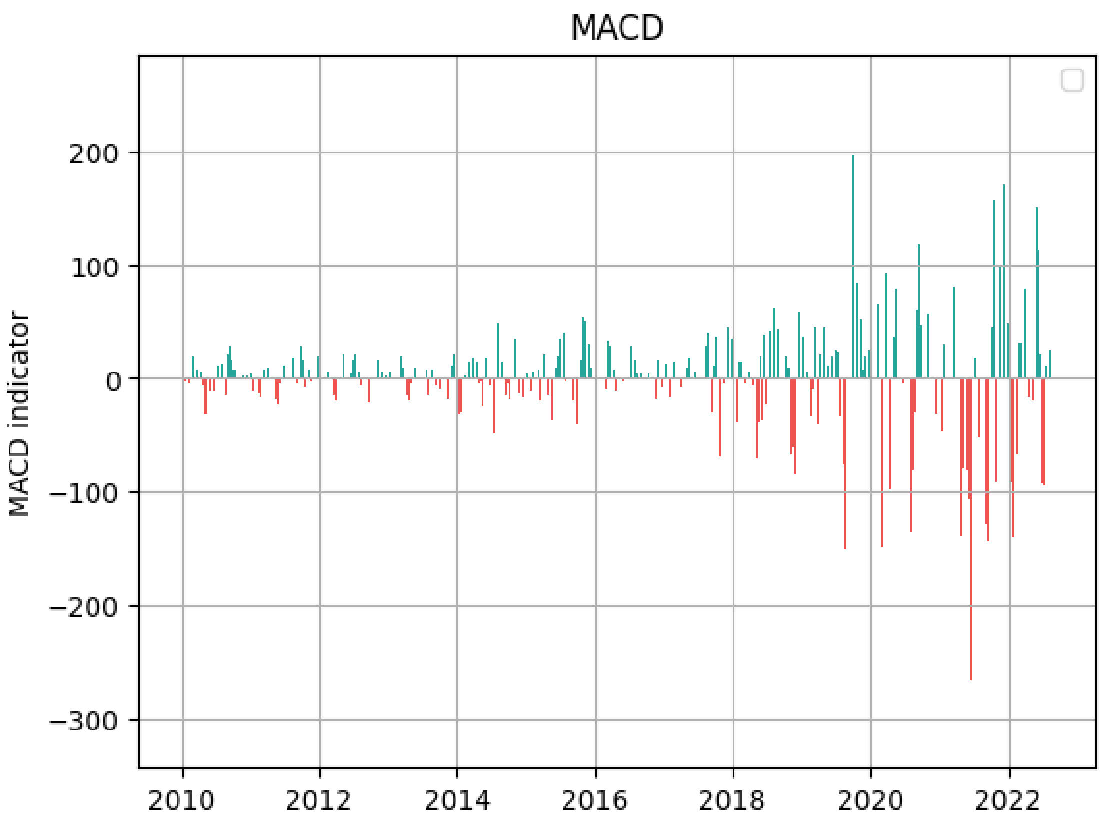


Out[59]: [<matplotlib.lines.Line2D at 0x23831f35c00>]



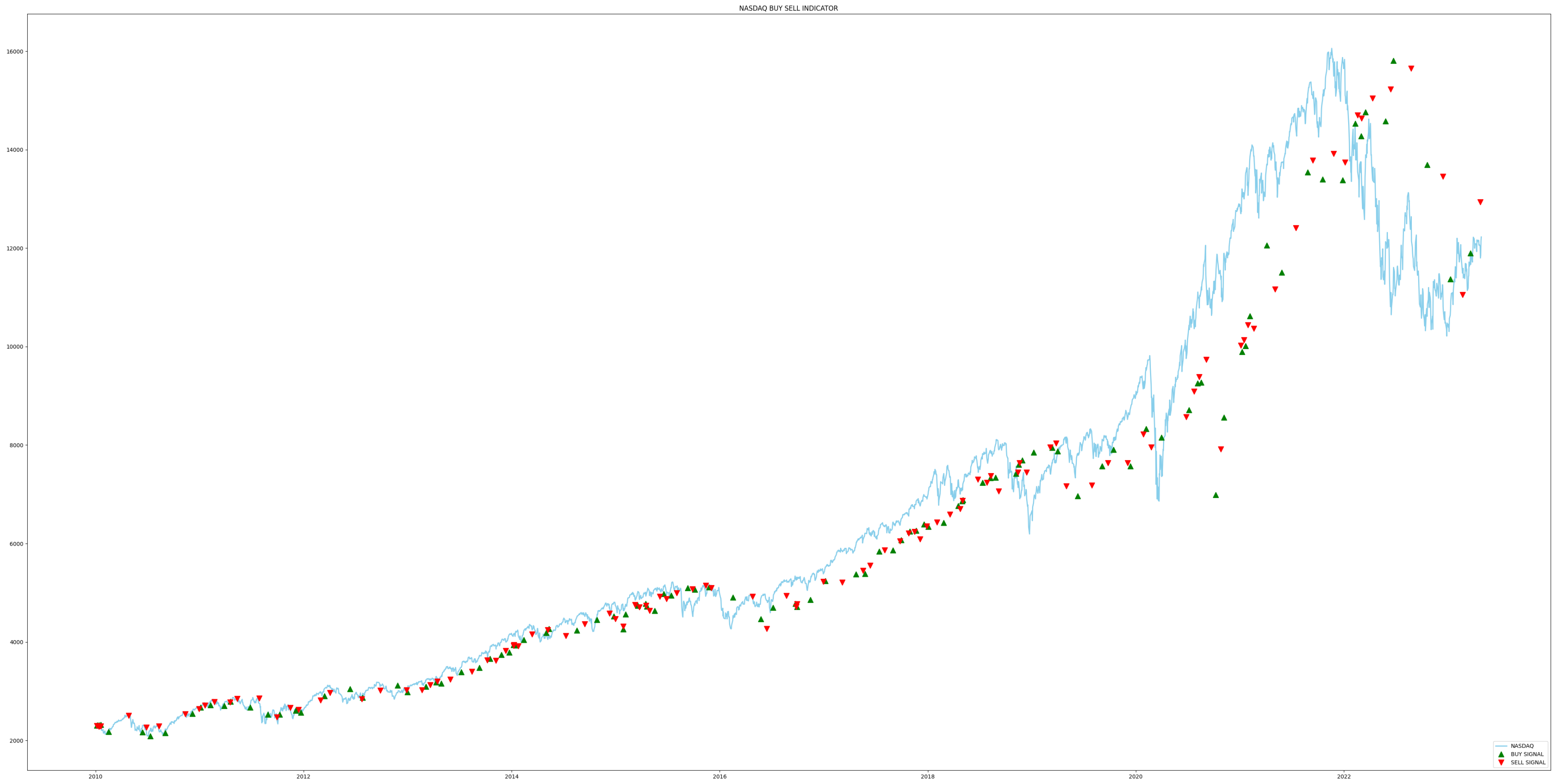
In [61]:
```
for i in range(len(data['Histogram'])):
    if str(data['Histogram'][i])[0] == '-':
        plt.bar(nasdaq_data['Close'].index[i], data['Histogram'][i], color = '#ef5350')
    else:
        plt.bar(nasdaq_data['Close'].index[i], data['Histogram'][i], color = '#26a69a')

plt.legend(loc = 'lower right')
plt.ylabel('MACD indicator')
plt.title('MACD')
plt.legend()
plt.grid(True)
plt.show()
```

```
No artists with labels found to put in legend.  Note that artists whose label start with an underscore are ignored when lege
nd() is called with no argument.
No artists with labels found to put in legend.  Note that artists whose label start with an underscore are ignored when lege
nd() is called with no argument.
```

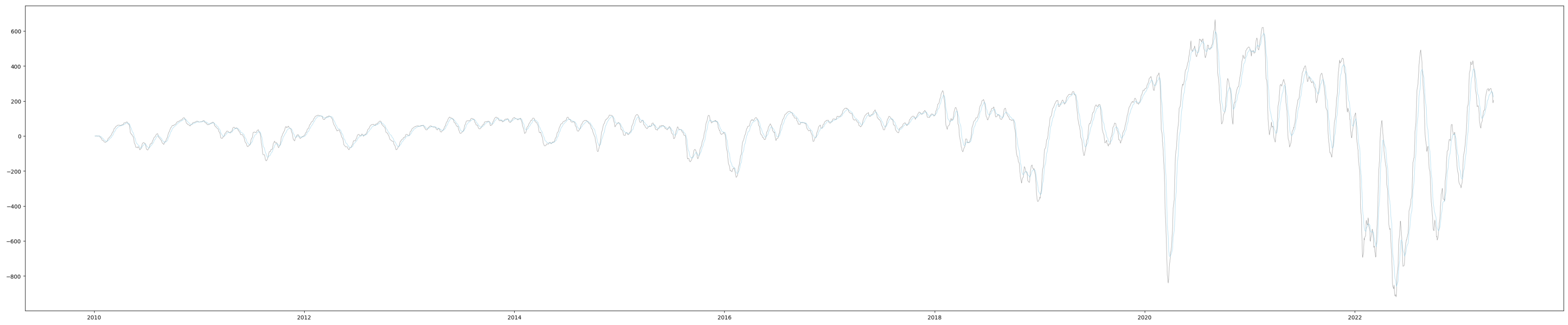


In [62]:
```
def implement_macd_strategy(data):
    buy_price = []
    sell_price = []
    macd_signal = []
    signal = 0

    for i in range(len(data)):
        if data['MACD'][i] > data['Signal'][i]:
            if signal != 1:
                buy_price.append(nasdaq_data['Close'][i])
                sell_price.append(np.nan)
                signal = 1
                macd_signal.append(signal)
            else:
                buy_price.append(np.nan)
                sell_price.append(np.nan)
                macd_signal.append(0)
        elif data['MACD'][i] < data['Signal'][i]:
            if signal != -1:
                buy_price.append(np.nan)
                sell_price.append(nasdaq_data['Close'][i])
                signal = -1
                macd_signal.append(signal)
            else:
                buy_price.append(np.nan)
                sell_price.append(np.nan)
                macd_signal.append(0)
        else:
            buy_price.append(np.nan)
            sell_price.append(np.nan)
            macd_signal.append(0)

    return buy_price, sell_price, macd_signal

buy_price, sell_price, macd_signal = implement_macd_strategy(data)
```

In [63]:
```
plt.figure(figsize=(50, 25))
plt.plot(nasdaq_data['Close'], color = 'skyblue', linewidth = 2, label = 'NASDAQ')
plt.plot(data.index, buy_price, marker = '^', color = 'green', markersize = 10, label = 'BUY SIGNAL', linewidth = 0)
plt.plot(data.index, sell_price, marker = 'v', color = 'r', markersize = 10, label = 'SELL SIGNAL', linewidth = 0)
#plt.legend()
plt.title('NASDAQ BUY SELL INDICATOR')

plt.legend(loc = 'lower right')
plt.show()

plt.figure(figsize=(50, 10))
plt.plot(data['MACD'], color = 'grey', linewidth = 0.5, label = 'MACD')
plt.plot(data['Signal'], color = 'skyblue', linewidth = 0.5, label = 'SIGNAL')
```

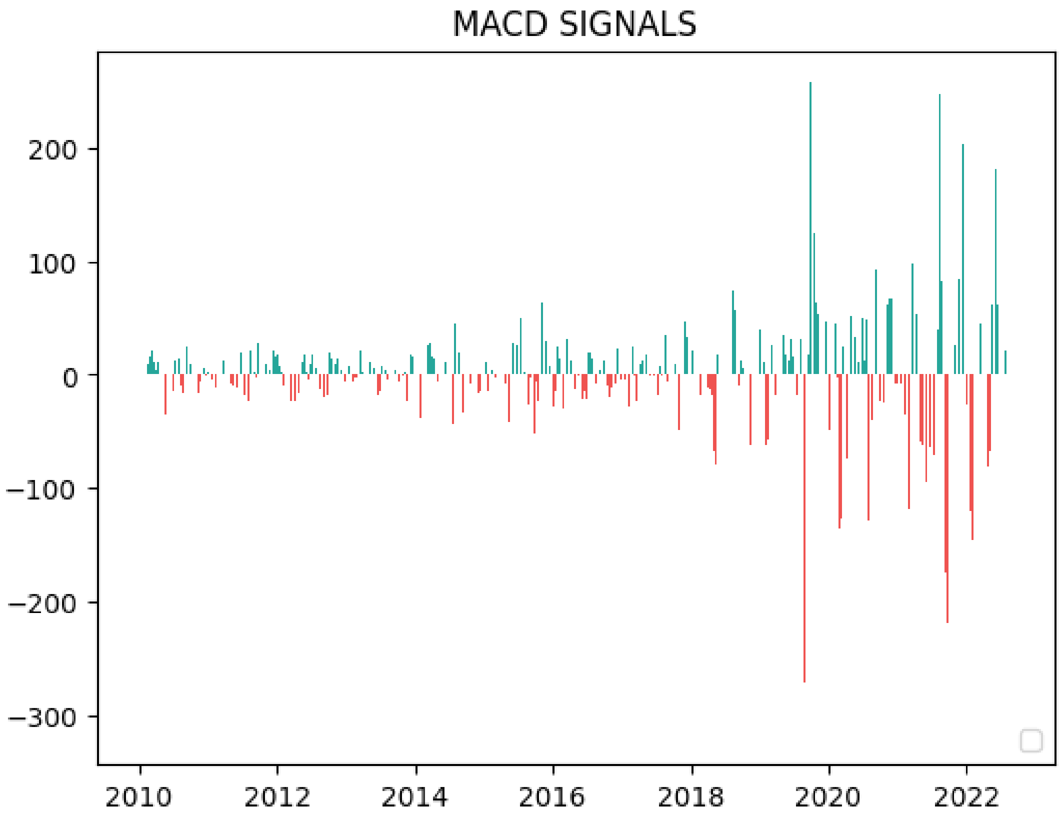


Out[63]: [<matplotlib.lines.Line2D at 0x2382f23f6d0>]



In [64]:
```
for i in range(len(data['Histogram'])):
    if str(data['Histogram'][i])[0] == '-':
        plt.bar(nasdaq_data['Close'].index[i], data['Histogram'][i], color = '#ef5350')
    else:
        plt.bar(nasdaq_data['Close'].index[i], data['Histogram'][i], color = '#26a69a')
plt.title('MACD SIGNALS')
plt.legend(loc = 'lower right')
plt.show()
```

```
No artists with labels found to put in legend.  Note that artists whose label start with an underscore are ignored when lege
nd() is called with no argument.
```



In [67]:
```
position = []
for i in range(len(data['Signal'])):
    if macd_signal[i] > 1:
        position.append(0)
    else:
        position.append(1)

for i in range(len(nasdaq_data['Close'])):
    if macd_signal[i] == 1:
        position[i] = 1
    elif macd_signal[i] == -1:
        position[i] = 0
    else:
        position[i] = position[i-1]

macd = data['MACD']
signal = data['Signal']
close_price = nasdaq_data['Close']
macd_signal = pd.DataFrame(macd_signal).rename(columns = {0:'macd_signal'}).set_index(data.index)
position = pd.DataFrame(position).rename(columns = {0:'macd_position'}).set_index(data.index)

frames = [close_price, macd, signal, macd_signal, position]
strategy = pd.concat(frames, join = 'inner', axis = 1)

strategy
```

...

In [69]:
```
get_ipython().system('pip install termcolor')
```

```
Collecting termcolor
  Downloading termcolor-2.3.0-py3-none-any.whl (6.9 kB)
Installing collected packages: termcolor
Successfully installed termcolor-2.3.0
```

In [71]:
```python
import math
from termcolor import colored as cl

NASDAQ_ret = pd.DataFrame(np.diff(nasdaq_data['Close'])).rename(columns = {0:'returns'})
macd_strategy_ret = []
for i in range(len(NASDAQ_ret)):
    try:
        returns = NASDAQ_ret['returns'][i]*strategy['macd_position'][i]
        macd_strategy_ret.append(returns)
    except:
        pass

macd_strategy_ret_df = pd.DataFrame(macd_strategy_ret).rename(columns = {0:'macd_returns'})

investment_value = 100000
number_of_stocks = math.floor(investment_value/nasdaq_data['Close'][0])
macd_investment_ret = []

for i in range(len(macd_strategy_ret_df['macd_returns'])):
    returns = number_of_stocks*macd_strategy_ret_df['macd_returns'][i]
    macd_investment_ret.append(returns)

macd_investment_ret_df = pd.DataFrame(macd_investment_ret).rename(columns = {0:'investment_returns'})
total_investment_ret = round(sum(macd_investment_ret_df['investment_returns']), 2)
profit_percentage = math.floor((total_investment_ret/investment_value)*100)
print(cl('Profit gained from the MACD strategy by investing Rs 100k in NSE : {}'.format(total_investment_ret), attrs = ['bold'
print(cl('Profit percentage of the MACD strategy : {}%'.format(profit_percentage), attrs = ['bold']))


# In[105]:


returns=macd_investment_ret_df['investment_returns']/1000
cum_ret_c=0
cum_returns=[]
for i in range(len(NASDAQ_ret)):
    if (i==0):
        cum_returns.append(returns[0])
        cum_ret_c+=returns[0]
    else:
        cum_ret_c+=returns[i]
        cum_returns.append(cum_ret_c)
cum_returns_df=pd.DataFrame(cum_returns).rename(columns = {0:'cum_returns'})
print(cum_returns_df.iloc[-1])

volatility=returns.std()*np.sqrt(252)
print(volatility)

rolling_max=cum_returns_df.rolling(window=len(cum_returns_df), min_periods=1).max()
drawdown=(cum_returns_df/rolling_max)-1
max_drawdown=drawdown.min()
print(max_drawdown)

sharpe_ratio=(returns.mean()-(0.02/252))*np.sqrt(252)/returns.std()
print(sharpe_ratio)
```

...

In [ ]: