In [1]:
```
import pandas as pd
import numpy as np
```

In [2]:
```
mrf_df = pd.read_excel('MRF Stock Price.xlsx')
```

In [3]:
```
mrf_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 34 entries, 0 to 33
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Year    34 non-null     int64  
 1   Price   34 non-null     float64
dtypes: float64(1), int64(1)
memory usage: 672.0 bytes
```

In [4]:
```
mrf_df.head(34)
```

Out[4]:

	Year	Price
0	1990	350.00
1	1991	630.00
2	1992	925.00
3	1993	1250.00
4	1994	2350.00
5	1995	1750.00
6	1996	2850.00
7	1997	1933.25
8	1998	1430.00
9	1999	2187.70
10	2000	1217.10
11	2001	710.35
12	2002	900.00
13	2003	2301.15
14	2004	2420.15
15	2005	2750.95
16	2006	4303.50
17	2007	7220.65
18	2008	2002.45
19	2009	6034.00
20	2010	7207.60
21	2011	6991.35
22	2012	12806.85
23	2013	19372.15
24	2014	37882.90

25	2015	39847.60
26	2016	48781.15
27	2017	72348.15
28	2018	67088.25
29	2019	66327.75
30	2020	75712.40
31	2021	73293.55
32	2022	88531.10
33	2023	109035.05

In [5]:
```
mrf_df.set_index(mrf_df.Year, inplace= True)
```

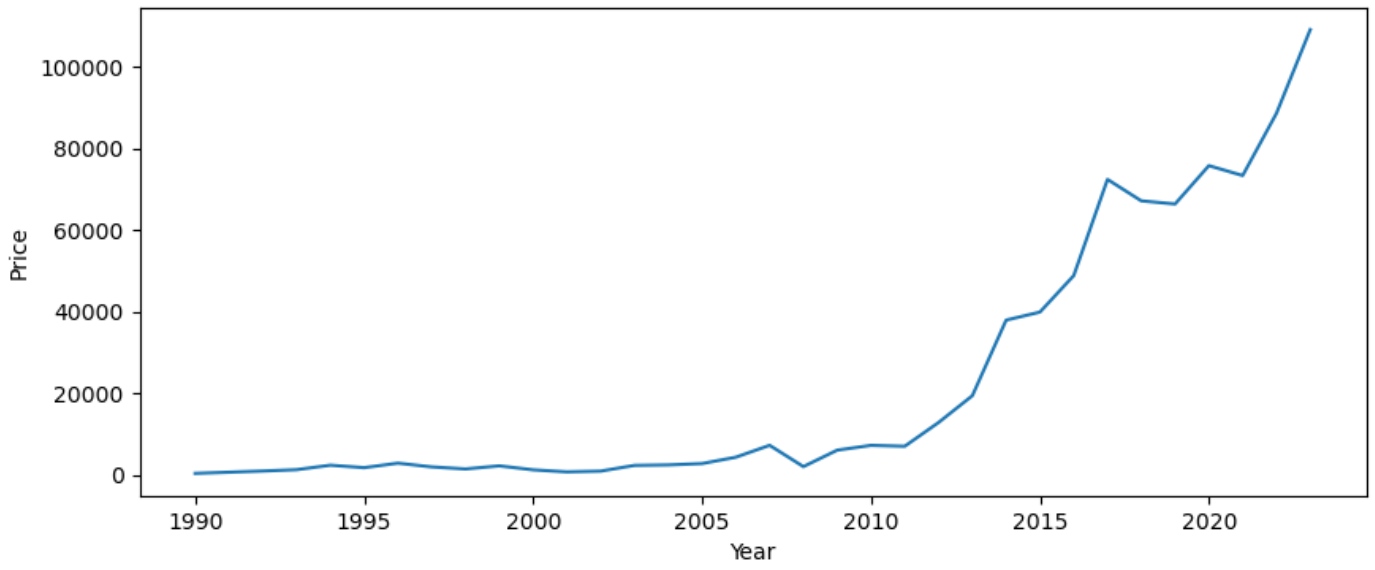
In [6]:
```
mrf_df.head(5)
```

Out[6]:

	Year	Price
Year		
1990	1990	350.0
1991	1991	630.0
1992	1992	925.0
1993	1993	1250.0
1994	1994	2350.0

In [7]:
```
import matplotlib.pyplot as plt
import seaborn as sn
%matplotlib inline
```

