In [2]:
```
# Importing libraries
import pandas as pd
import plotly.express as px
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm
import statsmodels.formula.api as smf
from sklearn.linear_model import LinearRegression
```

In [3]:
```
# Load .csv File
df= pd.read_csv('US_Stock_Data.csv')
```

In [4]:
```
# view data types and basic structure
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1013 entries, 0 to 1012
Data columns (total 39 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Unnamed: 0         1013 non-null   int64  
 1   Date               1013 non-null   object 
 2   Natural_Gas_Price  1013 non-null   float64
 3   Natural_Gas_Vol.   1009 non-null   float64
 4   Crude_oil_Price    1013 non-null   float64
 5   Crude_oil_Vol.     990 non-null    float64
 6   Copper_Price       1013 non-null   float64
 7   Copper_Vol.        976 non-null    float64
 8   Bitcoin_Price      1013 non-null   object 
 9   Bitcoin_Vol.       1013 non-null   int64  
 10  Platinum_Price     1013 non-null   object 
 11  Platinum_Vol.      636 non-null    float64
 12  Ethereum_Price     1013 non-null   object 
 13  Ethereum_Vol.      1013 non-null   int64  
 14  S&P_500_Price      1013 non-null   object 
 15  Nasdaq_100_Price   1013 non-null   object 
 16  Nasdaq_100_Vol.    1012 non-null   float64
 17  Apple_Price        1013 non-null   float64
 18  Apple_Vol.         1013 non-null   int64  
 19  Tesla_Price        1013 non-null   float64
 20  Tesla_Vol.         1013 non-null   int64  
 21  Microsoft_Price    1013 non-null   float64
 22  Microsoft_Vol.     1013 non-null   int64  
 23  Silver_Price       1013 non-null   float64
 24  Silver_Vol.        967 non-null    float64
 25  Google_Price       1013 non-null   float64
 26  Google_Vol.        1013 non-null   int64  
 27  Nvidia_Price       1013 non-null   float64
 28  Nvidia_Vol.        1013 non-null   int64  
 29  Berkshire_Price    1013 non-null   object 
 30  Berkshire_Vol.     1013 non-null   int64  
 31  Netflix_Price      1013 non-null   float64
 32  Netflix_Vol.       1013 non-null   int64  
 33  Amazon_Price       1013 non-null   float64
 34  Amazon_Vol.        1013 non-null   int64  
 35  Meta_Price         1013 non-null   float64
 36  Meta_Vol.          1013 non-null   int64  
 37  Gold_Price         1013 non-null   object 
 38  Gold_Vol.          1011 non-null   float64
dtypes: float64(19), int64(12), object(8)
memory usage: 308.8+ KB
```

In [4]:
```
list (df.columns.values)
```

Out[4]:
```
['Unnamed: 0',
 'Date',
 'Natural_Gas_Price',
 'Natural_Gas_Vol.',
 'Crude_oil_Price',
 'Crude_oil_Vol.',
 'Copper_Price',
 'Copper_Vol.',
 'Bitcoin_Price',
 'Bitcoin_Vol.',
 'Platinum_Price',
 'Platinum_Vol.',
 'Ethereum_Price',
 'Ethereum_Vol.',
 'S&P_500_Price',
 'Nasdaq_100_Price',
 'Nasdaq_100_Vol.',
 'Apple_Price',
 'Apple_Vol.',
 'Tesla_Price',
 'Tesla_Vol.',
 'Microsoft_Price',
 'Microsoft_Vol.',
 'Silver_Price',
 'Silver_Vol.',
 'Google_Price',
 'Google_Vol.',
 'Nvidia_Price',
 'Nvidia_Vol.',
 'Berkshire_Price',
 'Berkshire_Vol.',
 'Netflix_Price',
 'Netflix_Vol.',
 'Amazon_Price',
 'Amazon_Vol.',
 'Meta_Price',
 'Meta_Vol.',
 'Gold_Price',
 'Gold_Vol.']
```

In [5]:
```
# Check if any null values
print(df.isnull().sum())
```

```
Unnamed: 0            0
Date                  0
Natural_Gas_Price     0
Natural_Gas_Vol.      4
Crude_oil_Price       0
Crude_oil_Vol.       23
Copper_Price          0
Copper_Vol.          37
Bitcoin_Price         0
Bitcoin_Vol.          0
Platinum_Price        0
Platinum_Vol.       377
Ethereum_Price        0
Ethereum_Vol.         0
S&P_500_Price         0
Nasdaq_100_Price      0
Nasdaq_100_Vol.       1
Apple_Price           0
Apple_Vol.            0
Tesla_Price           0
Tesla_Vol.            0
Microsoft_Price       0
Microsoft_Vol.        0
Silver_Price          0
Silver_Vol.          46
Google_Price          0
Google_Vol.           0
Nvidia_Price          0
Nvidia_Vol.           0
Berkshire_Price       0
Berkshire_Vol.        0
Netflix_Price         0
Netflix_Vol.          0
Amazon_Price          0
Amazon_Vol.           0
Meta_Price            0
Meta_Vol.             0
Gold_Price            0
Gold_Vol.             2
dtype: int64
```

In [6]:
```
# Drop rows with any null
df = df.dropna()
```

In [7]:
```
# Check if any null values
print(df.isnull().sum())
```

```
Unnamed: 0           0
Date                 0
Natural_Gas_Price    0
Natural_Gas_Vol.     0
Crude_oil_Price      0
Crude_oil_Vol.       0
Copper_Price         0
Copper_Vol.          0
Bitcoin_Price        0
Bitcoin_Vol.         0
Platinum_Price       0
Platinum_Vol.        0
Ethereum_Price       0
Ethereum_Vol.        0
S&P_500_Price        0
Nasdaq_100_Price     0
Nasdaq_100_Vol.      0
Apple_Price          0
Apple_Vol.           0
Tesla_Price          0
Tesla_Vol.           0
Microsoft_Price      0
Microsoft_Vol.       0
Silver_Price         0
Silver_Vol.          0
Google_Price         0
Google_Vol.          0
Nvidia_Price         0
Nvidia_Vol.          0
Berkshire_Price      0
Berkshire_Vol.       0
Netflix_Price        0
Netflix_Vol.         0
Amazon_Price         0
Amazon_Vol.          0
Meta_Price           0
Meta_Vol.            0
Gold_Price           0
Gold_Vol.            0
dtype: int64
```

In [8]:

```
# view data types and basic structure
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 609 entries, 28 to 867
Data columns (total 39 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Unnamed: 0         609 non-null    int64
 1   Date               609 non-null    object
 2   Natural_Gas_Price  609 non-null    float64
 3   Natural_Gas_Vol.   609 non-null    float64
 4   Crude_oil_Price    609 non-null    float64
 5   Crude_oil_Vol.     609 non-null    float64
 6   Copper_Price       609 non-null    float64
 7   Copper_Vol.        609 non-null    float64
 8   Bitcoin_Price      609 non-null    object
 9   Bitcoin_Vol.       609 non-null    int64
 10  Platinum_Price     609 non-null    object
 11  Platinum_Vol.      609 non-null    float64
 12  Ethereum_Price     609 non-null    object
 13  Ethereum_Vol.      609 non-null    int64
 14  S&P_500_Price      609 non-null    object
 15  Nasdaq_100_Price   609 non-null    object
 16  Nasdaq_100_Vol.    609 non-null    float64
 17  Apple_Price        609 non-null    float64
 18  Apple_Vol.         609 non-null    int64
 19  Tesla_Price        609 non-null    float64
 20  Tesla_Vol.         609 non-null    int64
 21  Microsoft_Price    609 non-null    float64
 22  Microsoft_Vol.     609 non-null    int64
 23  Silver_Price       609 non-null    float64
 24  Silver_Vol.        609 non-null    float64
 25  Google_Price       609 non-null    float64
 26  Google_Vol.        609 non-null    int64
 27  Nvidia_Price       609 non-null    float64
 28  Nvidia_Vol.        609 non-null    int64
 29  Berkshire_Price    609 non-null    object
 30  Berkshire_Vol.     609 non-null    int64
 31  Netflix_Price      609 non-null    float64
 32  Netflix_Vol.       609 non-null    int64
 33  Amazon_Price       609 non-null    float64
 34  Amazon_Vol.        609 non-null    int64
 35  Meta_Price         609 non-null    float64
 36  Meta_Vol.          609 non-null    int64
 37  Gold_Price         609 non-null    object
 38  Gold_Vol.          609 non-null    float64
dtypes: float64(19), int64(12), object(8)
memory usage: 190.3+ KB
```

In [9]:

```
df.head()
```

Out[9]:

	Unnamed: 0	Date	Natural_Gas_Price	Natural_Gas_Vol.	Crude_oil_Price	Crude_oil_
28	28	21-12-2023	2.572	84550.0	73.89	251
29	29	20-12-2023	2.447	125260.0	74.22	273
30	30	19-12-2023	2.492	170440.0	73.44	256
31	31	18-12-2023	2.503	154300.0	72.47	739
32	32	15-12-2023	2.491	189240.0	71.43	95

5 rows × 39 columns

In [10]:

```
print(df.head().to_string(header=False, index=False))
```

```
28 21-12-2023 2.572  84550.0 73.89 251980.0 3.9175 70080.0 43,865.90 48960 9
70.3 26550.0 2,239.62 471460 4,746.75 16,757.41 217170000.0 194.68  44080000
254.50 108960000 373.54 17630000 24.585 46760.0 140.42 27400000 489.90 29920
000 5,41,000  7700 491.61 2750000 153.84  35950000 354.09 15220000 2,041.80
540.0
29 20-12-2023 2.447 125260.0 74.22 273360.0 3.9060 66320.0 43,662.80 70190
974 30010.0 2,202.19 440350 4,701.19 16,554.16 275300000.0 194.83  50130000
247.14 124130000 370.62 26020000 24.631 46980.0 138.34 48940000 481.11 39400
000 5,43,740  8150 489.27 4520000 152.12  50000000 349.28 15990000 2,038.10
260.0
30 19-12-2023 2.492 170440.0 73.44  25690.0 3.8980 84950.0 42,259.30 55290 9
65.8 25860.0 2,177.44 400940 4,768.37 16,811.86 228940000.0 196.94  40230000
257.22 106290000 373.26 20530000 24.321 37540.0 136.65 25440000 496.04 46310
000 5,54,650  7500 495.02 3840000 153.79  42890000 350.36 17660000 2,042.60
470.0
31 18-12-2023 2.503 154300.0 72.47  73940.0 3.8520 54990.0 42,659.70 61580 9
54.3 26230.0 2,218.80 388260 4,740.56 16,729.80 249620000.0 195.89  55750000
252.08 116420000 372.65 21800000 24.107 42680.0 135.80 32260000 500.77 41260
000 5,51,182 10460 486.12 6410000 154.07  62510000 344.62 18360000 2,030.90
250.0
32 15-12-2023 2.491 189240.0 71.43  95510.0 3.8905 73670.0 41,929.00 45280 9
52.6 38070.0 2,220.41 349630 4,719.19 16,623.45 982560000.0 197.57 128540000
253.50 135930000 370.73 78500000 24.154 57000.0 132.60 50850000 488.90 47990
000 5,44,478  8430 472.06 7840000 149.97 110090000 334.92 31780000 2,026.00
630.0
```

In [11]:
```python
df.tail()
```

Out[11]:

	Unnamed: 0	Date	Natural_Gas_Price	Natural_Gas_Vol.	Crude_oil_Price	Cruc
863	863	7/8/2020	2.238	206250.0	41.22	
864	864	6/8/2020	2.165	161990.0	41.95	
865	865	5/8/2020	2.191	182430.0	42.19	
866	866	4/8/2020	2.193	230890.0	41.70	
867	867	3/8/2020	2.101	381970.0	41.01	

5 rows × 39 columns

In [12]:
```python
print(df.tail().to_string(header=False, index=False))
```

```
863 7/8/2020 2.238 206250.0 41.22 399000.0 2.8020 310.0 11,592.00 517000
977.4 340.0 379.57 8340000 3,351.28 11,139.39 178260000.0 111.11 198050000 9
6.85 133450000 212.48 27820000 27.540 283920.0 74.92 27730000 112.00 3425000
0 3,14,334 440 494.73 5910000 158.37  78720000 268.44 72770000 2,028.00 3981
30.0
864 6/8/2020 2.165 161990.0 41.95 359610.0 2.9190 290.0 11,757.10 554850 1,0
20.80 820.0 394.83 7920000 3,349.16 11,267.08 166940000.0 113.90 202430000 9
9.31  89880000 216.35 32660000 28.400 236120.0 75.25 33310000 113.36 2443000
0 3,07,455 440 509.08 3730000 161.25  78810000 265.28 45240000 2,069.40 3127
60.0
865 5/8/2020 2.191 182430.0 42.19 491270.0 2.9250  30.0 11,735.10 570830
994.9 260.0 400.79 8740000 3,327.77 11,125.44 153580000.0 110.06 121990000 9
9.00  74670000 212.94 28860000 26.890 212520.0 73.95 29150000 112.87 2505000
0 3,05,200 560 502.11 4310000 160.25  78600000 249.12 13090000 2,049.30 3663
80.0
866 4/8/2020 2.193 230890.0 41.70 451580.0 2.9030  50.0 11,184.70 485790
961 200.0 389.62 9840000 3,306.51 11,096.54 187550000.0 109.67 172790000 99.
13 126220000 213.29 49280000 26.028 157040.0 73.67 37210000 112.28 31030000
3,00,330 270 509.64 5610000 156.94  93890000 249.83 17180000 2,021.00 27442
0.0
867 3/8/2020 2.101 381970.0 41.01 338330.0 2.9205  40.0 11,224.40 470240
938  90.0  385.8 9920000 3,294.61 11,055.08 188620000.0 108.94 308150000 99.
00 132140000 216.54 78980000 24.417  78820.0 74.14 45520000 110.10 41300000
2,98,800 470 498.62 5880000 155.59 101490000 251.96 23130000 1,986.30 17875
0.0
```

In [13]:
```
# Summary statistics
df.describe().round(0)
```

Out[13]:

	Unnamed: 0	Natural_Gas_Price	Natural_Gas_Vol.	Crude_oil_Price	Crude_oil_Vol
count	609.0	609.0	609.0	609.0	609.(
mean	528.0	4.0	127938.0	74.0	338257.(
std	223.0	2.0	56208.0	21.0	142336.(
min	28.0	2.0	2350.0	36.0	17020.(
25%	393.0	3.0	92080.0	62.0	279000.(
50%	552.0	3.0	125390.0	73.0	349080.(
75%	710.0	5.0	159820.0	87.0	421770.(
max	867.0	10.0	381970.0	124.0	722480.(

8 rows × 31 columns

In [14]:
```
print(df.describe().round(0).to_string(header=False, index=False))
```

```
609.0 609.0    609.0 609.0    609.0 609.0    609.0        609.0   609.0
609.0       609.0 609.0       609.0 609.0       609.0 609.0      609.0 609.0
609.0 609.0       609.0 609.0       609.0   609.0 609.0       609.0 609.0
609.0 609.0       609.0    609.0
528.0   4.0 127938.0  74.0 338257.0   4.0  29899.0   80110861.0  8637.0   28
153227.0 227534089.0 148.0  94598785.0 245.0 104910690.0 274.0 28492874.0  2
4.0  66682.0 116.0  32586223.0 216.0  45025813.0  2388.0 446.0   6464269.0 1
50.0  71898112.0 270.0  23777553.0 189749.0
223.0   2.0  56208.0  21.0 142336.0   1.0  38977.0  415806239.0  8756.0  188
917026.0  86844503.0  22.0  42209925.0  63.0  58481994.0  44.0 10188901.0
2.0  38283.0  22.0  13179950.0  99.0  19122496.0  2199.0 135.0   7694205.0
22.0  29305811.0  65.0  13739298.0  79755.0
 28.0   2.0   2350.0  36.0  17020.0   3.0     20.0        260.0     0.0
75180.0  68570000.0 107.0  24040000.0  92.0  29400000.0 200.0  9380000.0  1
8.0      0.0  70.0   9310000.0 108.0   9790000.0   130.0 166.0   1140000.0
91.0  21620000.0 109.0   5470000.0    160.0
393.0   3.0  92080.0  62.0 279000.0   4.0    280.0      71430.0   830.0
619620.0 177200000.0 130.0  67810000.0 210.0  67970000.0 240.0 21900000.0  2
3.0  47280.0 101.0  23970000.0 138.0  29750000.0   980.0 362.0   3270000.0 1
38.0  51960000.0 215.0  15800000.0 147260.0
552.0   3.0 125390.0  73.0 349080.0   4.0   1620.0     106440.0  4900.0    1
070000.0 212830000.0 148.0  85590000.0 241.0  90850000.0 272.0 26120000.0  2
4.0  60020.0 118.0  29760000.0 188.0  43150000.0  1810.0 489.0   4710000.0 1
57.0  64300000.0 276.0  20520000.0 177710.0
710.0   5.0 159820.0  87.0 421770.0   4.0  63830.0     228650.0 14020.0    2
320000.0 251200000.0 165.0 109300000.0 287.0 123520000.0 303.0 33110000.0  2
6.0  76730.0 136.0  37320000.0 247.0  56390000.0  2680.0 534.0   7090000.0 1
66.0  84250000.0 327.0  28130000.0 229880.0
867.0  10.0 381970.0 124.0 722480.0   5.0 176040.0 4470000000.0 42830.0 1790
000000.0 982560000.0 198.0 345940000.0 410.0 666380000.0 383.0 90430000.0  2
9.0 355280.0 150.0 123200000.0 504.0 146370000.0 10660.0 692.0 133390000.0 1
87.0 272660000.0 382.0 188120000.0 565000.0
```

In [15]:
```
#format inconcistent date values
df['Date'] = pd.to_datetime(df['Date'], format='mixed', dayfirst=True, error
```

In [16]:
```
#Check for any null dates.
print(df[df['Date'].isnull()])
```

```
Empty DataFrame
Columns: [Unnamed: 0, Date, Natural_Gas_Price, Natural_Gas_Vol., Crude_oil_P
rice, Crude_oil_Vol., Copper_Price, Copper_Vol., Bitcoin_Price, Bitcoin_Vo
l., Platinum_Price, Platinum_Vol., Ethereum_Price, Ethereum_Vol., S&P_500_Pr
ice, Nasdaq_100_Price, Nasdaq_100_Vol., Apple_Price, Apple_Vol., Tesla_Pric
e, Tesla_Vol., Microsoft_Price, Microsoft_Vol., Silver_Price, Silver_Vol., G
oogle_Price, Google_Vol., Nvidia_Price, Nvidia_Vol., Berkshire_Price, Berksh
ire_Vol., Netflix_Price, Netflix_Vol., Amazon_Price, Amazon_Vol., Meta_Pric
e, Meta_Vol., Gold_Price, Gold_Vol.]
Index: []

[0 rows x 39 columns]
```

In [20]:

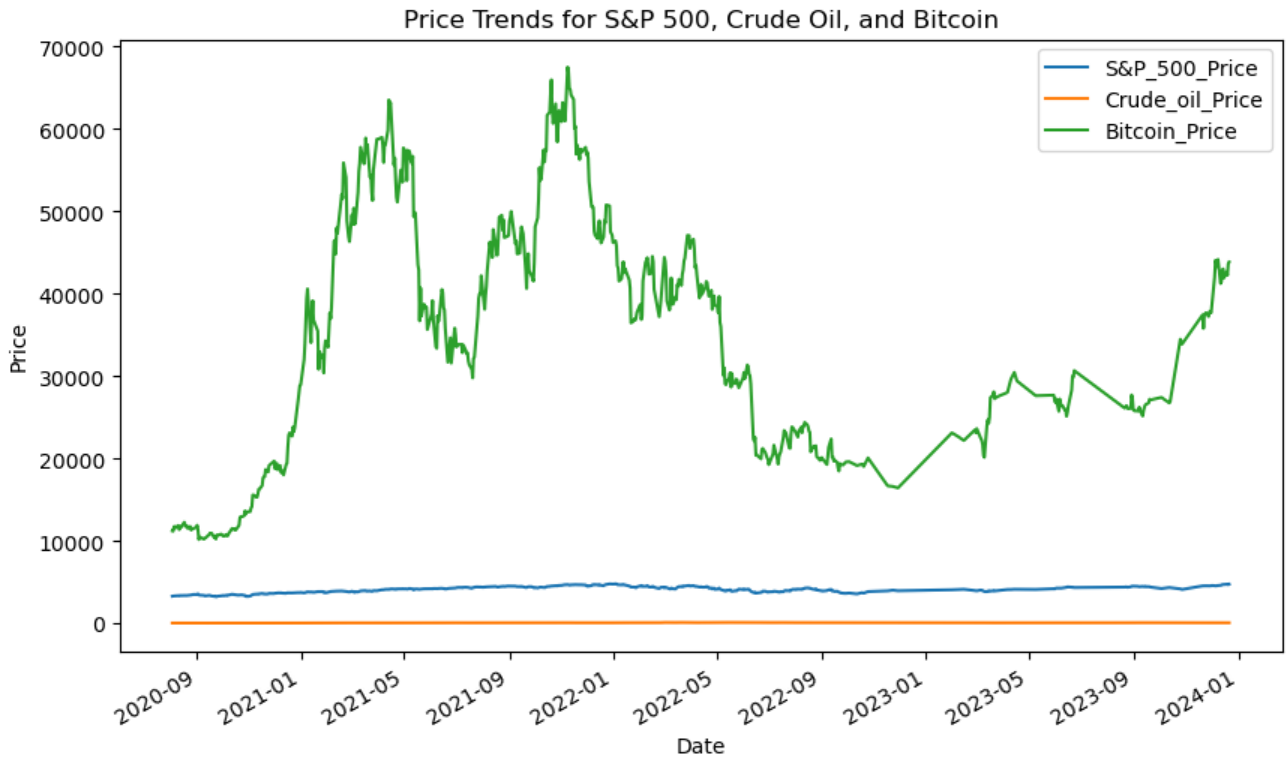
```
df['S&P_500_Price'] = df['S&P_500_Price'].replace({',': ''}, regex=True)
df['Bitcoin_Price'] = df['Bitcoin_Price'].replace({',': ''}, regex=True)
```

