Time Series Analysis

Task: Choose a dataset with a time component and perform time series analysis

- Embark on a time series analysis project using a dataset with a time component, specifically historical stock prices. The objective is to uncover patterns, trends, and insights from the temporal data, enabling a better understanding of stock price movements over time

Time series analysis is a statistical method that involves studying and analyzing data points collected over time to identify patterns, trends, and make predictions about future values.

Importing necessary libraries

In [1]:
```
import pandas as pd
import numpy as np
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
%matplotlib inline
```

In [2]:
```
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.seasonal import seasonal_decompose as sd
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
```

Loading and Exploring the dataset

In [3]:
```
df = pd.read_csv('Microsoft_Stock.csv')
df.head()
```

Out[3]:

	Date	Open	High	Low	Close	Volume
0	4/1/2015 16:00:00	40.60	40.76	40.31	40.72	36865322
1	4/2/2015 16:00:00	40.66	40.74	40.12	40.29	37487476
2	4/6/2015 16:00:00	40.34	41.78	40.18	41.55	39223692
3	4/7/2015 16:00:00	41.61	41.91	41.31	41.53	28809375
4	4/8/2015 16:00:00	41.48	41.69	41.04	41.42	24753438

In [4]:
```
df['Date'] = pd.to_datetime(df['Date'])
df.set_index('Date', inplace=True)
```

In [5]:
```
df.sample(5)
```

Out[5]:

Date	Open	High	Low	Close	Volume
2019-06-12 16:00:00	131.40	131.97	130.71	131.49	17092464
2018-01-05 16:00:00	87.66	88.41	87.43	88.19	23407110
2018-11-01 16:00:00	107.05	107.32	105.53	105.92	33384201
2020-03-03 16:00:00	173.80	175.00	162.26	164.51	71677019
2020-06-12 16:00:00	190.54	191.72	185.18	187.74	43373587

In [6]:
```
df.columns
```

Out[6]:
```
Index(['Open', 'High', 'Low', 'Close', 'Volume'], dtype='object')
```

