In [ ]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from pandas.plotting import register_matplotlib_converters
register_matplotlib_converters()
%matplotlib inline
```

In [ ]:
```
data = pd.read_excel('Rice_Price_Inflation.xlsx')
data.head()
```

Out[ ]:

	DATE	PRICE	INFLATION	FOOD INFLATION	SECURITY INDEX
0	2012-05-15	14974.95	12.7	12.9	4.5
1	2012-06-15	14700.00	12.9	12.0	4.5
2	2012-07-15	16075.00	12.8	12.1	4.5
3	2012-08-15	16960.00	11.7	11.1	4.5
4	2012-09-15	15600.00	11.3	10.2	4.5

In [ ]:
```
data.isna().sum()
```

Out[ ]:
```
DATE              0
PRICE             0
INFLATION         0
FOOD INFLATION    0
SECURITY INDEX    0
dtype: int64
```

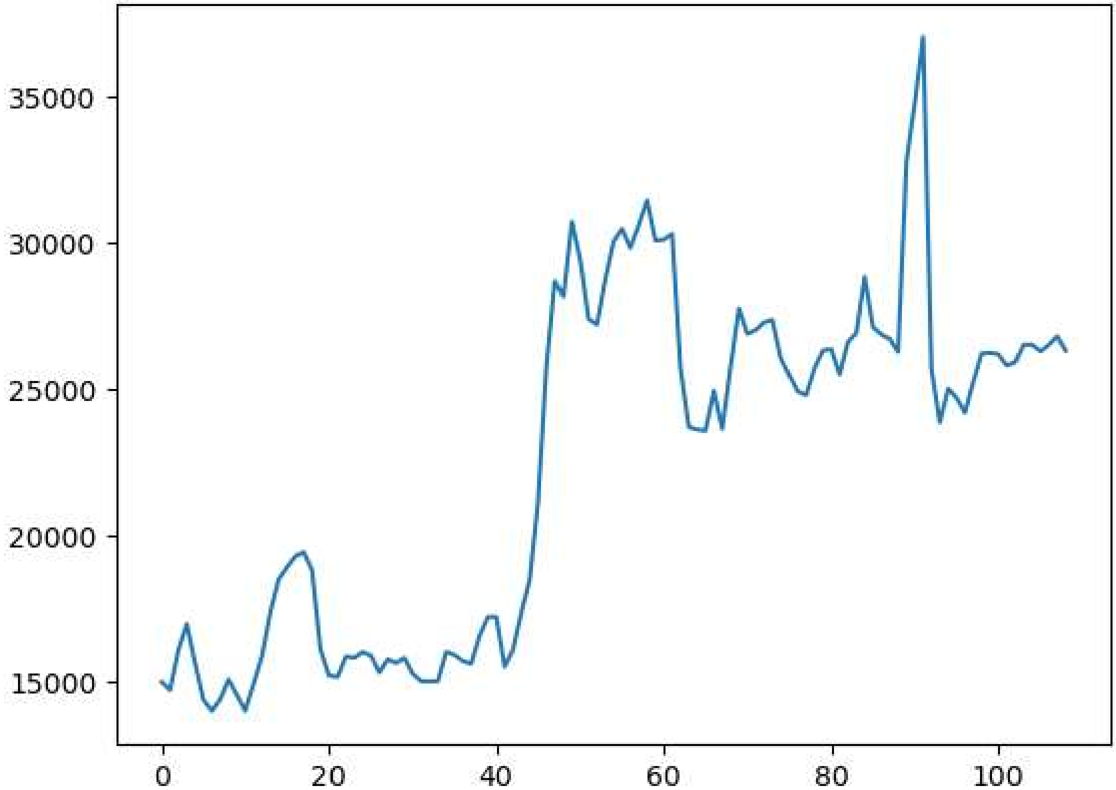
In [ ]:
```
data.describe()
```

Out[ ]:

	PRICE	INFLATION	FOOD INFLATION	SECURITY INDEX
count	109.000000	109.000000	109.000000	109.000000
mean	22523.659633	12.378165	14.002110	8.695138
std	5905.227961	3.395078	4.058187	1.325293
min	14000.000000	7.700000	9.100000	4.500000
25%	15900.000000	9.200000	10.100000	8.700000
50%	24900.000000	11.700000	13.390000	9.000000
75%	26785.710000	15.750000	17.190000	9.500000
max	37000.000000	18.720000	22.950000	9.900000

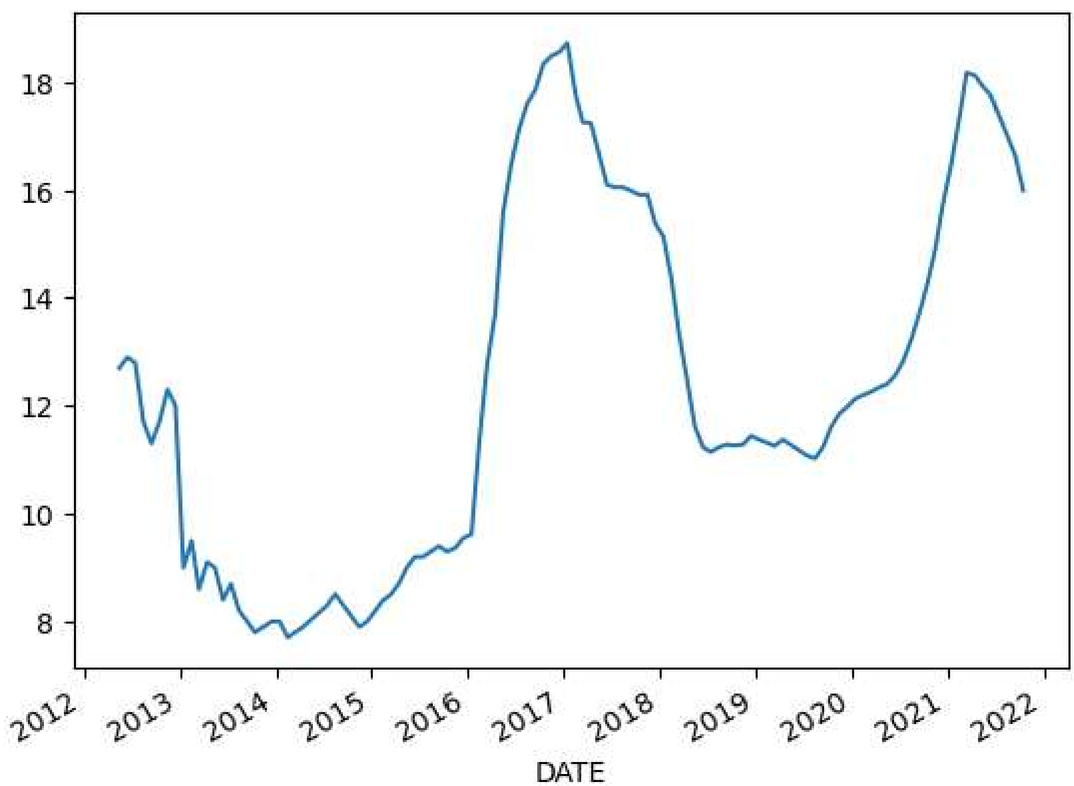
In [ ]:
```
data['PRICE'].plot()
```

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



In [ ]:
```
data['INFLATION'].plot()
```

Out[ ]: <AxesSubplot:xlabel='DATE'>



In [ ]:
```
steps = -1
dataset = data.copy()
```