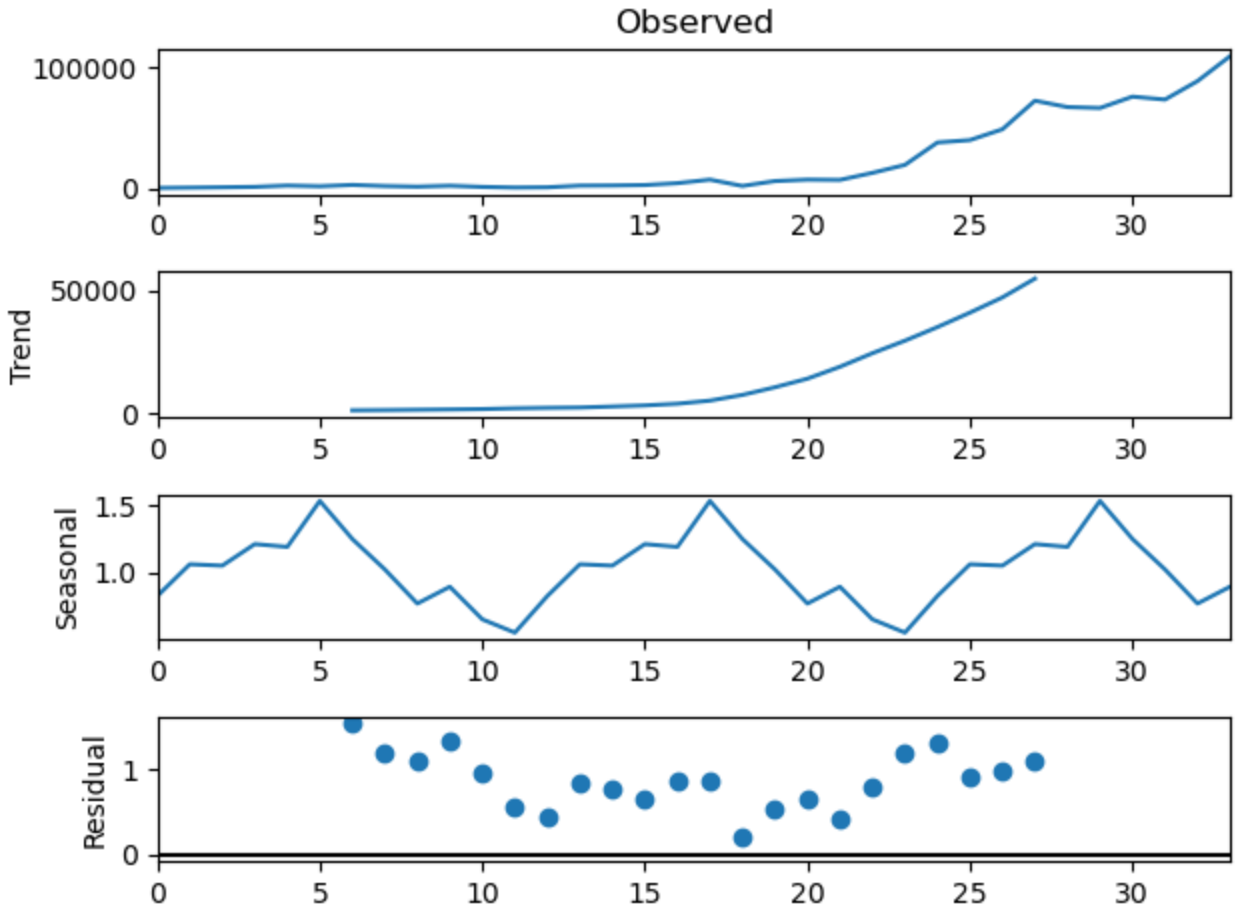
In [8]:
```
plt.figure(figsize=(10,4))
plt.xlabel('Year')
plt.ylabel('Price')
plt.plot(mrf_df['Price']);
```



In [9]:
```
from statsmodels.tsa.seasonal import seasonal_decompose
```

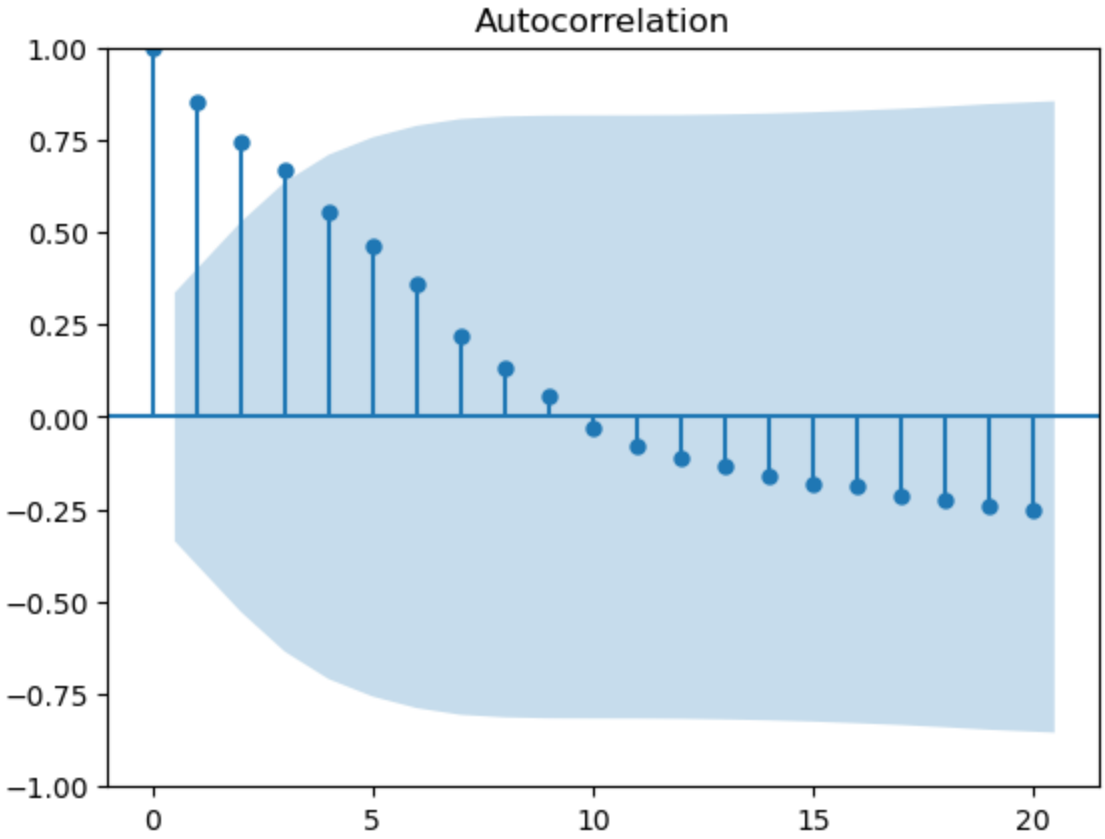
In [10]:
```
ts_decompose = seasonal_decompose(np.array(mrf_df['Price']), model= 'multiplicative', pe
```