In [7]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 1511 entries, 2015-04-01 16:00:00 to 2021-03-31 16:00:00
Data columns (total 5 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Open    1511 non-null   float64
 1   High    1511 non-null   float64
 2   Low     1511 non-null   float64
 3   Close   1511 non-null   float64
 4   Volume  1511 non-null   int64  
dtypes: float64(4), int64(1)
memory usage: 70.8 KB
```

In [8]:
```
df.describe()
```

Out[8]:

	Open	High	Low	Close	Volume
count	1511.000000	1511.000000	1511.000000	1511.000000	1.511000e+03
mean	107.385976	108.437472	106.294533	107.422091	3.019863e+07
std	56.691333	57.382276	55.977155	56.702299	1.425266e+07
min	40.340000	40.740000	39.720000	40.290000	1.016120e+05
25%	57.860000	58.060000	57.420000	57.855000	2.136213e+07
50%	93.990000	95.100000	92.920000	93.860000	2.662962e+07
75%	139.440000	140.325000	137.825000	138.965000	3.431962e+07
max	245.030000	246.130000	242.920000	244.990000	1.352271e+08

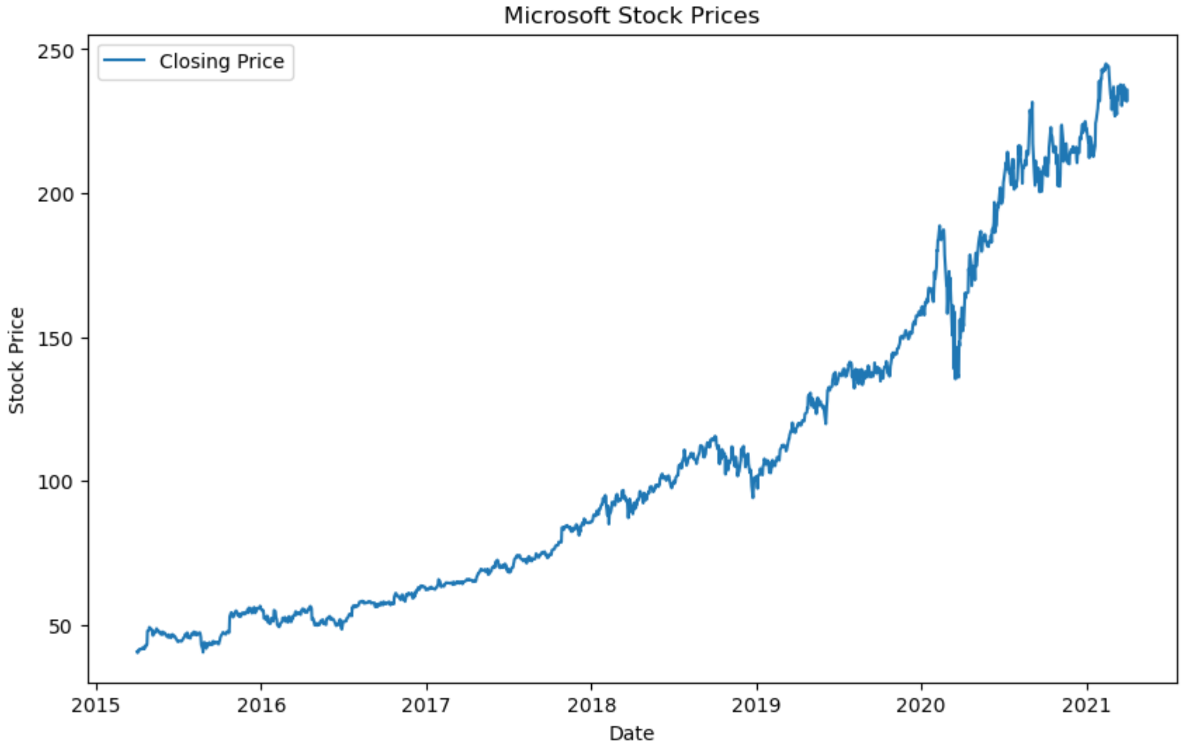
In [9]:
```
df.isnull().sum()
```

Out[9]:
```
Open      0
High      0
Low       0
Close     0
Volume    0
dtype: int64
```

Visualization of Time Series

(Exploratory Data Analysis)

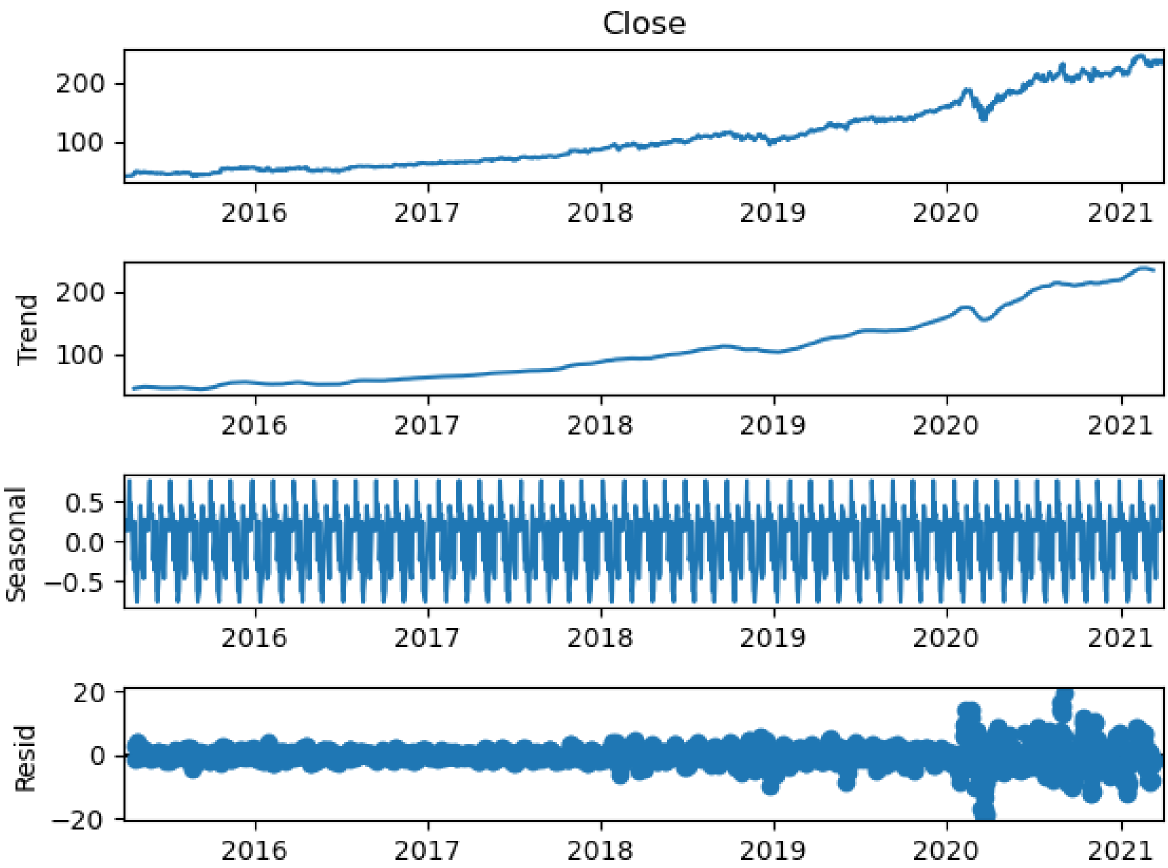
In [10]:
```
plt.figure(figsize=(10, 6))
plt.plot(df['Close'], label='Closing Price')
plt.title('Microsoft Stock Prices')
plt.xlabel('Date')
plt.ylabel('Stock Price')
plt.legend()
plt.show()
```



Time Series Decomposition

Decompose the time series into its components: trend, seasonality, and residuals. This helps in understanding the underlying patterns.

In [11]:
```
result = sd(df['Close'], model='additive', period=30)
result.plot()
plt.show()
```



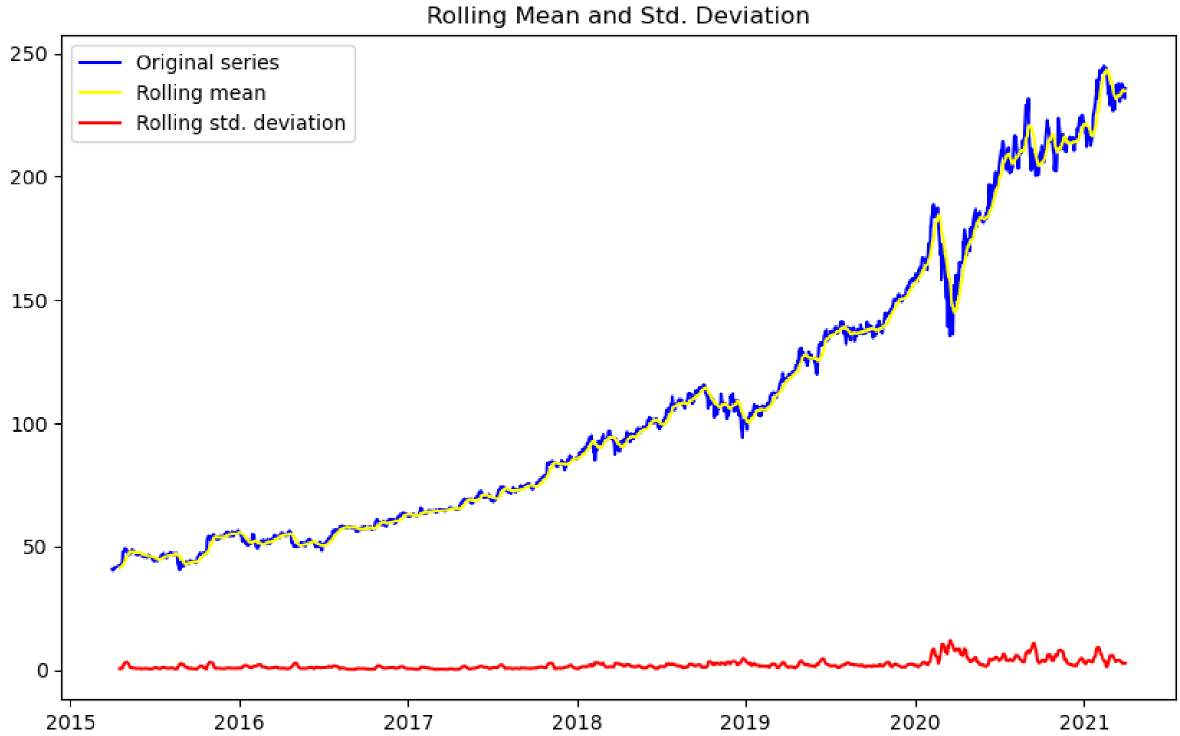
Statistical Analysis