```
dataset['Actual'] = dataset['PRICE'].shift(steps)
dataset.head()
```

Out[ ]:

	DATE	PRICE	INFLATION	FOOD INFLATION	SECURITY INDEX	Actual
0	2012-05-15	14974.95	12.7	12.9	4.5	14700.0
1	2012-06-15	14700.00	12.9	12.0	4.5	16075.0
2	2012-07-15	16075.00	12.8	12.1	4.5	16960.0
3	2012-08-15	16960.00	11.7	11.1	4.5	15600.0
4	2012-09-15	15600.00	11.3	10.2	4.5	14360.0

In [ ]:

```
dataset['DATE'] = pd.to_datetime(dataset['DATE']) #freq='12M')
dataset.index = dataset['DATE']
dataset.drop(columns='DATE',inplace=True)
dataset.head(2)
```

Out[ ]:

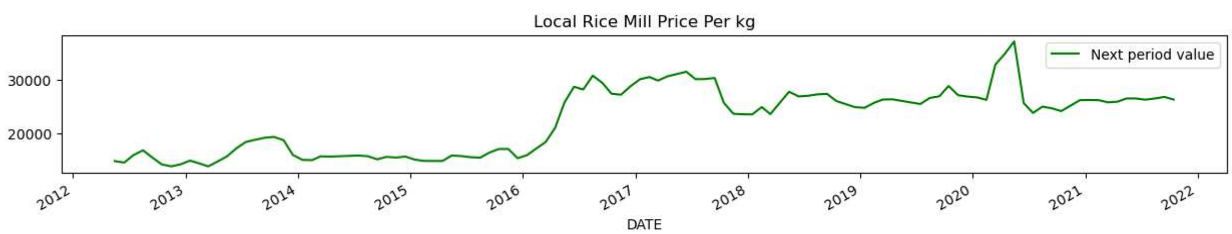
	PRICE	INFLATION	FOOD INFLATION	SECURITY INDEX	Actual
DATE					
2012-05-15	14974.95	12.7	12.9	4.5	14700.0
2012-06-15	14700.00	12.9	12.0	4.5	16075.0

In [ ]:

```
dataset['PRICE'].plot(color='green', figsize=(15,2))
plt.legend(['Next period value', 'prediction'])
plt.title("Local Rice Mill Price Per kg")
```

Out[ ]:

```
Text(0.5, 1.0, 'Local Rice Mill Price Per kg')
```

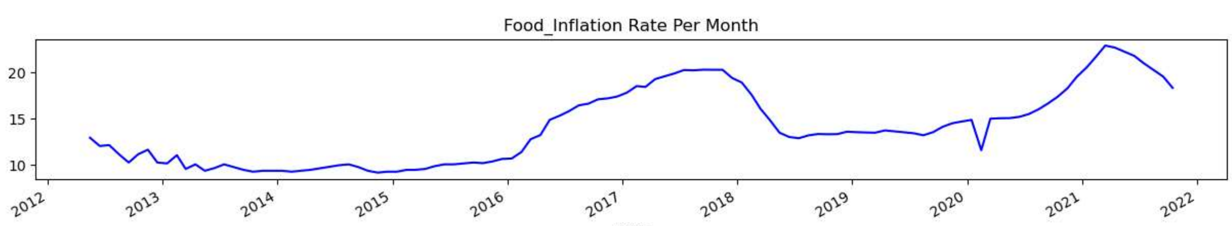


In [ ]:

```
dataset['FOOD INFLATION'].plot(color='blue', figsize=(15,2))
plt.title("Food_Inflation Rate Per Month")
```

Out[ ]:

```
Text(0.5, 1.0, 'Food_Inflation Rate Per Month')
```

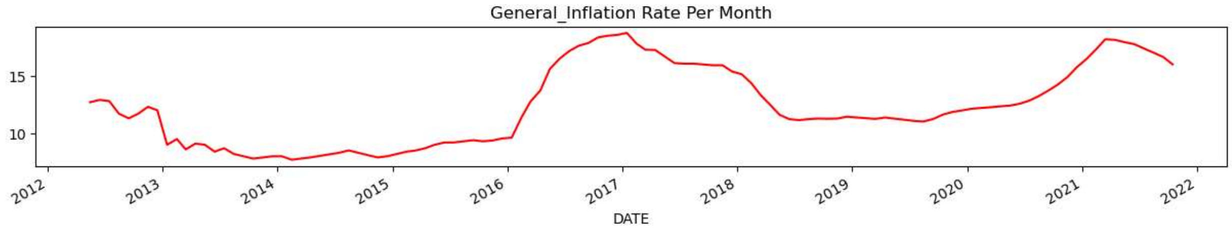


In [ ]:

```
dataset['INFLATION'].plot(color='red', figsize=(15,2))
plt.title("General_Inflation Rate Per Month")
```

Out[ ]:

```
Text(0.5, 1.0, 'General_Inflation Rate Per Month')
```



In [ ]:
```python
from sklearn.preprocessing import MinMaxScaler
sc_in = MinMaxScaler(feature_range=(0, 1))

scaled_input = sc_in.fit_transform(dataset[['PRICE','INFLATION','FOOD INFLATION','SECU

scaled_input = pd.DataFrame(scaled_input)

x = scaled_input
x.head(2)
```

Out[ ]:

	0	1	2	3
0	0.042389	0.453721	0.274368	0.0
1	0.030435	0.471869	0.209386	0.0

In [ ]:
```python
sc_out = MinMaxScaler(feature_range=(0, 1))
scaled_output = sc_out.fit_transform(dataset[['Actual']])

scaled_output = pd.DataFrame(scaled_output)

y = scaled_output
```

In [ ]:
```python
x.rename(columns={0:'Price', 1:'Inflation', 2:'Food_Inflation', 3:'Security_Index'}, i
x.index = dataset.index
x.head()
```

Out[ ]:

	Price	Inflation	Food_Inflation	Security_Index
DATE				
2012-05-15	0.042389	0.453721	0.274368	0.0
2012-06-15	0.030435	0.471869	0.209386	0.0
2012-07-15	0.090217	0.462795	0.216606	0.0
2012-08-15	0.128696	0.362976	0.144404	0.0
2012-09-15	0.069565	0.326679	0.079422	0.0

In [ ]:
```python
y.rename(columns={0:'Price_prediction'}, inplace=True)
y.index= dataset.index
y.head(2)
```

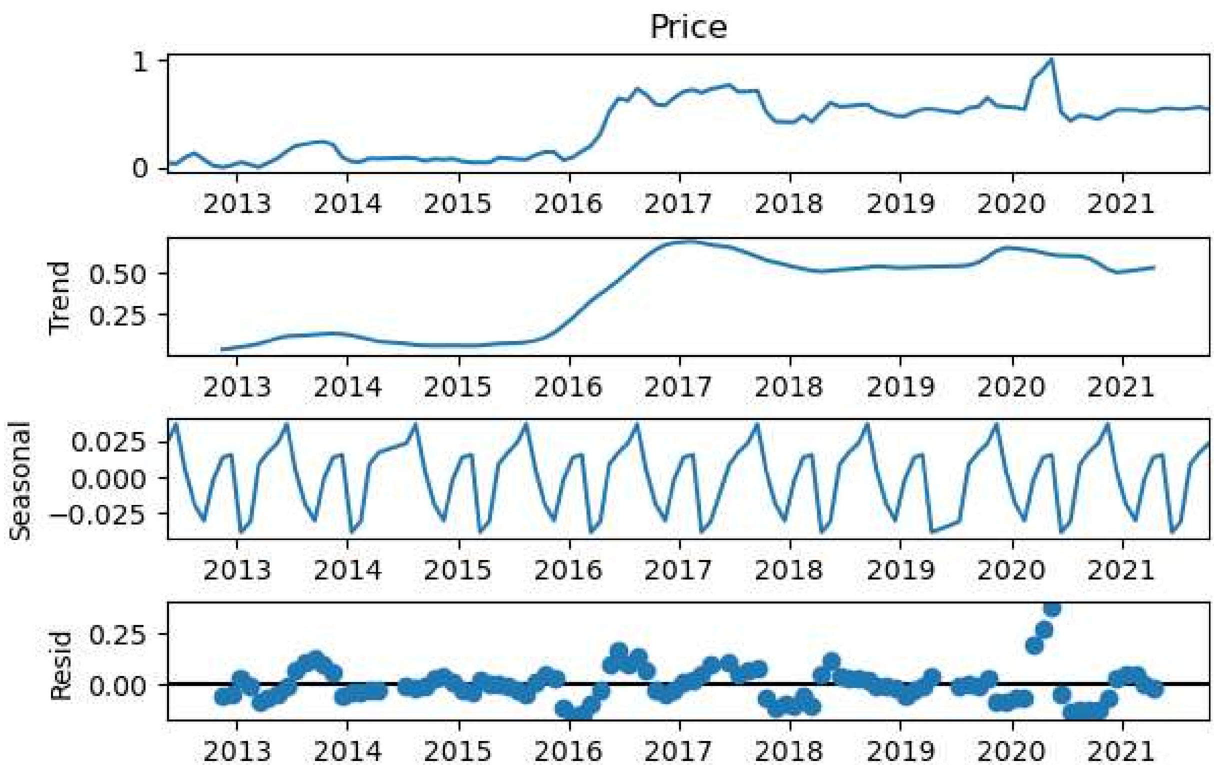
Out[ ]:

DATE	Price_prediction
2012-05-15	0.030435
2012-06-15	0.090217

In [ ]:
```
train_size=int(len(dataset) *0.8)
test_size = int(len(dataset)) - train_size

train_x, train_y = x[:train_size-1].dropna(), y[:train_size-1].dropna()
test_x, test_y = x[train_size:].dropna(), y[train_size :].dropna()
```

In [ ]:
```
import statsmodels.api as sm
seas_d=sm.tsa.seasonal_decompose(x['Price'], model='add', period=12);
fig=seas_d.plot()
fig.set_figheight(4)
plt.show()
```



Check for Data Stationarity using Augmented Dickey-Fuller(ADF) test.

If we make the data stationary, then the model can make predictions based on the fact that mean and variance will remain the same in the future. A stationarized series is easier to predict

To check if the data is stationary, we will use the Augmented Dickey-Fuller test. It is the most popular statistical method to find if the series is stationary or not. It is also called as Unit Root Test. If the value of p <0.05, then the data is stationary.

In [ ]:
```
from statsmodels.tsa.stattools import adfuller
def test_adf(series, title=''):
```

```
    dfout={}
    dftest=sm.tsa.adfuller(series.dropna(), autolag='AIC', regression='ct')
    for key,val in dftest[4].items():
        dfout[f'critical value ({key})']=val
    if dftest[1]<=0.05:
        print("Strong evidence against Null Hypothesis")
        print("Reject Null Hypothesis - Data is Stationary")
        print("Data is Stationary", title)
    else:
        print("Strong evidence for  Null Hypothesis")
        print("Accept Null Hypothesis - Data is not Stationary")
        print("Data is NOT Stationary for", title)
```

In [ ]:

```
y_test=y['Price_prediction'][:train_size].dropna()
test_adf(y_test, " Locally Milled Rice Price")
```

```
Strong evidence for  Null Hypothesis
Accept Null Hypothesis - Data is not Stationary
Data is NOT Stationary for  Locally Milled Rice Price
```

In [ ]:

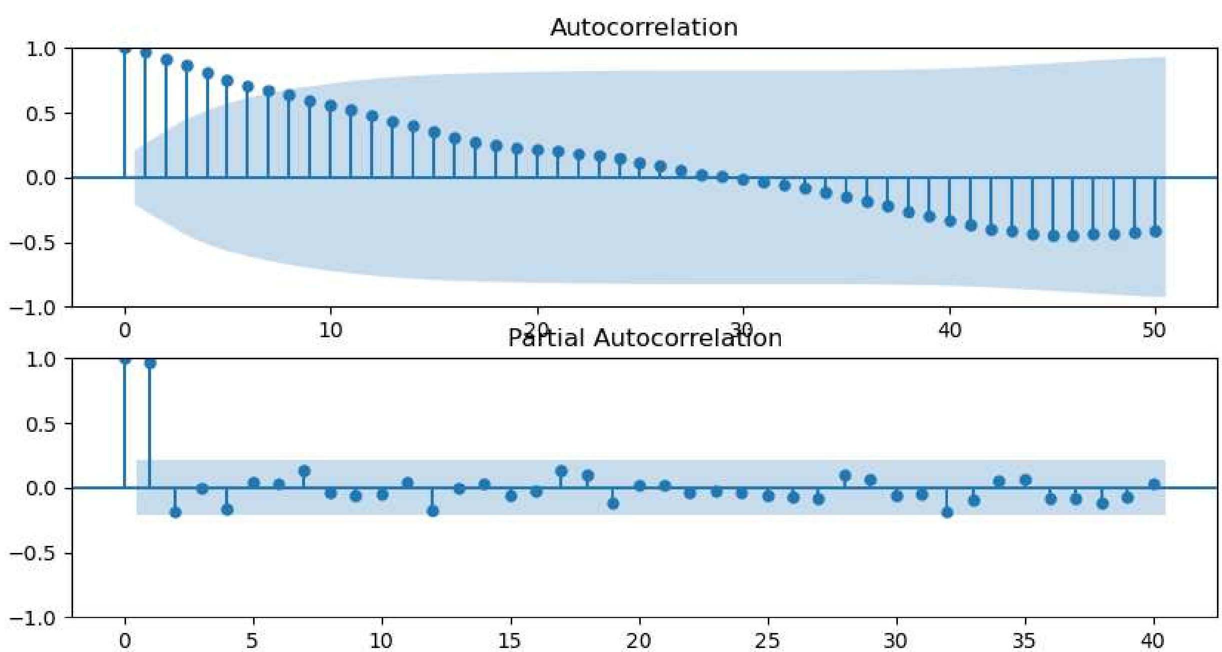
```
y_test_diff = test_adf(y_test.diff(), " -- Rice Price --")
```

```
Strong evidence against Null Hypothesis
Reject Null Hypothesis - Data is Stationary
Data is Stationary  -- Rice Price --
```

We need to know the AR and MA terms to correct any autocorrelation in the differenced series ACF plot: is a bar chart of the coefficients of correlation between a time series and its lags. It helps determine the value of p or the AR term. PACF plot: a plot of the partial correlation coefficients between the series and lags of itself. Helps determine the value of q or the MA term

In [ ]:

```
fig,ax= plt.subplots(2,1, figsize=(10,5))
fig=sm.tsa.graphics.plot_acf(y_test, lags=50, ax=ax[0])
fig=sm.tsa.graphics.plot_pacf(y_test, lags=40, ax=ax[1], method='ywm')
plt.show()
```



We see that the PACF plot has a significant spike at lag 1 and lag 2, meaning that all the higher-order autocorrelations are effectively explained by the lag-1 and lag 2 autocorrelations

In [ ]:
```
import pyramid as pm
step_wise=pm.auto_arima(train_y, exogenous= train_x, start_p=1, start_q=1,
 max_p=7, max_q=7, d=1, max_d=7, trace=True, error_action='ignore',
 suppress_warnings=True, stepwise=True)
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_7216\789207240.py in <module>
----> 1 import pyramid as pm
      2 step_wise=pm.auto_arima(train_y, exogenous= train_x, start_p=1, start_q=1,
      3  max_p=7, max_q=7, d=1, max_d=7, trace=True, error_action='ignore',
      4  suppress_warnings=True, stepwise=True)

ModuleNotFoundError: No module named 'pyramid'
```

In [ ]:

In [ ]:
```
from statsmodels.tsa.statespace.sarimax import SARIMAX
model= SARIMAX(train_y, exog=train_x, order=(0,1,1),
 enforce_invertibility=False, enforce_stationarity=False)
```

```
c:\ProgramData\Anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: Val
ueWarning: A date index has been provided, but it has no associated frequency informa
tion and so will be ignored when e.g. forecasting.
  self._init_dates(dates, freq)
c:\ProgramData\Anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: Val
ueWarning: A date index has been provided, but it has no associated frequency informa
tion and so will be ignored when e.g. forecasting.
  self._init_dates(dates, freq)
```

In [ ]:
```
results = model.fit()
result_summary = results.summary()
```

In [ ]:
```
result_summary
```

Out[ ]:

SARIMAX Results

Dep. Variable:	Price_prediction	**No. Observations:**	86
Model:	SARIMAX(0, 1, 1)	**Log Likelihood**	128.745
Date:	Sun, 12 Feb 2023	**AIC**	-245.491
Time:	15:17:26	**BIC**	-230.978
Sample:	0	**HQIC**	-239.660
	- 86		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
Price	0.3856	0.236	1.633	0.102	-0.077	0.848
Inflation	0.3067	0.188	1.633	0.103	-0.061	0.675
Food_Inflation	-0.2115	0.213	-0.994	0.320	-0.628	0.205
Security_Index	0.0229	0.489	0.047	0.963	-0.935	0.981
ma.L1	-0.1315	0.262	-0.502	0.616	-0.645	0.382
sigma2	0.0026	0.000	8.537	0.000	0.002	0.003

Ljung-Box (L1) (Q):	0.01	**Jarque-Bera (JB):**	24.94
Prob(Q):	0.94	**Prob(JB):**	0.00
Heteroskedasticity (H):	2.96	**Skew:**	-0.64
Prob(H) (two-sided):	0.01	**Kurtosis:**	5.36

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

In [ ]:
```
predictions= results.predict(start =train_size, end=train_size+test_size+(steps)-1,exo
```

```
c:\ProgramData\Anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:834: Val
ueWarning: No supported index is available. Prediction results will be given with an
integer index beginning at `start`.
  return get_prediction_index(
```

In [ ]:
```
act= pd.DataFrame(scaled_output.iloc[train_size:, 0])
```

In [ ]:
```
predictions=pd.DataFrame(predictions)
predictions.reset_index(drop=True, inplace=True)
predictions.index=test_x.index[1:]
predictions['Actual'] = act['Price_prediction']
predictions.rename(columns={0:'Pred'}, inplace=True)
```

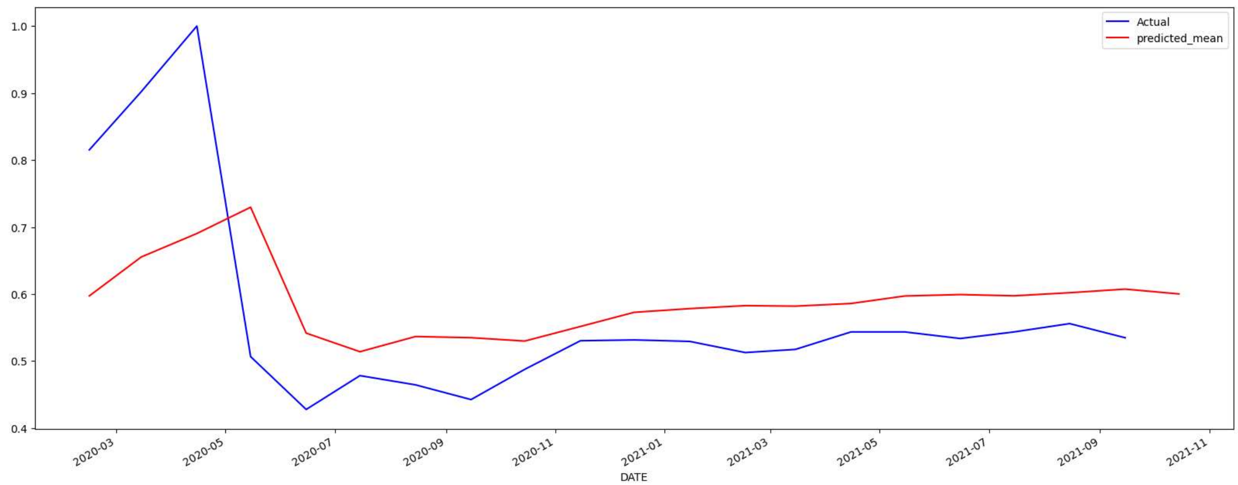
In [ ]:
```
predictions.head(2)
```

Out[ ]:

	predicted_mean	Actual
DATE		
2020-02-15	0.597266	0.815217
2020-03-15	0.655373	0.902174

In [ ]:
```
predictions['Actual'].plot(figsize=(20,8), legend=True, color='blue')
predictions['predicted_mean'].plot(legend=True, color='red', figsize=(20,8))
```

Out[ ]: <AxesSubplot:xlabel='DATE'>



In [ ]:
```
from statsmodels.tools.eval_measures import rmse
error=rmse(predictions['predicted_mean'], predictions['Actual'])
error
```

Out[ ]: nan

In [ ]:
```
trainPredict = sc_out.inverse_transform(predictions[['predicted_mean']])
testPredict = sc_out.inverse_transform(predictions[['Actual']])
```

In [ ]:
```
trainPredict.plot(figsize=(20,8), legend=True, color='blue')
testPredict.plot(legend=True, color='red', figsize=(20,8))
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_7216\3649369440.py in <module>
----> 1 trainPredict.plot(figsize=(20,8), legend=True, color='blue')
      2 testPredict.plot(legend=True, color='red', figsize=(20,8))

AttributeError: 'numpy.ndarray' object has no attribute 'plot'
```

In [ ]:
```
trainPredict
```

Out[ ]:
```
array([[27737.11538133],
       [29073.5721988 ],
       [29878.33477372],
       [30780.72823982],
       [26458.81581969],
       [25820.73420125],
       [26341.94494212],
       [26301.77918815],
       [26186.78728508],
       [26686.9296076 ],
       [27172.9088113 ],
       [27300.10383618],
       [27402.66040124],
       [27385.2794759 ],
       [27475.80630076],
       [27733.61296097],
       [27782.80634754],
       [27739.80764137],
       [27846.34972505],
       [27969.58428183],
       [27804.52933757]])
```

In [ ]:
```
len(testPredict)
```

Out[ ]:
```
21
```

In [ ]:
```
testPredict
```

Out[ ]:
```
array([[32750.  ],
       [34750.  ],
       [37000.  ],
       [25652.17],
       [23838.71],
       [25000.  ],
       [24683.33],
       [24177.42],
       [25216.67],
       [26196.77],
       [26225.81],
       [26175.  ],
       [25791.67],
       [25900.  ],
       [26500.  ],
       [26500.  ],
       [26274.19],
       [26500.  ],
       [26785.71],
       [26300.  ],
       [     nan]])
```