In [11]:
```
ts_plot = ts_decompose.plot()
```



In [12]:
```
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
```

In [13]:
```
acf_plot = plot_acf(mrf_df.Price, lags= 20)
```



In [14]:
```
from statsmodels.tsa.stattools import adfuller
```

In [15]:
```
def adfuller_test(ts):
    adfuller_result = adfuller(ts, autolag= None)
    adfuller_out = pd.Series(adfuller_result[0:4], index= ['Test Statistics', 'p-value',
                                                           'Number of observations used']

    print(adfuller_out)
```

In [16]:
```
adfuller_test(mrf_df.Price)
```

```
Test Statistics                  2.056582
p-value                          0.998742
Lags Used                       10.000000
Number of observations used     23.000000
dtype: float64
```

In [17]:
```
mrf_df['diff_p'] = mrf_df.Price - mrf_df.Price.shift(1)
```

In [18]:
```
mrf_df.head(5)
```

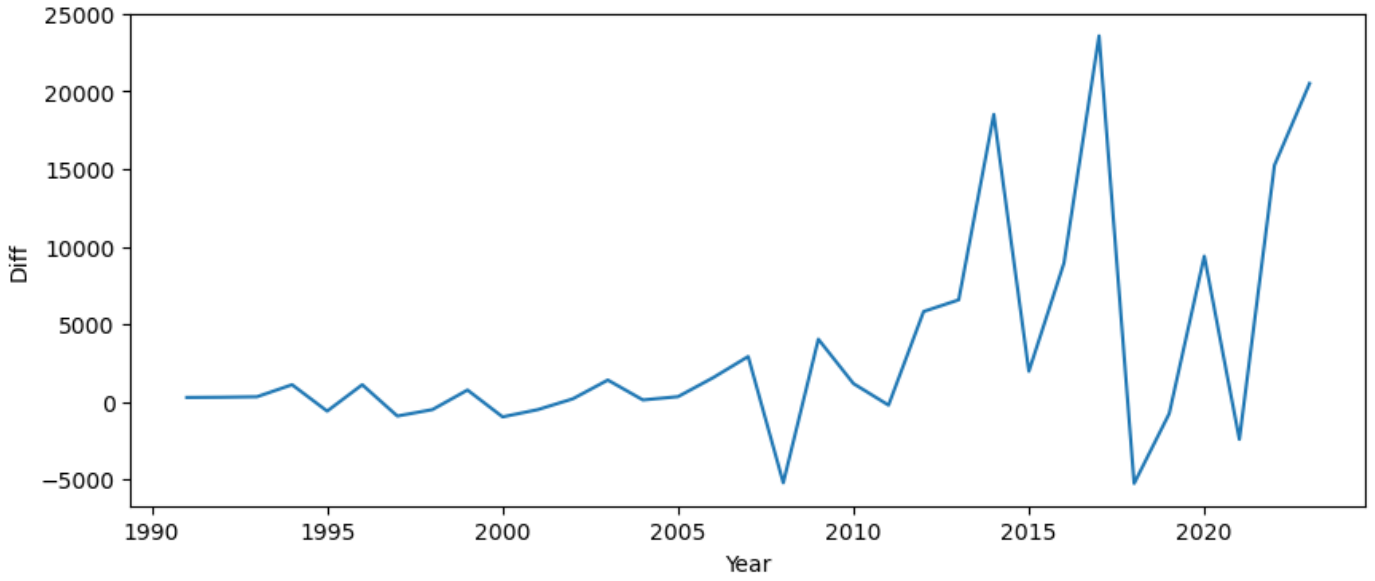
Out[18]:

	Year	Price	diff_p
Year			
1990	1990	350.0	NaN
1991	1991	630.0	280.0
1992	1992	925.0	295.0
1993	1993	1250.0	325.0
1994	1994	2350.0	1100.0

In [19]:
```
mrf_df_diff = mrf_df.dropna()
```

In [20]:
```
plt.figure(figsize=(10,4))
plt.xlabel('Year')
plt.ylabel('Diff')
plt.plot(mrf_df_diff.diff_p)
```

Out[20]:
```
[<matplotlib.lines.Line2D at 0x7fda26b507c0>]
```

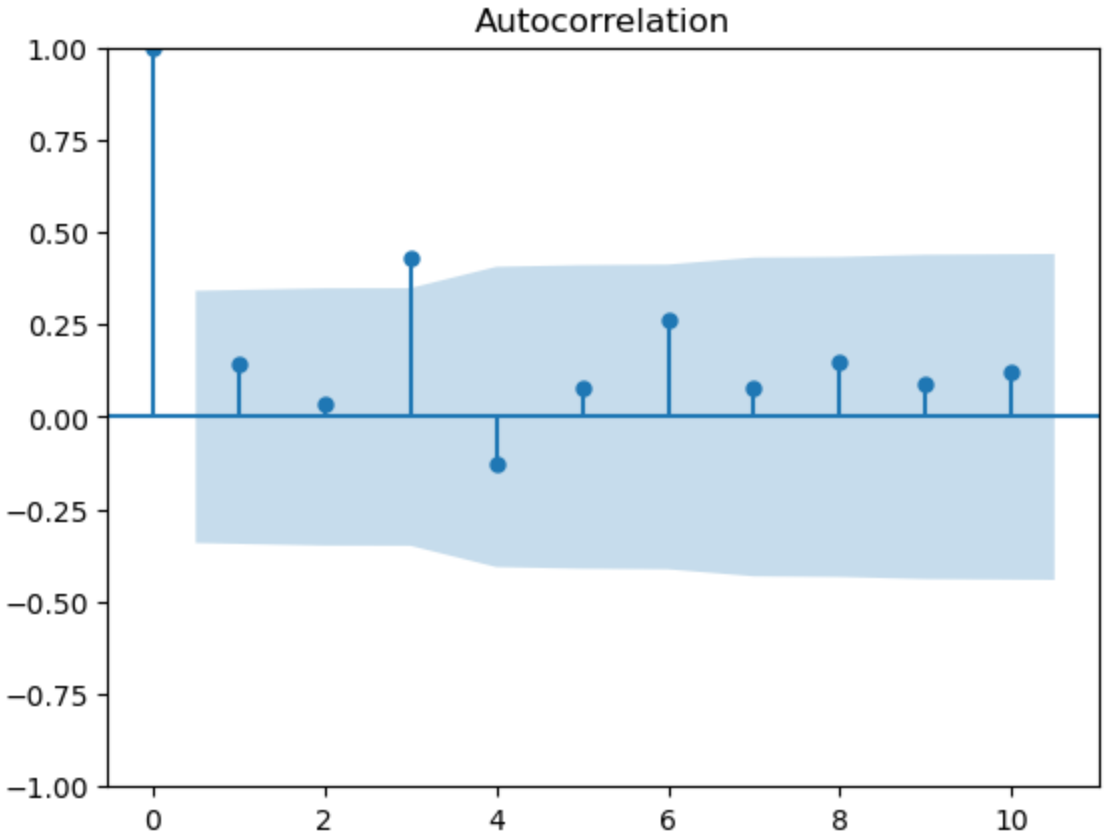


In [21]:
```
mrf_df_diff.head(5)
```

Out[21]:

	Year	Price	diff_p
Year			
1991	1991	630.0	280.0
1992	1992	925.0	295.0
1993	1993	1250.0	325.0
1994	1994	2350.0	1100.0
1995	1995	1750.0	-600.0

In [22]:
```
acf_plot = plot_acf(mrf_df_diff.diff_p, lags= 10)
```



In [23]:
```
from statsmodels.tsa.arima.model import ARIMA
```

In [24]:
```
mrf_df_diff_train = mrf_df_diff[0:20]
mrf_df_diff_test = mrf_df_diff[20:]
```

In [25]:
```
arima = ARIMA(mrf_df_diff_train.Price.astype(np.float64).values, order = (1,1,1))
```

In [26]:
```
arima_model = arima.fit()
```

In [27]:
```
arima_model.summary()
```

Out[27]:

SARIMAX Results

Dep. Variable:	y	**No. Observations:**	20
Model:	ARIMA(1, 1, 1)	**Log Likelihood**	-167.105
Date:	Thu, 16 Nov 2023	**AIC**	340.210
Time:	18:39:27	**BIC**	343.043
Sample:	0	**HQIC**	340.689
	- 20		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	-0.2287	0.382	-0.598	0.550	-0.978	0.520
ma.L1	-0.3415	0.350	-0.976	0.329	-1.027	0.345
sigma2	2.538e+06	7.55e+05	3.360	0.001	1.06e+06	4.02e+06

Ljung-Box (L1) (Q):	0.53	**Jarque-Bera (JB):**	0.82
Prob(Q):	0.47	**Prob(JB):**	0.66

Heteroskedasticity (H):	11.23	**Skew:**	-0.19
Prob(H) (two-sided):	0.01	**Kurtosis:**	3.94

Warnings:

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

In [28]:
```
def get_mape(actual, predicted):
    y_true, y_pred = np.array(actual), np.array(predicted)
    return np.round(np.mean(np.abs((actual-predicted)/predicted))*100,2)
```

In [29]:
```
mrf_df_diff_test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 13 entries, 2011 to 2023
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Year    13 non-null     int64  
 1   Price   13 non-null     float64
 2   diff_p  13 non-null     float64
dtypes: float64(2), int64(1)
memory usage: 416.0 bytes
```

In [30]:
```
mrf_df_diff_test.head(5)
```

Out[30]:

	Year	Price	diff_p
Year			
2011	2011	6991.35	-216.25
2012	2012	12806.85	5815.50
2013	2013	19372.15	6565.30
2014	2014	37882.90	18510.75
2015	2015	39847.60	1964.70

In [34]:
```
mrf_predict = arima_model.forecast(steps = 3)
```

In [35]:
```
mrf_predict
```

Out[35]:
```
array([6020.70406842, 6292.18806696, 6230.09032149])
```

In [36]:
```
get_mape(mrf_df_diff_test, mrf_predict)
```

```
/Users/ishutejwani/opt/anaconda3/lib/python3.9/site-packages/numpy/core/fromnumeric.py:3462: FutureWarning: In a future version, DataFrame.mean(axis=None) will return a scalar mean over the entire DataFrame. To retain the old behavior, use 'frame.mean(axis=0)' or just 'frame.mean()'
  return mean(axis=axis, dtype=dtype, out=out, **kwargs)
```

Out[36]:
```
Year       66.50
Price     777.79
diff_p    120.20
dtype: float64
```

Forecasting using rolling moving average

In [37]:
```
mrf_df['rollavg'] = mrf_df['Price'].rolling(window = 12).mean().shift(1)
```

In [40]:
```
pd.set_option('display.float_format', lambda x: '%.2f' %x)
mrf_df[['Price', 'rollavg']][24:]
```

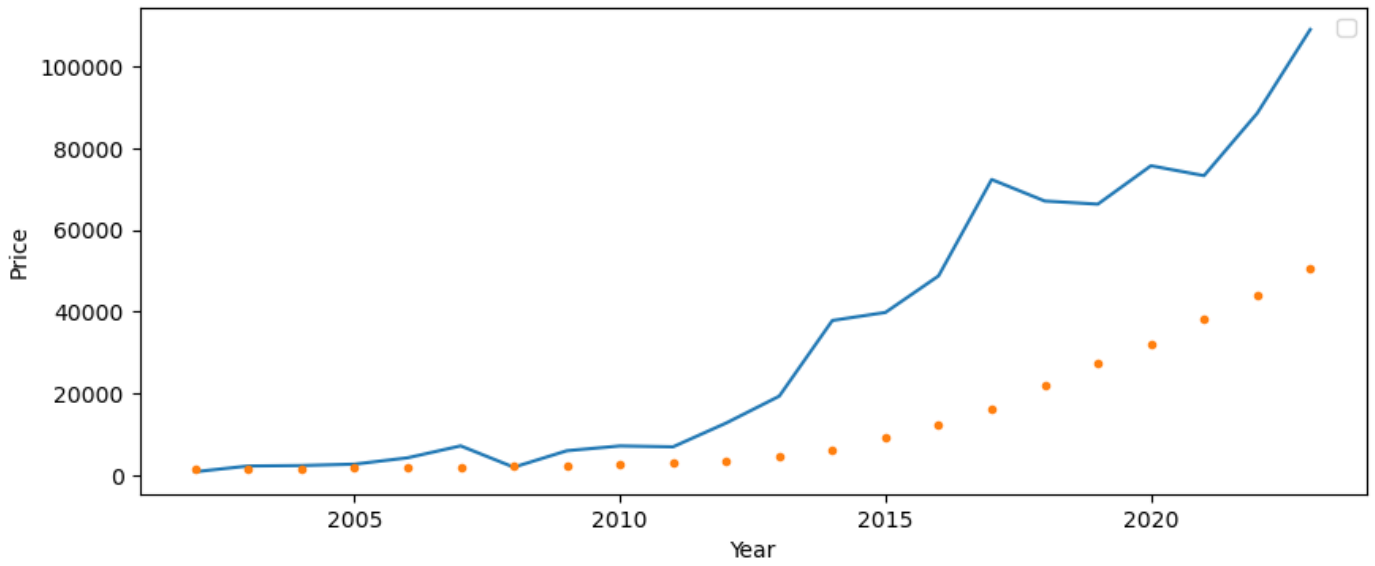
Out[40]:

Year	Price	rollavg
2014	37882.90	6192.57
2015	39847.60	9274.48
2016	48781.15	12403.35
2017	72348.15	16266.76
2018	67088.25	22066.53
2019	66327.75	27298.59
2020	75712.40	32224.18
2021	73293.55	38366.68
2022	88531.10	43971.64
2023	109035.05	50748.60

In [41]:
```
plt.figure(figsize= (10,4))
plt.xlabel('Year')
plt.ylabel('Price')
plt.plot(mrf_df['Price'][12:])
plt.plot(mrf_df['rollavg'][12:], '.')
plt.legend()
```

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

Out[41]:
```
<matplotlib.legend.Legend at 0x7fda0d20f880>
```



In [43]:
```
get_mape(mrf_df['Price'][24:].values, mrf_df['rollavg'][24:].values)
```

Out[43]:
```
226.86
```

Forecasting using exponential smoothing

In [54]:
```
mrf_df['es'] = mrf_df.Price.ewm(alpha= 0.5).mean()
```

In [55]:
```
mrf_df['es'][24:]
```

Out[55]:
```
Year
2014    26198.76
2015    33023.18
2016    40902.16
2017    56625.16
2018    61856.70
2019    64092.23
2020    69902.31
2021    71597.93
2022    80064.52
2023    94549.78
Name: es, dtype: float64
```

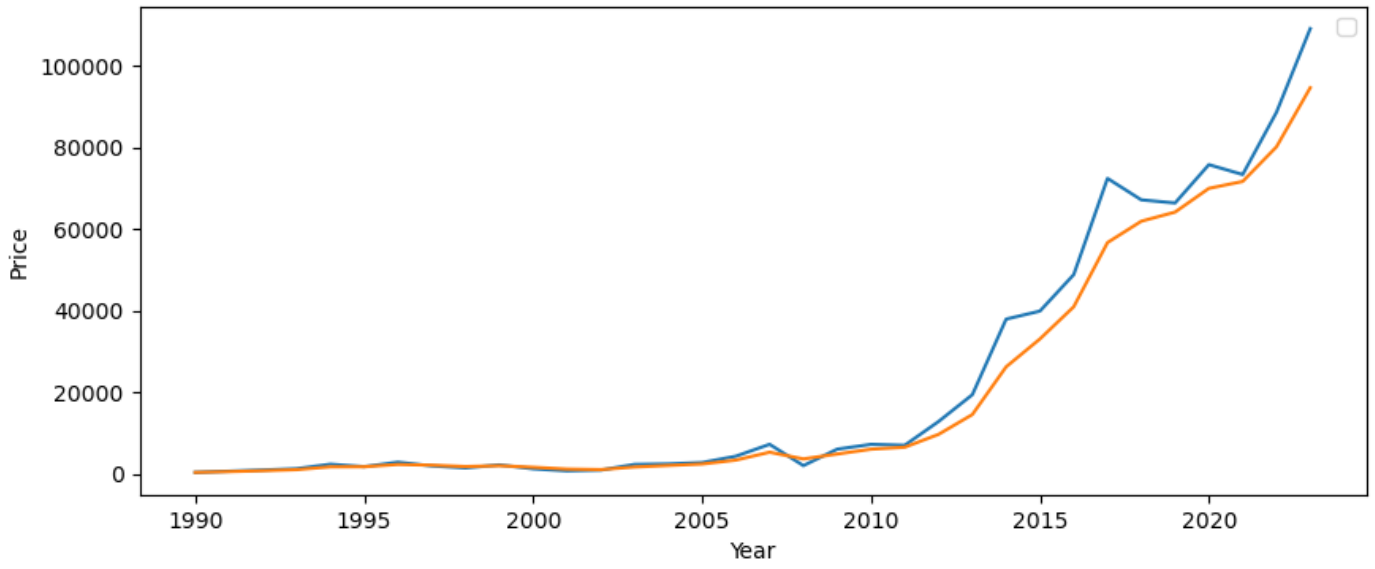
In [56]:
```
get_mape(mrf_df['Price'][24:].values, mrf_df['es'][24:].values)
```

Out[56]:
```
16.08
```

In [57]:
```
plt.figure(figsize=(10,4))
plt.xlabel('Year')
plt.ylabel('Price')
plt.plot(mrf_df.Price)
plt.plot(mrf_df.es)
plt.legend()
```

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

Out[57]:
```
<matplotlib.legend.Legend at 0x7fda0f8cffd0>
```



In [58]:
```
from statsmodels.tsa.holtwinters import ExponentialSmoothing
```

In [59]:
```
emodel = ExponentialSmoothing(mrf_df['Price'], trend= 'multiplicative')
```

```
/Users/ishutejwani/opt/anaconda3/lib/python3.9/site-packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: An unsupported index was provided and will be ignored when e.g. forecasting.
  self._init_dates(dates, freq)
```

In [78]:
```
efitmodel = emodel.fit()
```

```
/Users/ishutejwani/opt/anaconda3/lib/python3.9/site-packages/statsmodels/tsa/holtwinters/model.py:915: ConvergenceWarning: Optimization failed to converge. Check mle_retvals.
  warnings.warn(
```

In [65]:
```
mrf_df.index = pd.to_datetime(mrf_df.index, format='%Y')
```

In [66]:
```
mrf_df.head(5)
```

Out[66]:

	Year	Price	diff_p	rollavg	es
Year					
1990-01-01	1990	350.00	NaN	NaN	350.00
1991-01-01	1991	630.00	280.00	NaN	536.67
1992-01-01	1992	925.00	295.00	NaN	758.57
1993-01-01	1993	1250.00	325.00	NaN	1020.67
1994-01-01	1994	2350.00	1100.00	NaN	1706.77

In [79]:
```
emodel = ExponentialSmoothing(mrf_df['Price'], trend= 'multiplicative')
```

```
/Users/ishutejwani/opt/anaconda3/lib/python3.9/site-packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency information was provided, so inferred frequency AS-JAN will be used.
  self._init_dates(dates, freq)
```

without smoothing level

In [92]:
```
fitresult = emodel.fit()
```

```
/Users/ishutejwani/opt/anaconda3/lib/python3.9/site-packages/statsmodels/tsa/holtwinters/model.py:915: ConvergenceWarning: Optimization failed to converge. Check mle_retvals.
  warnings.warn(
```

In [69]:
```
from pandas.tseries.offsets import DateOffset
```

In [70]:
```
future_date = [mrf_df.index[-1]+DateOffset(years = x) for x in range (0,5)]
```

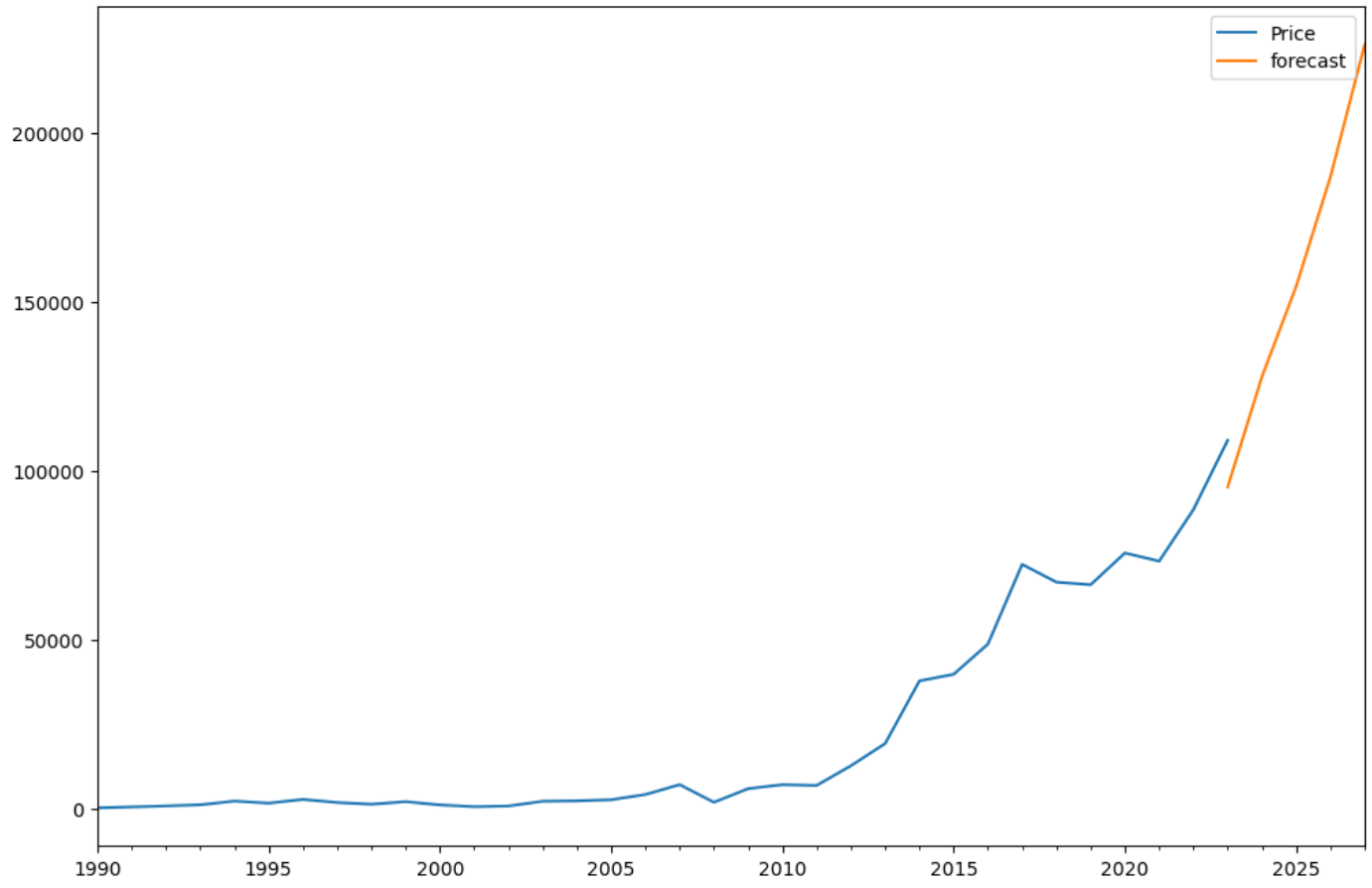
In [71]:
```
future_date_df = pd.DataFrame(index= future_date[1:], columns= mrf_df.columns)
```

In [72]:
```
future_df = pd.concat([mrf_df, future_date_df])
```

In [93]:
```
future_df['forecast'] = fitresult.predict(start = 33, end = 44)
```

In [94]:
```
future_df[['Price', 'forecast']].plot(figsize= (12,8))
```

Out[94]:
```
<AxesSubplot:>
```



with smoothing level

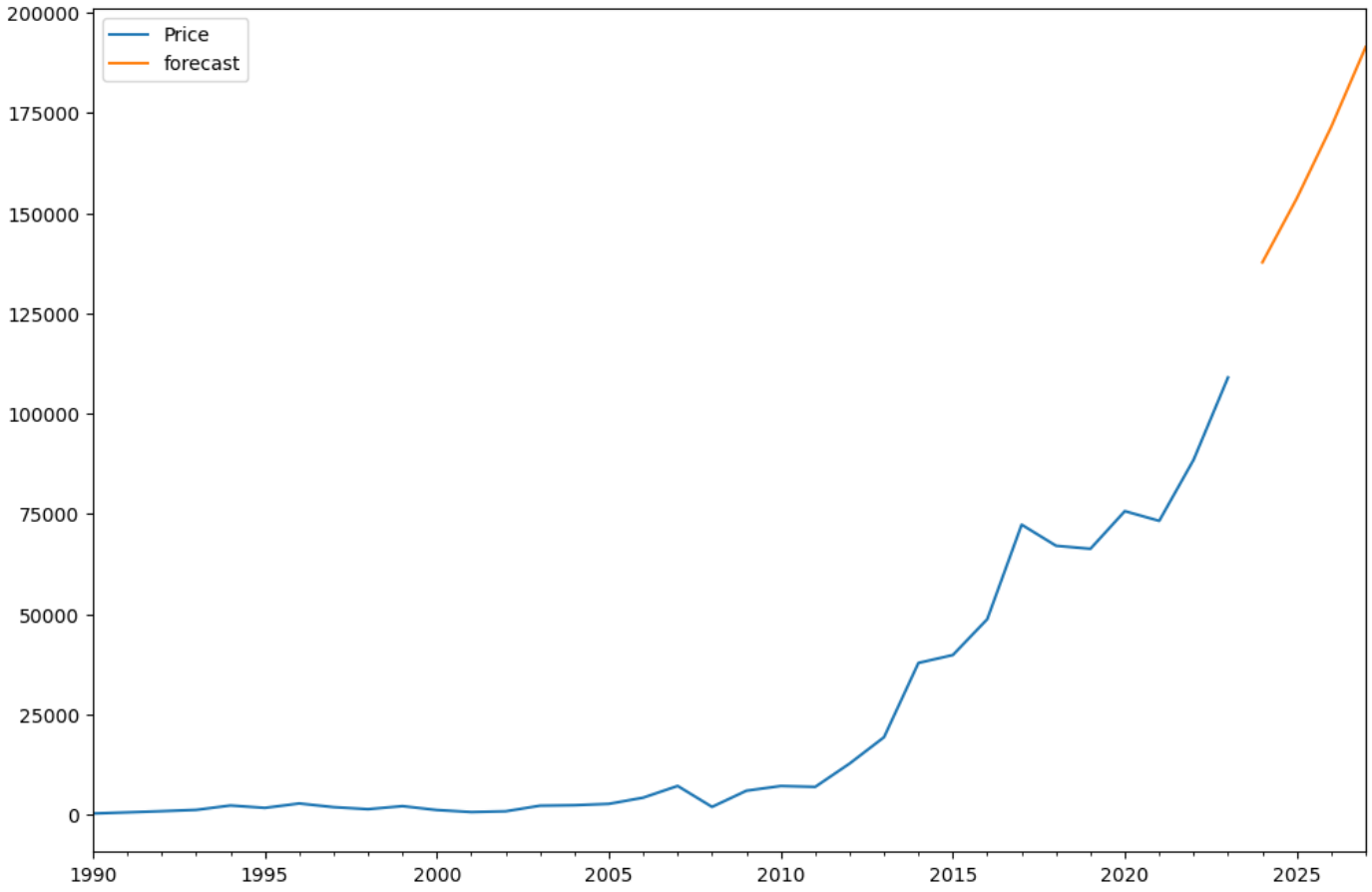
In [130…
```
fitresult = emodel.fit(smoothing_level= 0.1)
```

```
/Users/ishutejwani/opt/anaconda3/lib/python3.9/site-packages/statsmodels/tsa/holtwinters/model.py:915: ConvergenceWarning: Optimization failed to converge. Check mle_retvals.
  warnings.warn(
```

In [131…
```
future_df['forecast'] = fitresult.forecast(steps = 5)
```

In [132…
```
future_df[['Price', 'forecast']].plot(figsize= (12,8))
```

Out[132]: <AxesSubplot:>



In [133…
```
future_df.tail(5)
```

Out[133]:

	Year	Price	diff_p	rollavg	es	forecast
2023-01-01	2023	109035.05	20503.95	50748.60	94549.78	NaN
2024-01-01	NaN	NaN	NaN	NaN	NaN	137726.35
2025-01-01	NaN	NaN	NaN	NaN	NaN	153684.78
2026-01-01	NaN	NaN	NaN	NaN	NaN	171492.32
2027-01-01	NaN	NaN	NaN	NaN	NaN	191363.23