- Conduct statistical tests for stationarity, such as the Augmented Dickey-Fuller (ADF) test.
- Check autocorrelation and partial autocorrelation functions to identify lag values for potential autoregressive (AR) and moving average (MA) terms in later modeling.

In [12]:
```
adf_result = adfuller(df['Close'])
print('ADF Statistic:', adf_result[0])
print('p-value:', adf_result[1])
```

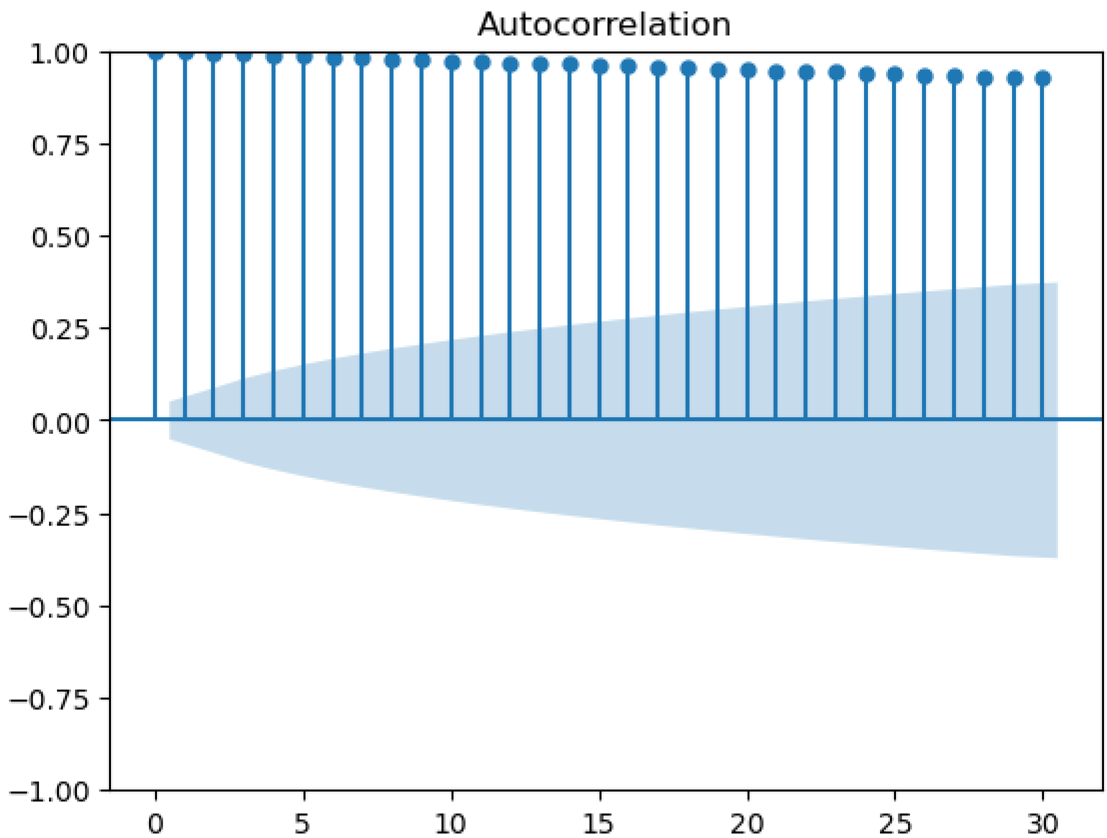
```
ADF Statistic: 1.7371362899270992
p-value: 0.9982158366942122
```

In [13]:
```
adf_mean = df['Close'].rolling(12).mean()
adf_std = df['Close'].rolling(12).std()

plt.figure(figsize=(10,6))
plt.plot(df['Close'], color='blue', label='Original series')
plt.plot(adf_mean, color='yellow', label='Rolling mean')
plt.plot(adf_std, color='red', label='Rolling std. deviation')

plt.legend(loc='best')
plt.title('Rolling Mean and Std. Deviation')
plt.show()
```



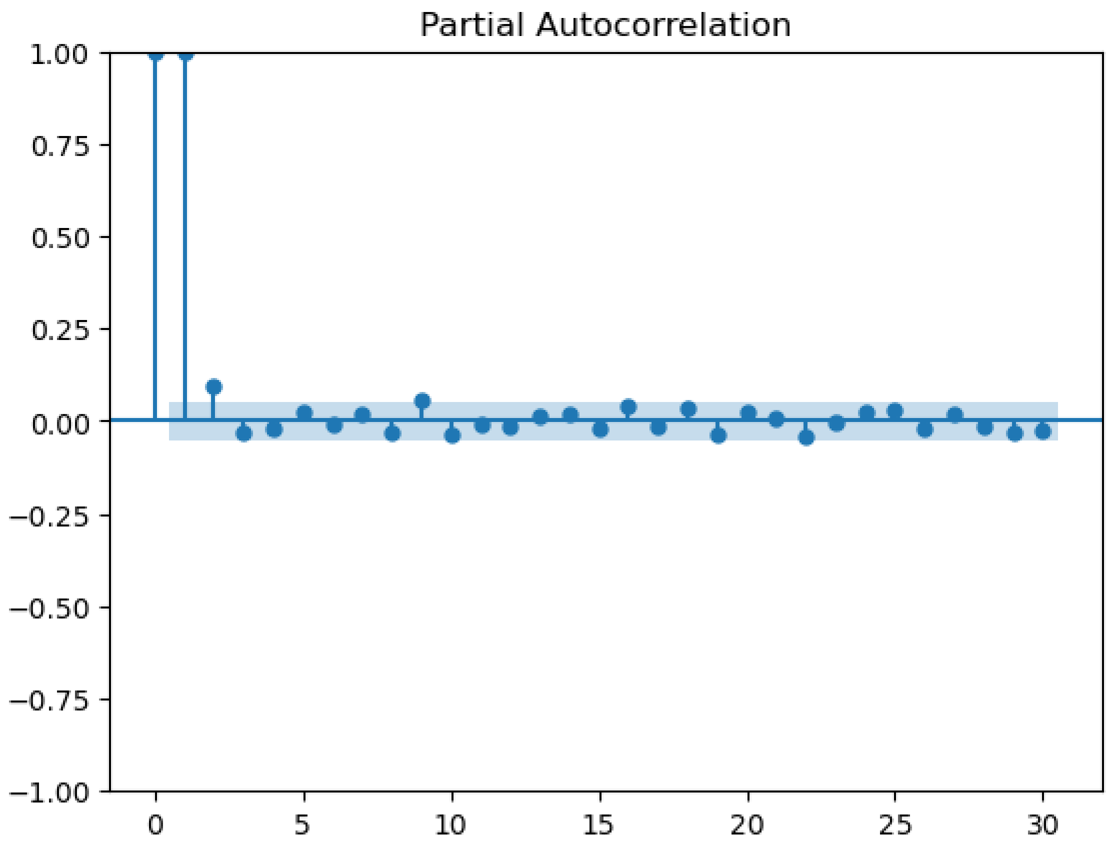
Plotting autocorrelation

In [14]:
```
plot_acf(df['Close'], lags=30)
plt.show()
```



Plotting Partial autocorrelation functions

In [15]:
```
plot_pacf(df['Close'], lags=30, method='ywm')
plt.show()
```



Resampling the data

(On a daily, weekly, and monthly basis and calculate the mean stock price)

In [16]:
```
daily_prices = df['Close'].resample('D').mean()
print('Daily Variation in Stock Prices: ')
daily_prices.head()
```

```
Daily Variation in Stock Prices:
```

Out[16]:
```
Date
2015-04-01    40.72
2015-04-02    40.29
2015-04-03      NaN
2015-04-04      NaN
2015-04-05      NaN
Freq: D, Name: Close, dtype: float64
```

In [17]:
```
weekly_prices = df['Close'].resample('W').mean()
print('Weekly Variation in Stock Prices: ')
weekly_prices.head()
```

```
Weekly Variation in Stock Prices:
```

Out[17]:
```
Date
2015-04-05    40.505
2015-04-12    41.540
2015-04-19    41.890
2015-04-26    43.950
2015-05-03    48.710
Freq: W-SUN, Name: Close, dtype: float64
```

In [18]:
```
monthly_prices = df['Close'].resample('M').mean()
print('Monthly Variation in Stock Prices: ')
monthly_prices.head()
```

```
Monthly Variation in Stock Prices:
```

Out[18]:
```
Date
2015-04-30    43.466667
2015-05-31    47.530000
2015-06-30    45.964091
2015-07-31    45.611818
2015-08-31    45.506667
Freq: M, Name: Close, dtype: float64
```

Model Selection and Training (ARIMA)

- ARIMA stands for "AutoRegressive Integrated Moving Average"
- It combines three key components: AutoRegressive (AR), Integrated (I), and Moving Average (MA).
- ARIMA models are widely used for analyzing and forecasting time series data.

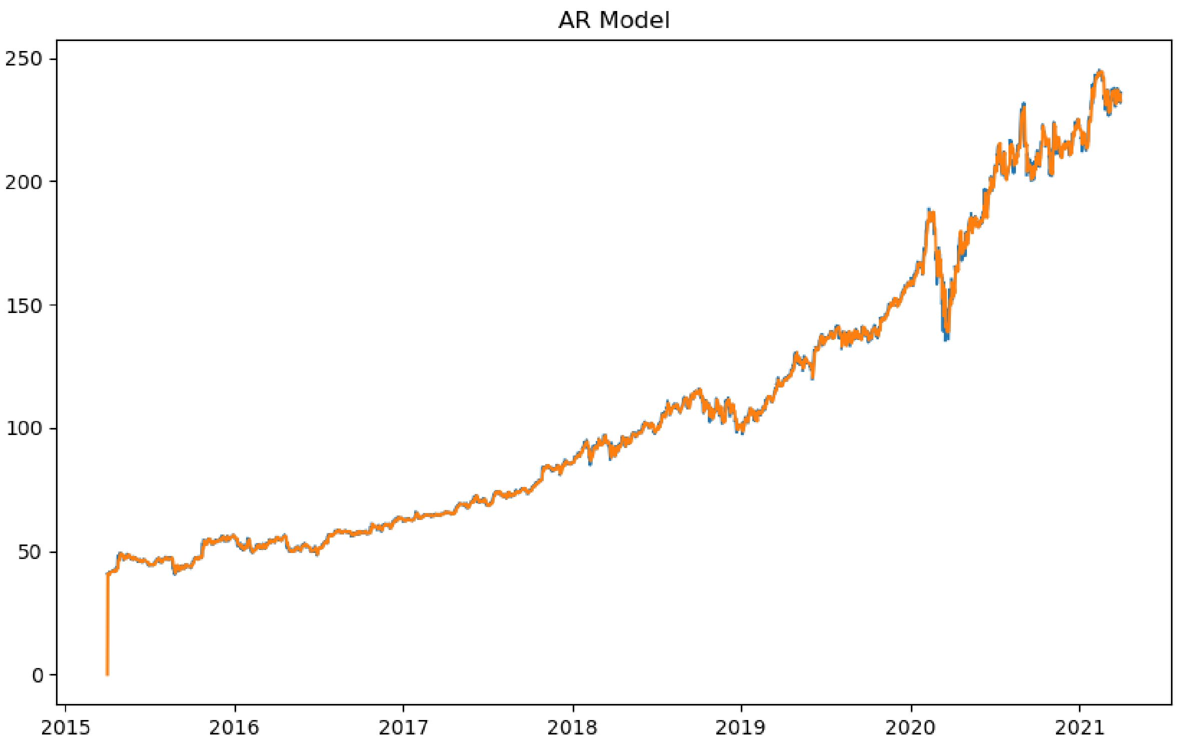
In [19]:
```
df = df.reindex(pd.date_range(start=df.index.min(), end=df.index.max(), freq='D'))
```

In [20]:
```
model = ARIMA(df['Close'], order=(5, 1, 5))
fit_model = model.fit()
```