In [21]:

```
df['S&P_500_Price'] = pd.to_numeric(df['S&P_500_Price'], errors='coerce')
df['Bitcoin_Price'] = pd.to_numeric(df['Bitcoin_Price'], errors='coerce')
```

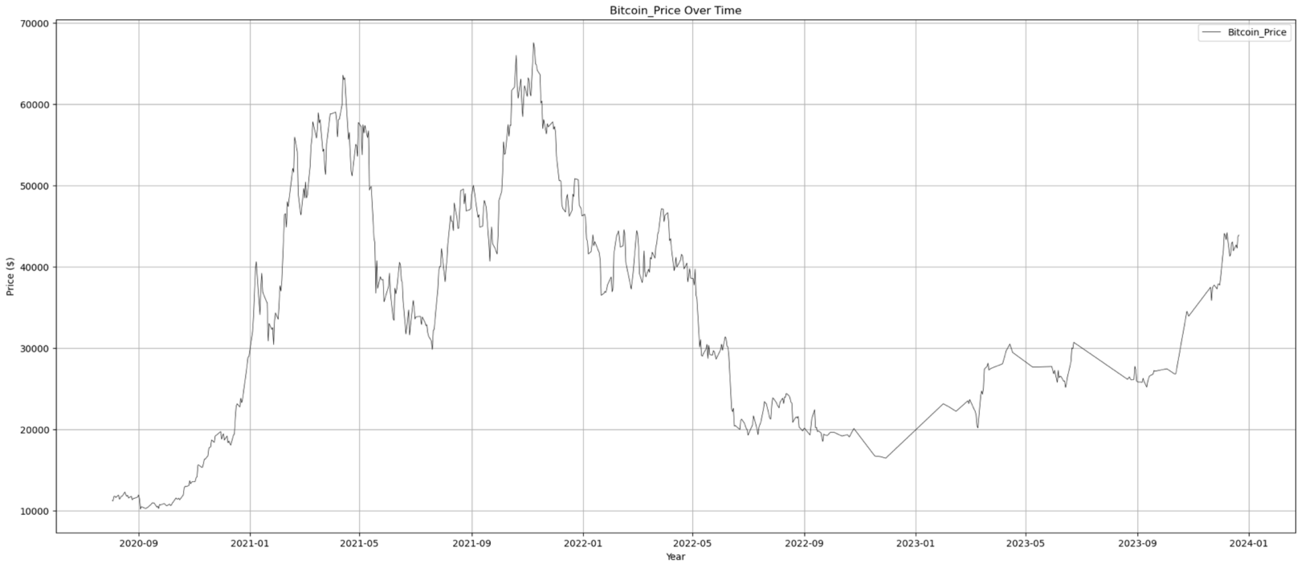
In [23]:

```
#Set Date Index
df.set_index('Date', inplace=True)

# Plot price trends for
df[['S&P_500_Price', 'Crude_oil_Price', 'Bitcoin_Price']].plot(figsize=(10,
plt.title('Price Trends for S&P 500, Crude Oil, and Bitcoin')
plt.xlabel('Date')
plt.ylabel('Price')
plt.legend(loc='best')
plt.show()
```

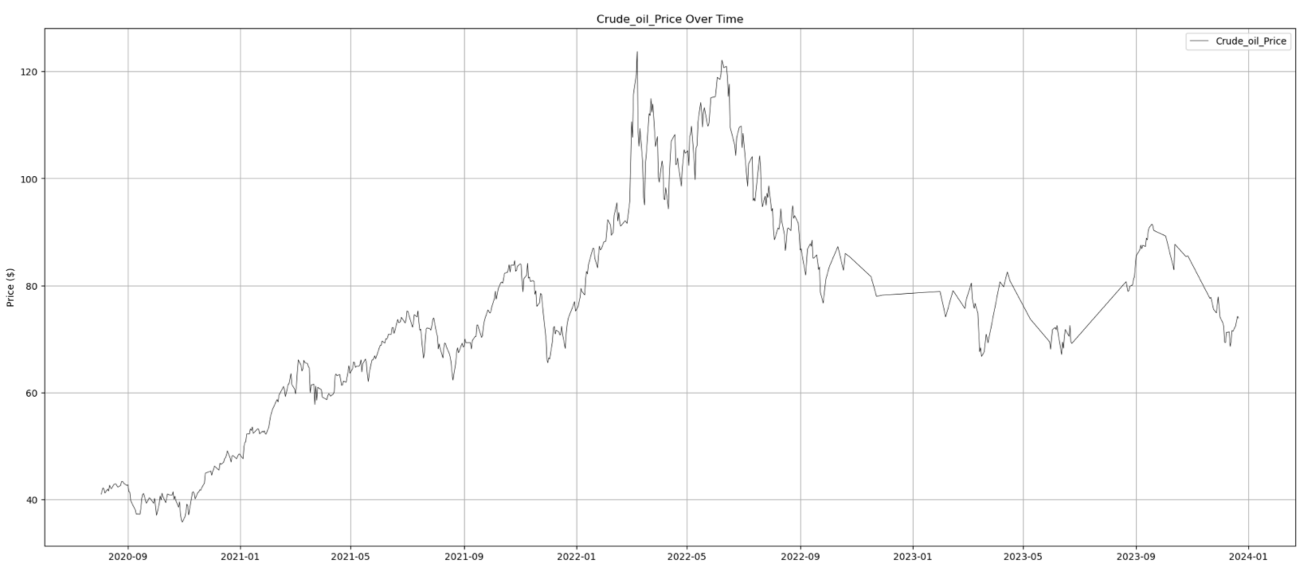


In [69]:

```python
# columns to visualize
columns = ['Bitcoin_Price']

# visualize
for col in columns:
    plt.figure(figsize=(24, 10))
    plt.plot(df.index, df[col], label=col, color='Black', linewidth=0.5)
    plt.title(f'{col} Over Time')
    plt.xlabel('Year')
    plt.ylabel('Price ($)')
    plt.legend()
    plt.grid()
    plt.show()
```

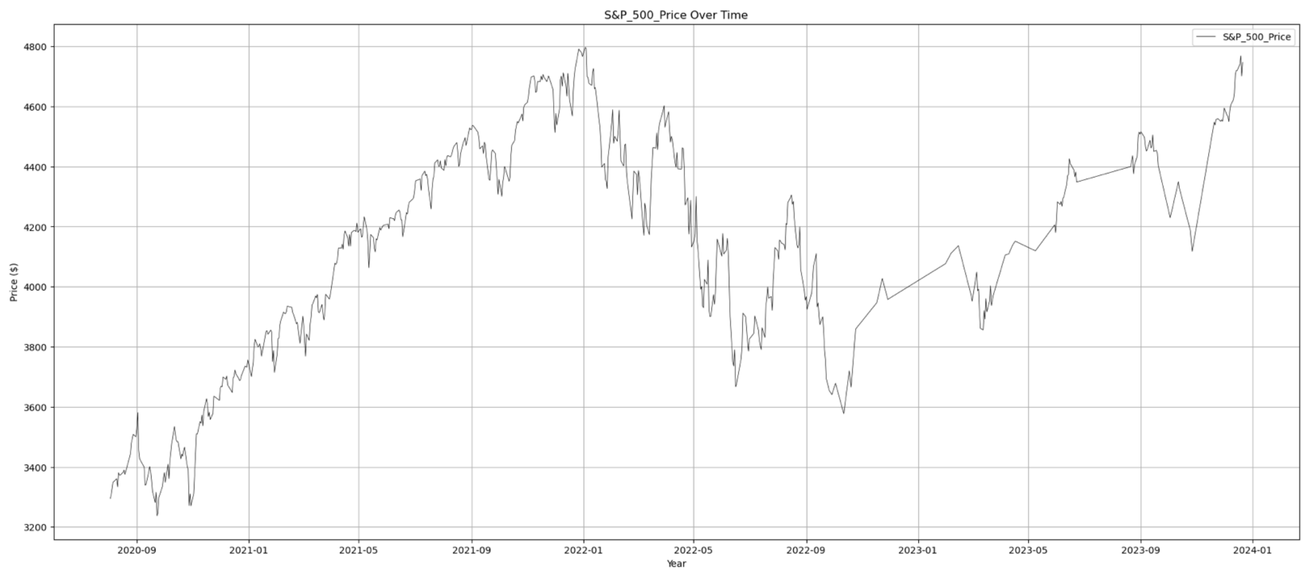


In [68]:
```
# columns to visualize
columns = ['Crude_oil_Price']

# visualize
for col in columns:
    plt.figure(figsize=(24, 10))
    plt.plot(df.index, df[col], label=col, color='Black', linewidth=0.5)
    plt.title(f'{col} Over Time')
    plt.xlabel('Year')
    plt.ylabel('Price ($)')
    plt.legend()
    plt.grid()
    plt.show()
```

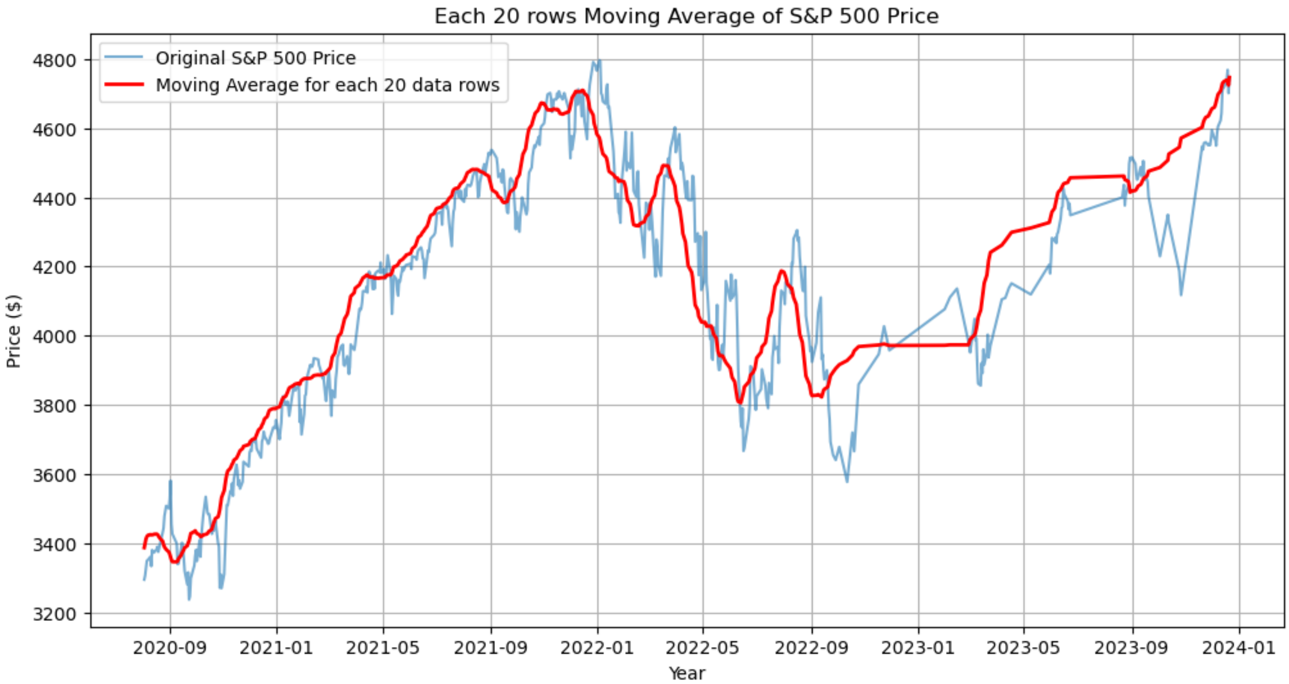


In [67]:
```
# columns to visualize
columns = ['S&P_500_Price']
```

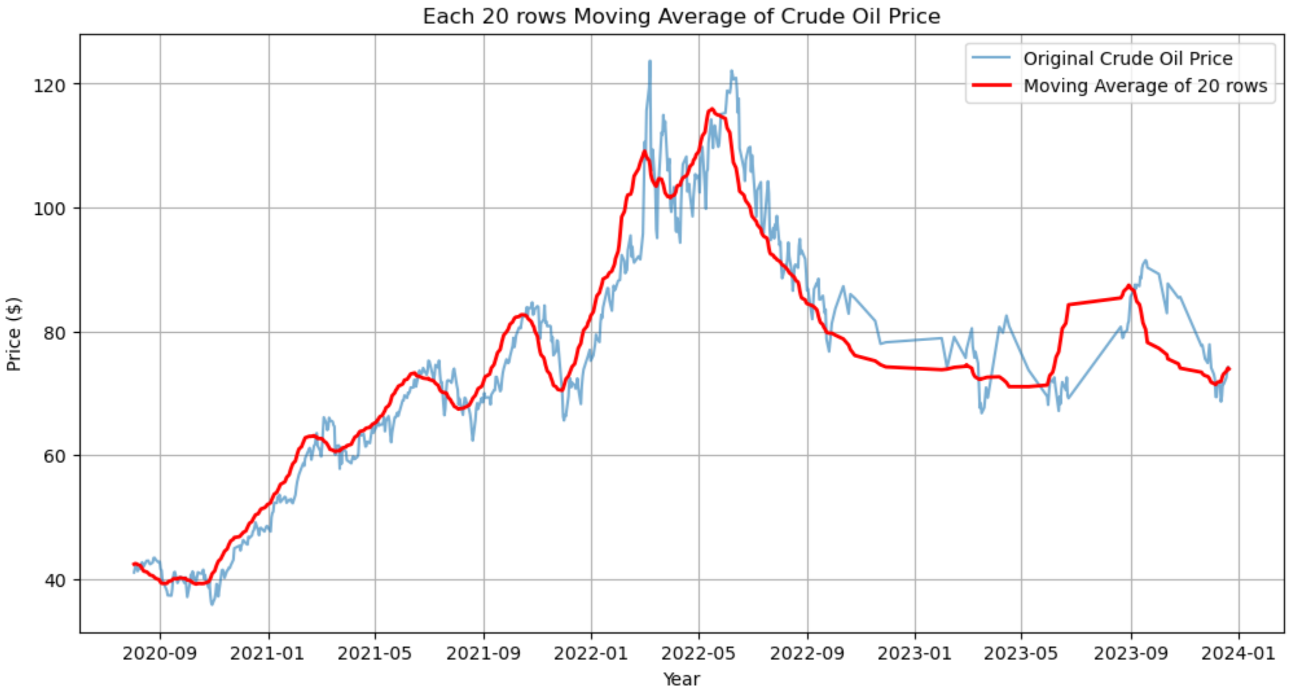
```
# visualize
for col in columns:
    plt.figure(figsize=(24, 10))
    plt.plot(df.index, df[col], label=col, color='Black', linewidth=0.5)
    plt.title(f'{col} Over Time')
    plt.xlabel('Year')
    plt.ylabel('Price ($)')
    plt.legend()
    plt.grid()
    plt.show()
```



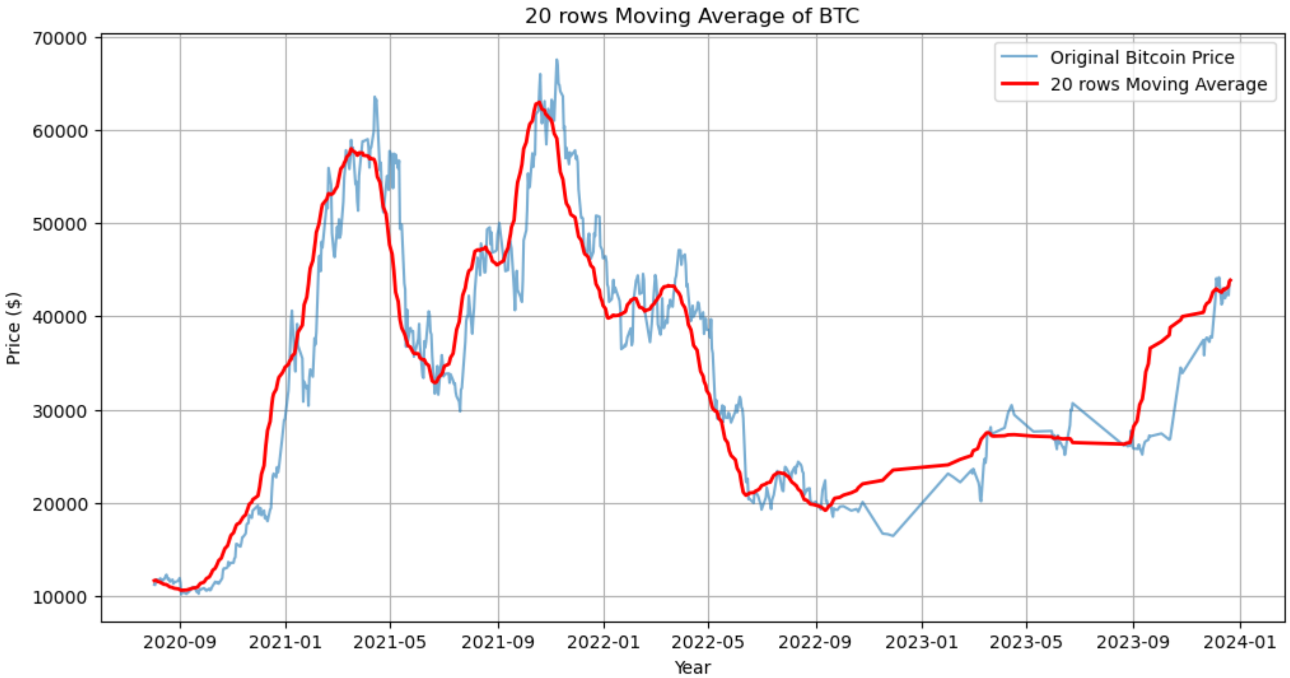
In [31]:
```
# Calculate the 10-year moving average for S&P 500 Price
df['S&P_500_Price_MovingAvg'] = df['S&P_500_Price'].rolling(window=20, min_p

# Plot S&P 500 Price and its moving average
plt.figure(figsize=(12, 6))
plt.plot(df.index, df['S&P_500_Price'], label='Original S&P 500 Price', alph
plt.plot(df.index, df['S&P_500_Price_MovingAvg'], label='Moving Average for 
plt.xlabel('Year')
plt.ylabel('Price ($)')
plt.title('Each 20 rows Moving Average of S&P 500 Price')
plt.legend()
plt.grid(True)
plt.show()
```



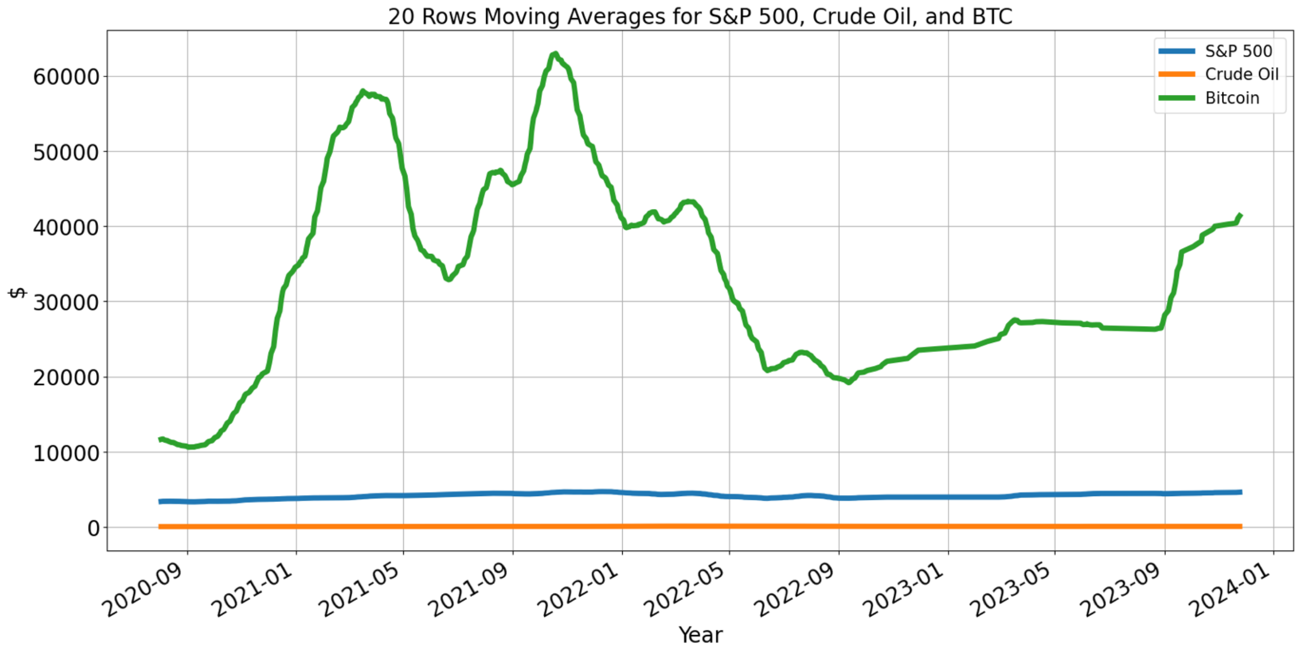
In [33]:
```
# Calculate the 10-year moving average for Crude Oil Price
df['Crude_oil_Price_MovingAvg'] = df['Crude_oil_Price'].rolling(window=20, m

# Plot Crude Oil Price and its moving average
plt.figure(figsize=(12, 6))
plt.plot(df.index, df['Crude_oil_Price'], label='Original Crude Oil Price',
plt.plot(df.index, df['Crude_oil_Price_MovingAvg'], label='Moving Average of
plt.xlabel('Year')
plt.ylabel('Price ($)')
plt.title('Each 20 rows Moving Average of Crude Oil Price')
plt.legend()
plt.grid(True)
plt.show()
```



In [34]:
```
# Calculate the 10-year moving average for Bitcoin Price
df['Bitcoin_Price_MovingAvg'] = df['Bitcoin_Price'].rolling(window=20, min_p

# Plot Bitcoin Price and its moving average
plt.figure(figsize=(12, 6))
plt.plot(df.index, df['Bitcoin_Price'], label='Original Bitcoin Price', alph
plt.plot(df.index, df['Bitcoin_Price_MovingAvg'], label='20 rows Moving Aver
plt.xlabel('Year')
plt.ylabel('Price ($)')
plt.title('20 rows Moving Average of BTC')
plt.legend()
plt.grid(True)
plt.show()
```

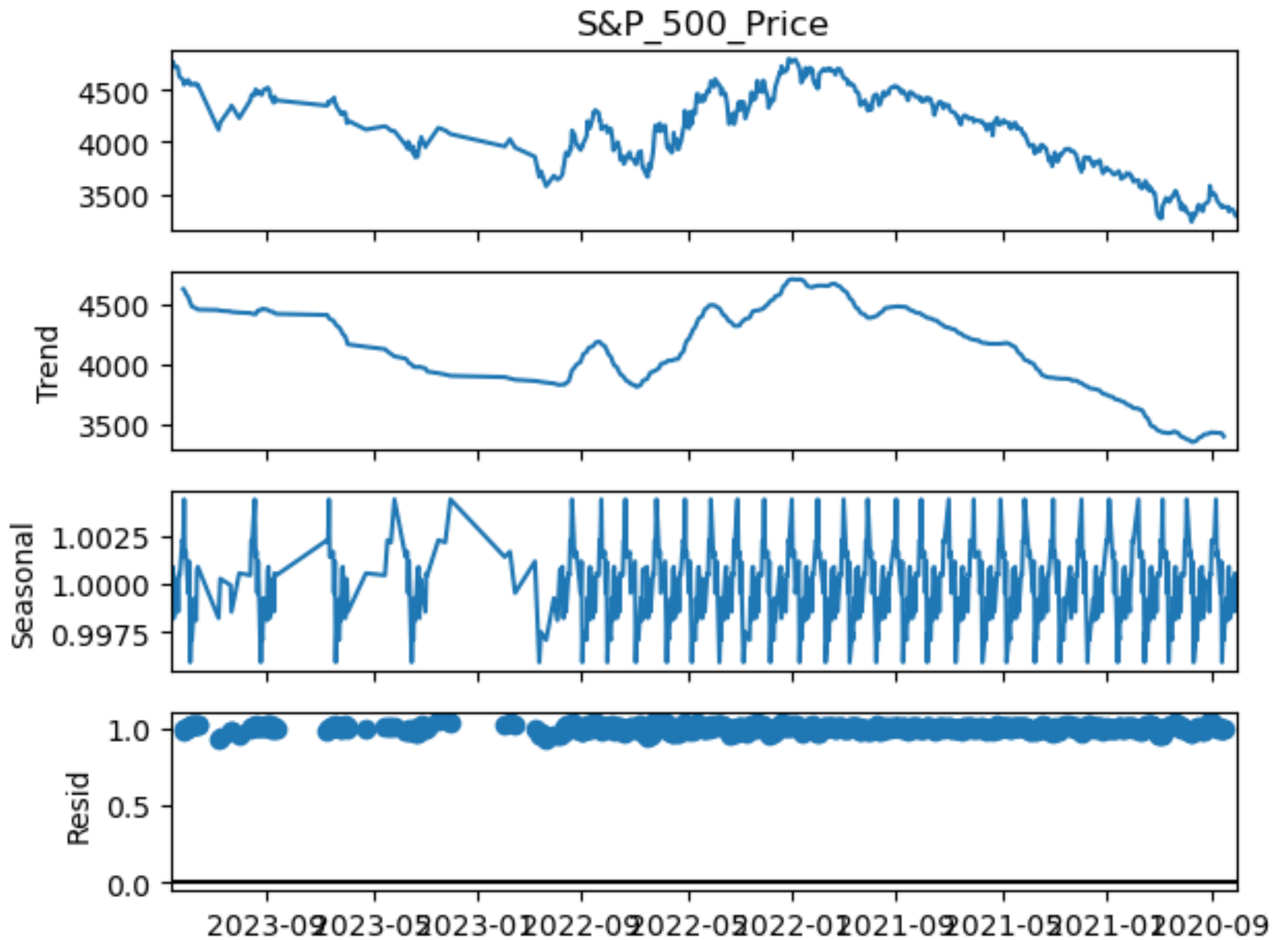


In [41]:
```
# 20 Rows rolling means
SP_MA = df['S&P_500_Price'].rolling(window=20).mean()
Crude_oil_MA = df['Crude_oil_Price'].rolling(window=20).mean()
Bitcoin_MA = df['Bitcoin_Price'].rolling(window=20).mean()


df_rm = pd.concat([SP_MA, Crude_oil_MA, Bitcoin_MA], axis=1)


df_rm.columns = ['S&P 500', 'Crude Oil', 'Bitcoin']


df_rm.plot(figsize=(20, 10), linewidth=5, fontsize=20)


plt.xlabel('Year', fontsize=20)
plt.ylabel('$', fontsize=20)
plt.title('20 Rows Moving Averages for S&P 500, Crude Oil, and BTC', fontsiz
plt.legend(loc='best', fontsize=15)
plt.grid(True)
plt.show()
```

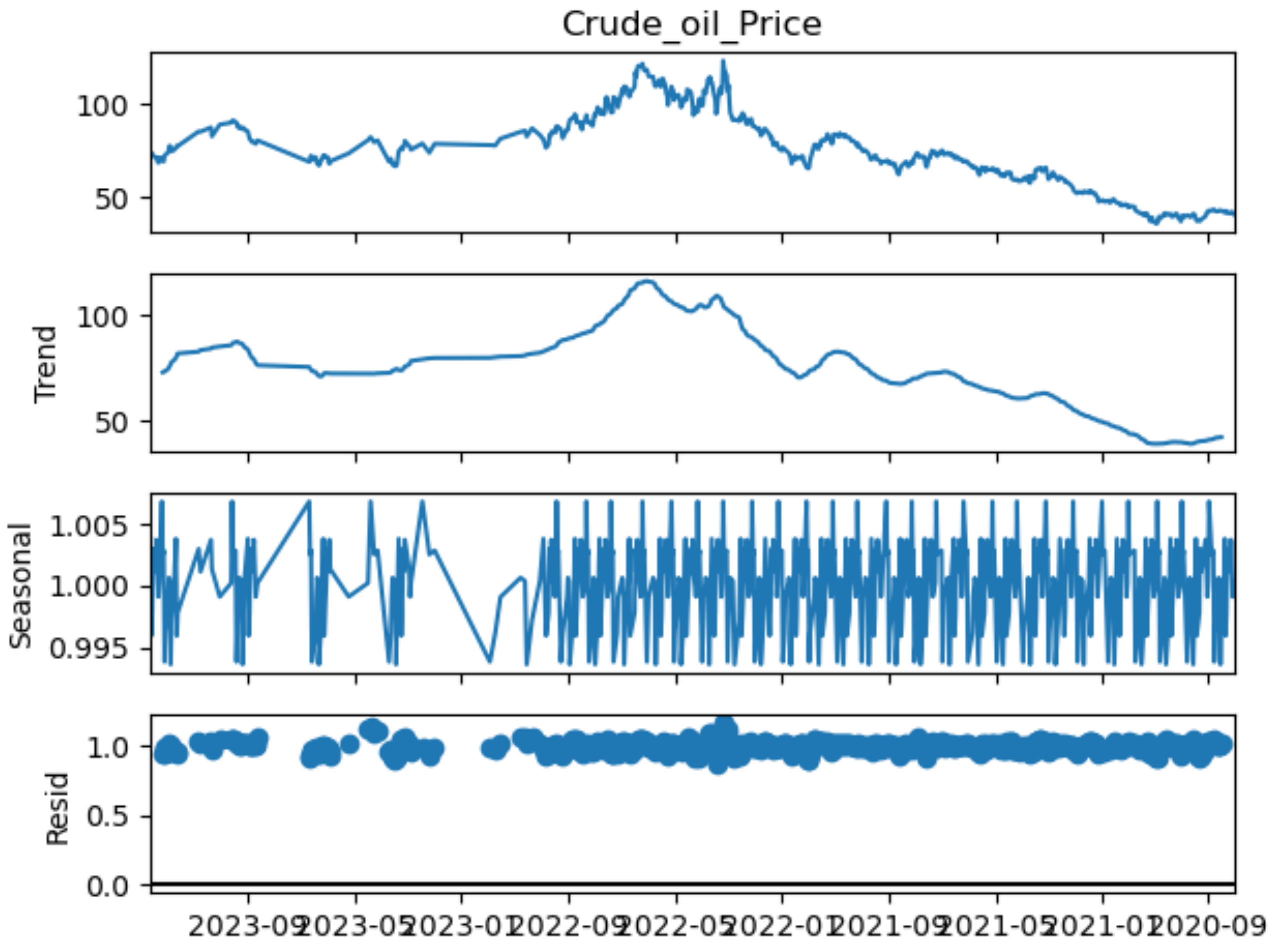


In [49]:
```
from statsmodels.tsa.seasonal import seasonal_decompose
```

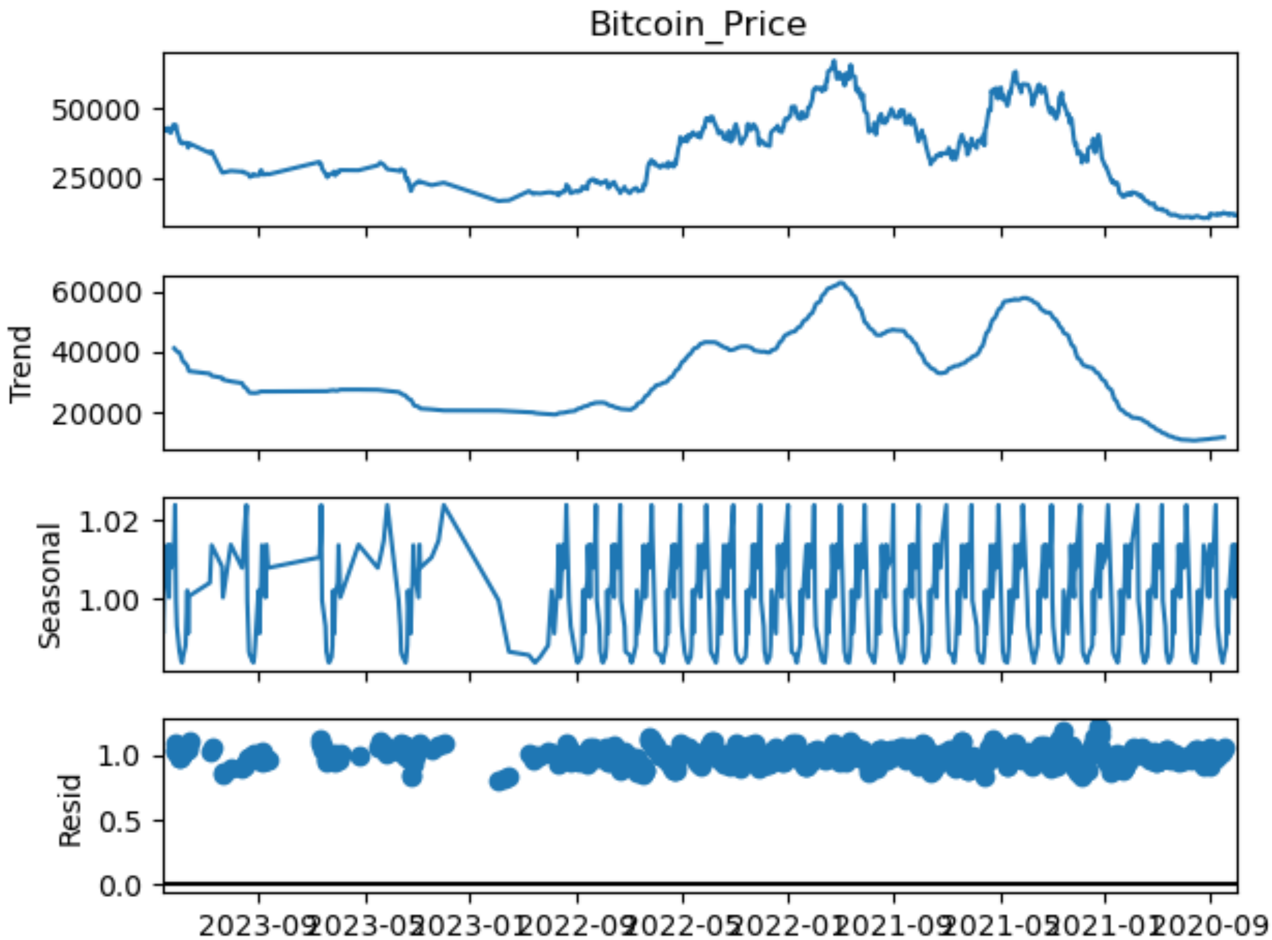
In [52]:
```
#Time series Decomposition and plot
decomposed = seasonal_decompose(df['S&P_500_Price'], model='multiplicative',

decomposed.plot()
plt.show()
```



In [53]:
```
#Time series Decomposition and plot
decomposed = seasonal_decompose(df['Crude_oil_Price'], model='multiplicative

decomposed.plot()
plt.show()
```

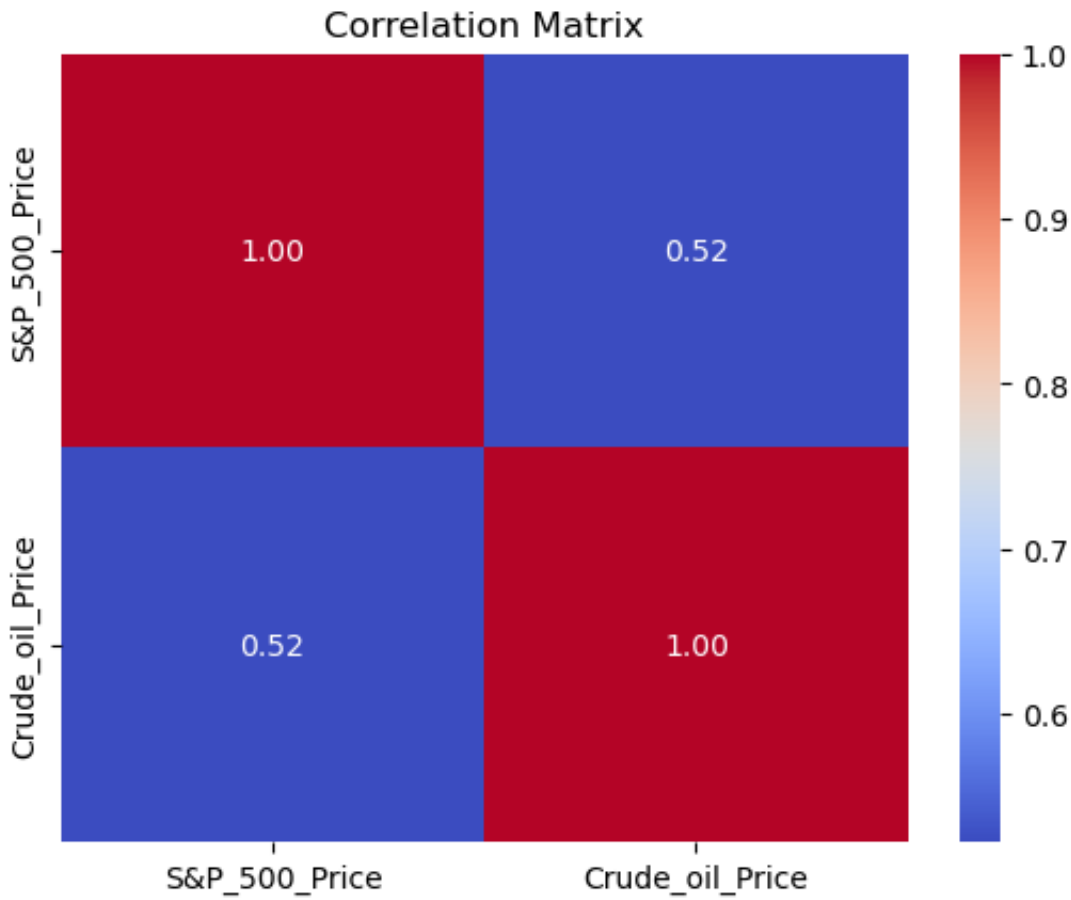


In [54]:
```
#Time series Decomposition and plot
decomposed = seasonal_decompose(df['Bitcoin_Price'], model='multiplicative',

decomposed.plot()
plt.show()
```



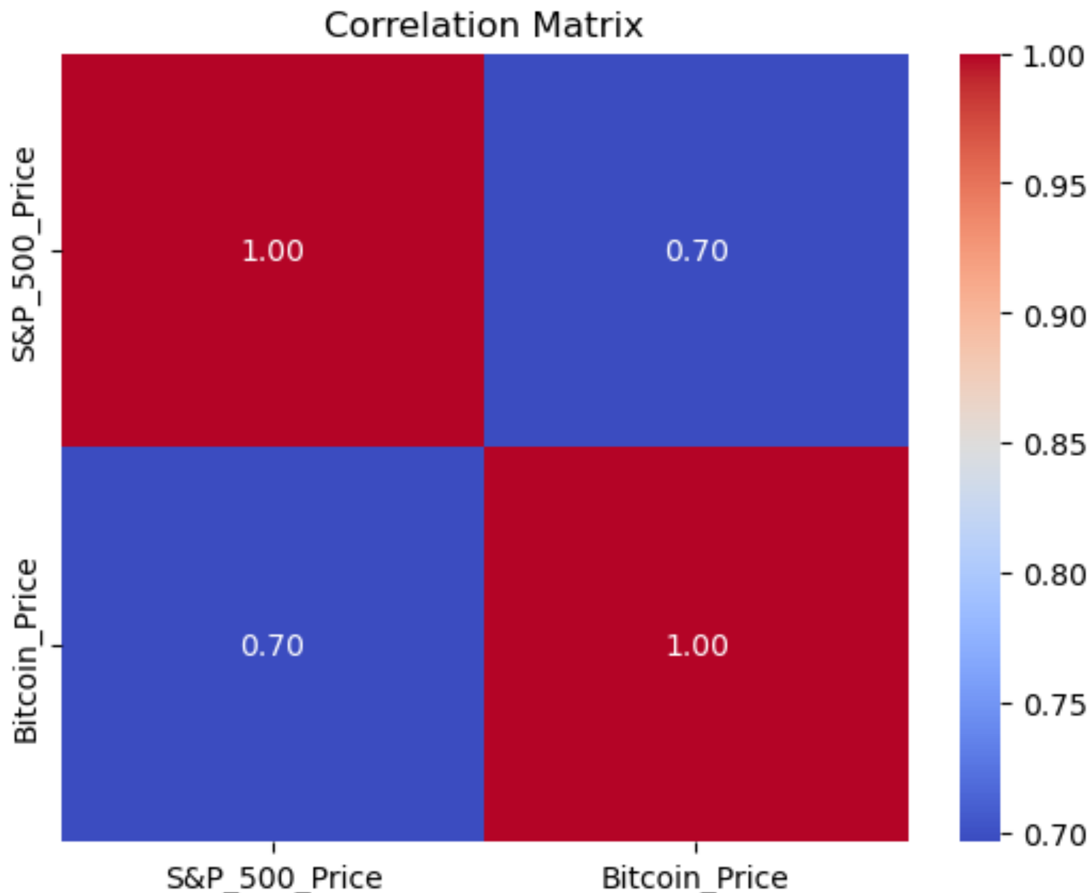
In [57]:
```
# Correlation matrix
correlation_matrix = df[['S&P_500_Price', 'Crude_oil_Price']].corr()
print(correlation_matrix)

sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix')
plt.show()
```

```
                 S&P_500_Price  Crude_oil_Price
S&P_500_Price         1.000000         0.522395
Crude_oil_Price       0.522395         1.000000
```



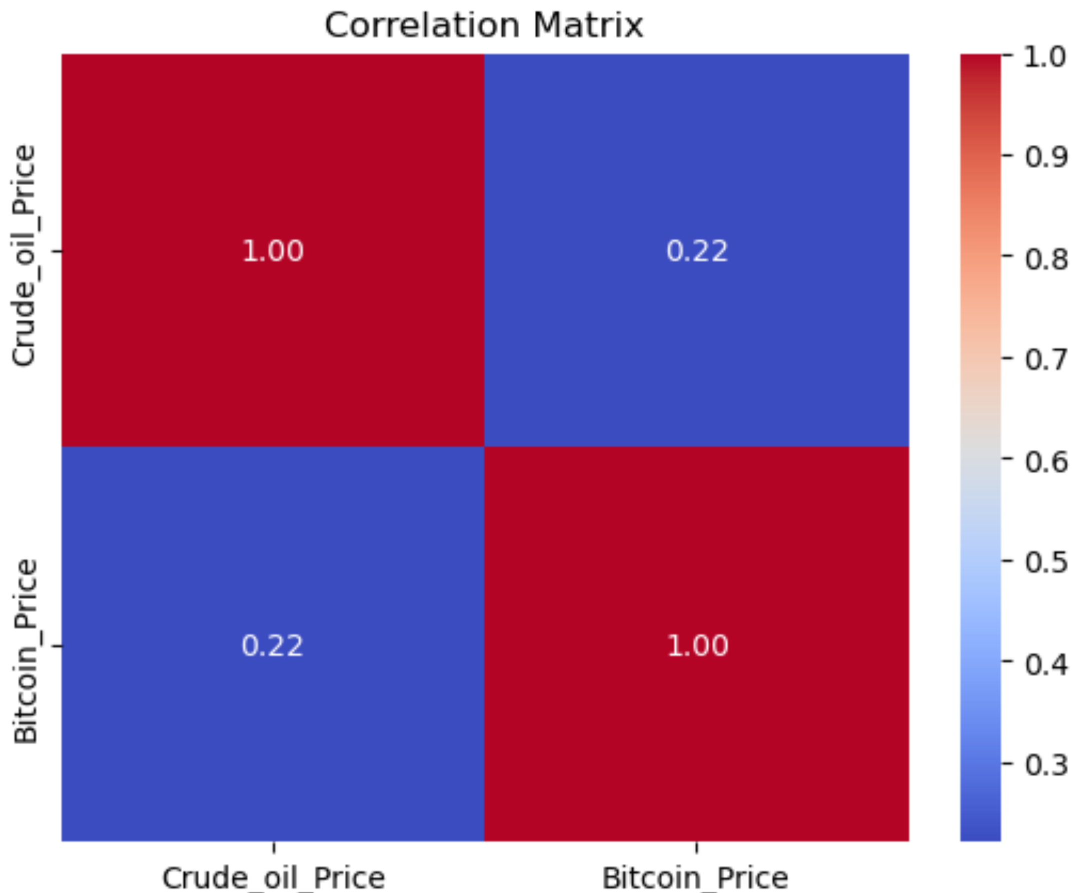
In [58]:
```
# Correlation matrix
correlation_matrix = df[['S&P_500_Price', 'Bitcoin_Price']].corr()
print(correlation_matrix)

sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix')
plt.show()
```

```
               S&P_500_Price  Bitcoin_Price
S&P_500_Price       1.000000       0.696271
Bitcoin_Price       0.696271       1.000000
```



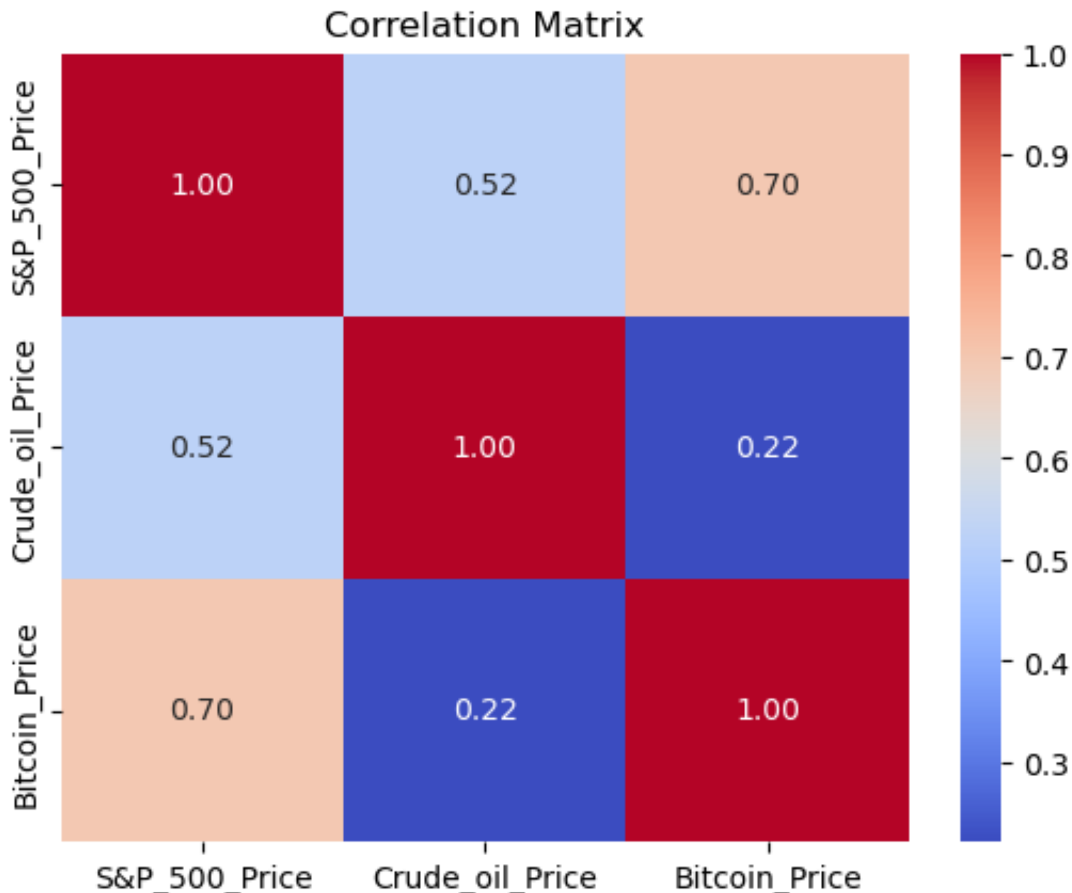
In [60]:
```
# Correlation matrix
correlation_matrix = df[['Crude_oil_Price', 'Bitcoin_Price']].corr()
print(correlation_matrix)

sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix')
plt.show()
```

```
                 Crude_oil_Price  Bitcoin_Price
Crude_oil_Price         1.000000       0.221003
Bitcoin_Price           0.221003       1.000000
```



In [56]:
```
# Correlation matrix
correlation_matrix = df[['S&P_500_Price', 'Crude_oil_Price', 'Bitcoin_Price'
print(correlation_matrix)

sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix')
plt.show()
```

```
                 S&P_500_Price  Crude_oil_Price  Bitcoin_Price
S&P_500_Price         1.000000         0.522395       0.696271
Crude_oil_Price       0.522395         1.000000       0.221003
Bitcoin_Price         0.696271         0.221003       1.000000
```

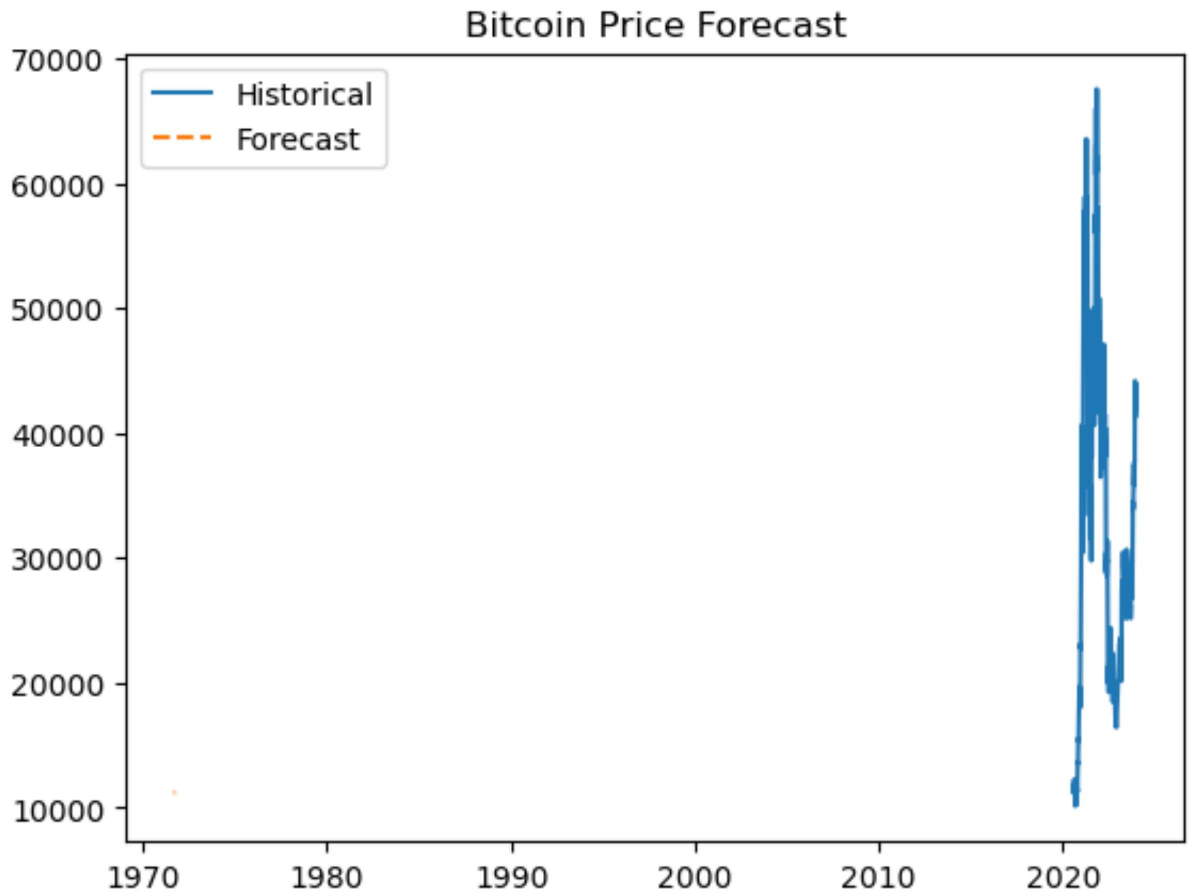


In [61]:
```
from statsmodels.tsa.arima.model import ARIMA
```

In [63]:
```
# ARIMA BTC
arima_model = ARIMA(df['Bitcoin_Price'], order=(1, 1, 1))
arima_result = arima_model.fit()

# Print ARIMA model summary and Forecast
print(arima_result.summary())


forecast = arima_result.forecast(steps=30)
plt.plot(df['Bitcoin_Price'], label='Historical')
plt.plot(forecast, label='Forecast', linestyle='--')
plt.legend()
plt.title('Bitcoin Price Forecast')
plt.show()
```

```
/opt/anaconda3/lib/python3.12/site-packages/statsmodels/tsa/base/tsa_model.p
y:473: ValueWarning: A date index has been provided, but it has no associate
d frequency information and so will be ignored when e.g. forecasting.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.12/site-packages/statsmodels/tsa/base/tsa_model.p
y:473: ValueWarning: A date index has been provided, but it is not monotonic
and so will be ignored when e.g. forecasting.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.12/site-packages/statsmodels/tsa/base/tsa_model.p
y:473: ValueWarning: A date index has been provided, but it has no associate
d frequency information and so will be ignored when e.g. forecasting.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.12/site-packages/statsmodels/tsa/base/tsa_model.p
y:473: ValueWarning: A date index has been provided, but it is not monotonic
and so will be ignored when e.g. forecasting.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.12/site-packages/statsmodels/tsa/base/tsa_model.p
y:473: ValueWarning: A date index has been provided, but it has no associate
d frequency information and so will be ignored when e.g. forecasting.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.12/site-packages/statsmodels/tsa/base/tsa_model.p
y:473: ValueWarning: A date index has been provided, but it is not monotonic
and so will be ignored when e.g. forecasting.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.12/site-packages/statsmodels/tsa/base/tsa_model.p
y:836: ValueWarning: No supported index is available. Prediction results wil
l be given with an integer index beginning at `start`.
  return get_prediction_index(
/opt/anaconda3/lib/python3.12/site-packages/statsmodels/tsa/base/tsa_model.p
y:836: FutureWarning: No supported index is available. In the next version,
calling this method in a model without a supported index will result in an e
xception.
  return get_prediction_index(
```

```
                               SARIMAX Results
==============================================================================
==
Dep. Variable:          Bitcoin_Price   No. Observations:                    6
09
Model:                 ARIMA(1, 1, 1)   Log Likelihood                 -5406.6
70
Date:                Sat, 21 Dec 2024   AIC                            10819.3
41
Time:                        14:58:00   BIC                            10832.5
71
Sample:                             0   HQIC                           10824.4
88
                                - 609
Covariance Type:                  opg
==============================================================================
==
                 coef    std err          z      P>|z|      [0.025      0.97
5]
------------------------------------------------------------------------------
--
ar.L1         -0.3648      6.512     -0.056      0.955     -13.128      12.3
99
ma.L1          0.3611      6.534      0.055      0.956     -12.446      13.1
68
sigma2      3.117e+06   1.13e+05     27.646      0.000     2.9e+06    3.34e+
06
==============================================================================
=======
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):
253.91
Prob(Q):                              0.99   Prob(JB):
0.00
Heteroskedasticity (H):               1.82   Skew:
-0.01
Prob(H) (two-sided):                  0.00   Kurtosis:
6.17
==============================================================================
=======

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (compl
ex-step).
```

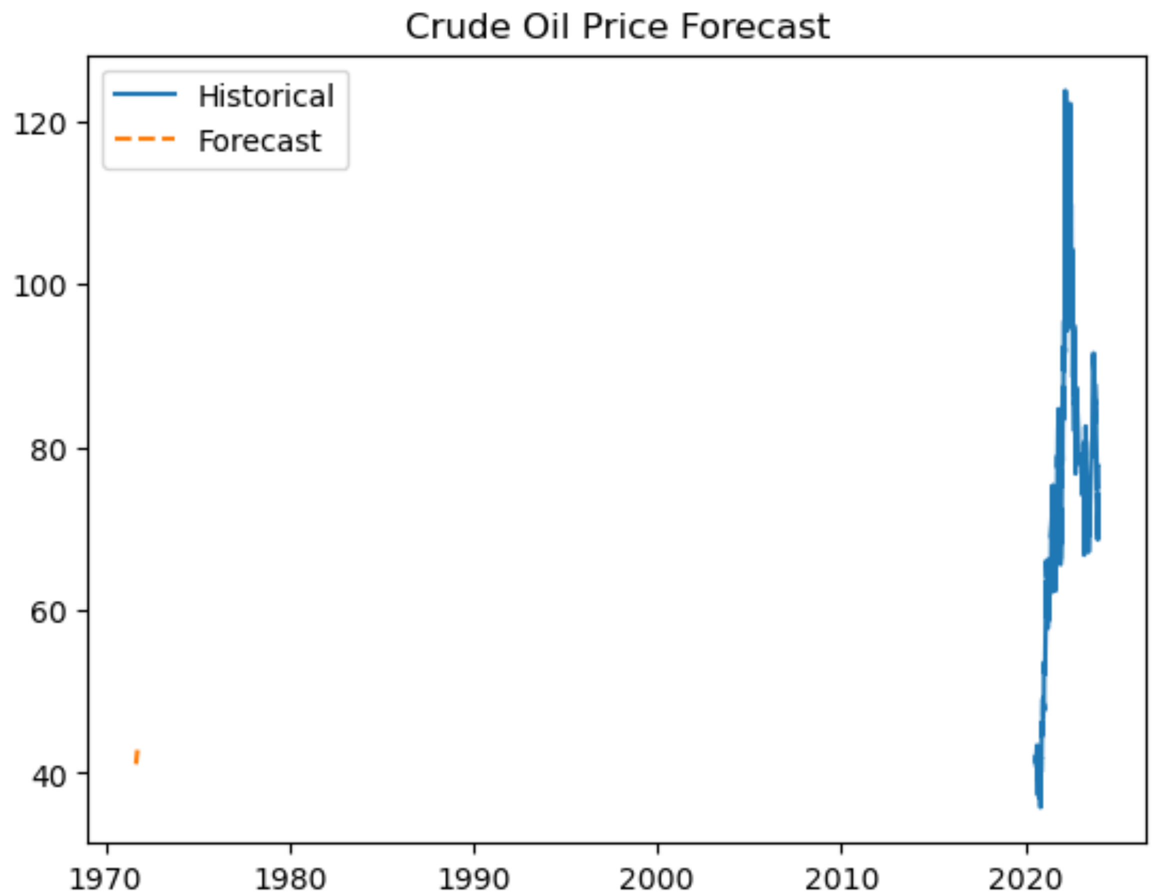


In [64]:
```
# Define Crude Oil
arima_model = ARIMA(df['Crude_oil_Price'], order=(1, 1, 1))
arima_result = arima_model.fit()

# Print ARIMA model summary and Forecast
print(arima_result.summary())

# Forecast future values
forecast = arima_result.forecast(steps=30)
plt.plot(df['Crude_oil_Price'], label='Historical')
plt.plot(forecast, label='Forecast', linestyle='--')
plt.legend()
plt.title('Crude Oil Price Forecast')
plt.show()
```

```
/opt/anaconda3/lib/python3.12/site-packages/statsmodels/tsa/base/tsa_model.p
y:473: ValueWarning: A date index has been provided, but it has no associate
d frequency information and so will be ignored when e.g. forecasting.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.12/site-packages/statsmodels/tsa/base/tsa_model.p
y:473: ValueWarning: A date index has been provided, but it is not monotonic
and so will be ignored when e.g. forecasting.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.12/site-packages/statsmodels/tsa/base/tsa_model.p
y:473: ValueWarning: A date index has been provided, but it has no associate
d frequency information and so will be ignored when e.g. forecasting.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.12/site-packages/statsmodels/tsa/base/tsa_model.p
y:473: ValueWarning: A date index has been provided, but it is not monotonic
and so will be ignored when e.g. forecasting.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.12/site-packages/statsmodels/tsa/base/tsa_model.p
y:473: ValueWarning: A date index has been provided, but it has no associate
d frequency information and so will be ignored when e.g. forecasting.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.12/site-packages/statsmodels/tsa/base/tsa_model.p
y:473: ValueWarning: A date index has been provided, but it is not monotonic
and so will be ignored when e.g. forecasting.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.12/site-packages/statsmodels/tsa/statespace/sarim
ax.py:978: UserWarning: Non-invertible starting MA parameters found. Using z
eros as starting parameters.
  warn('Non-invertible starting MA parameters found.'
```

```
                               SARIMAX Results
==============================================================================
==
Dep. Variable:        Crude_oil_Price   No. Observations:                    6
09
Model:                 ARIMA(1, 1, 1)   Log Likelihood               -1387.9
94
Date:                Sat, 21 Dec 2024   AIC                           2781.9
88
Time:                        15:00:25   BIC                           2795.2
19
Sample:                             0   HQIC                          2787.1
36
                                - 609
Covariance Type:                  opg
==============================================================================
==
                 coef    std err          z      P>|z|      [0.025      0.97
5]
------------------------------------------------------------------------------
--
ar.L1          0.9938      0.047     20.952      0.000       0.901       1.0
87
ma.L1         -0.9965      0.044    -22.538      0.000      -1.083      -0.9
10
sigma2         5.6280      0.163     34.551      0.000       5.309       5.9
47
==============================================================================
=======
Ljung-Box (L1) (Q):                   0.36   Jarque-Bera (JB):
955.47
Prob(Q):                              0.55   Prob(JB):
0.00
Heteroskedasticity (H):               0.16   Skew:
0.54
Prob(H) (two-sided):                  0.00   Kurtosis:
9.04
==============================================================================
=======

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (compl
ex-step).
```

```
/opt/anaconda3/lib/python3.12/site-packages/statsmodels/tsa/base/tsa_model.p
y:836: ValueWarning: No supported index is available. Prediction results wil
l be given with an integer index beginning at `start`.
  return get_prediction_index(
/opt/anaconda3/lib/python3.12/site-packages/statsmodels/tsa/base/tsa_model.p
y:836: FutureWarning: No supported index is available. In the next version,
calling this method in a model without a supported index will result in an e
xception.
  return get_prediction_index(
```




```
In [ ]:
```