```
C:\Users\suman\anaconda3\lib\site-packages\statsmodels\tsa\statespace\sarimax.py:9
66: UserWarning: Non-stationary starting autoregressive parameters found. Using ze
ros as starting parameters.
  warn('Non-stationary starting autoregressive parameters'
C:\Users\suman\anaconda3\lib\site-packages\statsmodels\tsa\statespace\sarimax.py:9
78: UserWarning: Non-invertible starting MA parameters found. Using zeros as start
ing parameters.
  warn('Non-invertible starting MA parameters found.'
C:\Users\suman\anaconda3\lib\site-packages\statsmodels\base\model.py:604: Converge
nceWarning: Maximum Likelihood optimization failed to converge. Check mle_retvals
  warnings.warn("Maximum Likelihood optimization failed to "
```

In [21]:
```
# Visual Representation of AR model
plt.figure(figsize=(10,6))
plt.plot(df['Close'])
plt.plot(fit_model.fittedvalues)
plt.title('AR Model')
```

Out[21]:
```
Text(0.5, 1.0, 'AR Model')
```

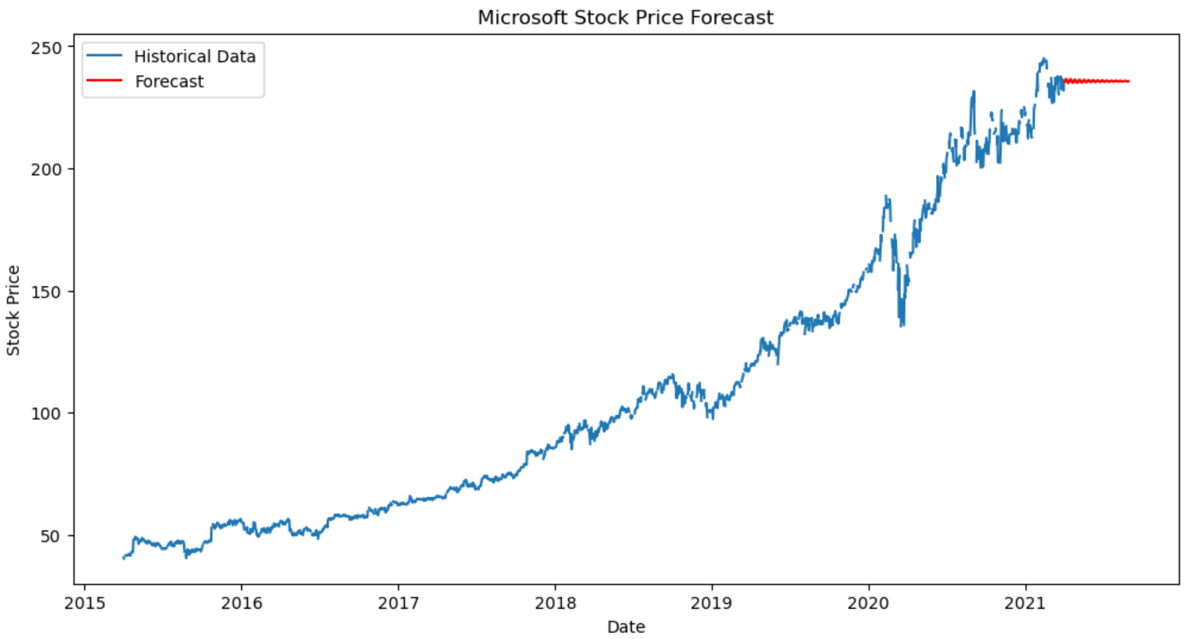


Forecasting

In [22]:
```
forecast = fit_model.get_forecast(steps=150)
forecast
```

Out[22]: <statsmodels.tsa.statespace.mlemodel.PredictionResultsWrapper at 0x2276292d210>

In [23]:
```
# Visualization and Interpretation
plt.figure(figsize=(12, 6))
plt.plot(df['Close'], label='Historical Data')
plt.plot(forecast.predicted_mean, label='Forecast', color='red')
plt.title('Microsoft Stock Price Forecast')
plt.xlabel('Date')
plt.ylabel('Stock Price')
plt.legend()
plt.show()
```



In [24]:
```python
# Reporting
fit_model.summary()
```

Out[24]:

SARIMAX Results

Dep. Variable:	Close	**No. Observations:**	2192
Model:	ARIMA(5, 1, 5)	**Log Likelihood**	-3328.254
Date:	Sun, 07 Jan 2024	**AIC**	6678.507
Time:	22:35:56	**BIC**	6741.120
Sample:	04-01-2015	**HQIC**	6701.392
	- 03-31-2021		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	-0.1114	0.054	-2.043	0.041	-0.218	-0.005
ar.L2	1.0703	0.014	76.145	0.000	1.043	1.098
ar.L3	-0.2797	0.057	-4.887	0.000	-0.392	-0.167
ar.L4	-0.9367	0.015	-61.010	0.000	-0.967	-0.907
ar.L5	0.0787	0.051	1.537	0.124	-0.022	0.179
ma.L1	-0.2097	0.049	-4.254	0.000	-0.306	-0.113
ma.L2	-1.0694	0.014	-78.950	0.000	-1.096	-1.043
ma.L3	0.5965	0.048	12.419	0.000	0.502	0.691
ma.L4	0.8781	0.019	45.140	0.000	0.840	0.916
ma.L5	-0.3739	0.045	-8.361	0.000	-0.462	-0.286
sigma2	4.1308	0.072	57.559	0.000	3.990	4.271

Ljung-Box (L1) (Q):	0.22	**Jarque-Bera (JB):**	23906.40
Prob(Q):	0.64	**Prob(JB):**	0.00
Heteroskedasticity (H):	17.54	**Skew:**	-0.55
Prob(H) (two-sided):	0.00	**Kurtosis:**	19.14

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

In [ ]: