In [ ]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import os
import seaborn as sns
%matplotlib inline
import statsmodels.api as sm
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.statespace.sarimax import SARIMAX
from pmdarima import auto_arima
from statsmodels.graphics.tsaplots import plot_acf , plot_pacf
from pandas.plotting import autocorrelation_plot
from statsmodels.tsa.stattools import adfuller
from math import sqrt
from sklearn import preprocessing
from sklearn.metrics import r2_score , mean_absolute_error , mean_absolut
import pickle
import warnings
warnings.filterwarnings('ignore')
```

Foreign Exchange Rate

In [ ]:
```python
os.chdir('C:\\Users\\santa\\OneDrive\\Documents\\KMUTT-4\\Final_PJ\\Data'
Forex = pd.read_csv('USDTHB_N.csv')
Forex.head()
```

Out[ ]:

	Date	Value
0	03/21/2024	35.915
1	03/20/2024	36.091
2	03/19/2024	35.985
3	03/18/2024	35.890
4	03/15/2024	35.780

In [ ]:
```python
Forex.shape
```

Out[ ]:
```
(5796, 2)
```

In [ ]:
```python
Forex.isnull().sum()
```

Out[ ]:
```
Date      0
Value     0
dtype: int64
```

In [ ]:
```python
Forex.duplicated().sum()
```

Out[ ]:
```
0
```

In [ ]:
```python
Forex.dtypes
```

Out[ ]:
```
Date      object
Value    float64
dtype: object
```

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

Out[ ]:

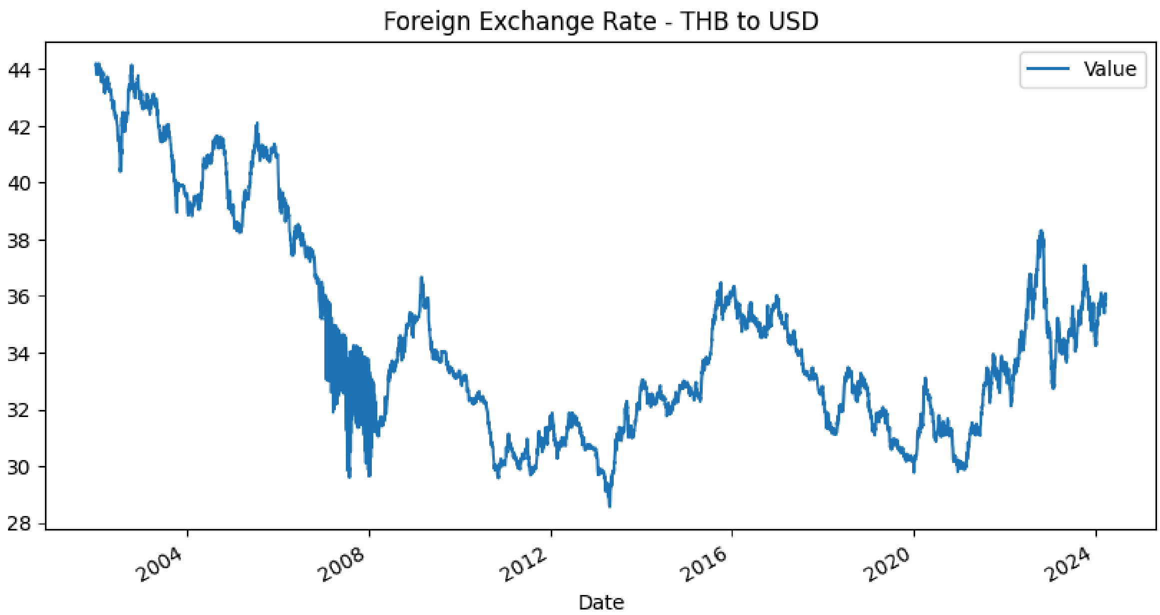
	Value
count	5796.000000
mean	34.561396
std	3.757668
min	28.560000
25%	31.630000
50%	33.440000
75%	36.052500
max	44.200000

Data Processing

In [ ]:
```
Forex['Date'] = pd.to_datetime(Forex['Date'])
```

In [ ]:
```
Forex.set_index('Date',inplace = True)
```

In [ ]:
```
Forex.plot(figsize = (10,5))
plt.title('Foreign Exchange Rate - THB to USD')
#plt.savefig('Foreign Exchange Rate - THB to USD.png')
plt.show()
```



In [ ]:
```
Forex_week = Forex.resample('W').mean()
print('Count of The Weekly Data Frame : ',Forex_week.shape[0])
Forex_week.head()
```

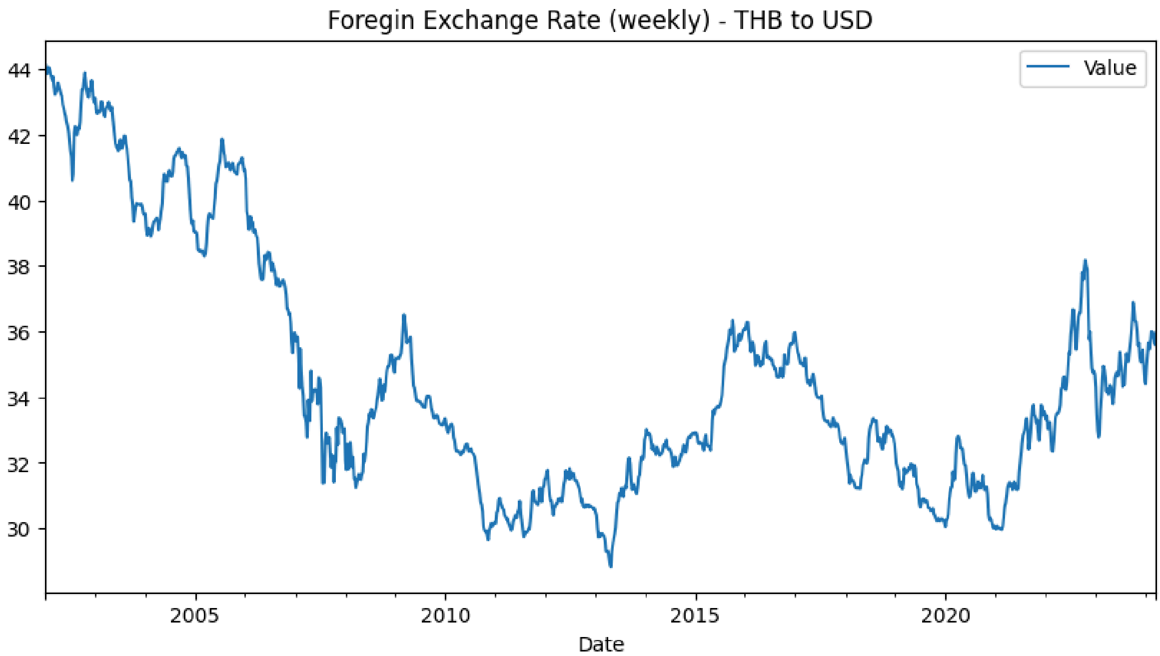
```
Count of The Weekly Data Frame :  1160
```

Out[ ]:

Date	Value
2002-01-06	44.136667
2002-01-13	43.972000
2002-01-20	43.876000
2002-01-27	44.076000
2002-02-03	44.026000

In [ ]:

```python
Forex_week.plot(figsize = (10,5))
plt.title('Foregin Exchange Rate (weekly) - THB to USD')
#plt.savefig('Foregin Exchange Rate (weekly) - THB to USD.png')
plt.show()
```



In [ ]:

```python
Forex_month = Forex.resample('M').mean()
print('Count of The Monthly Data Frame : ',Forex_month.shape[0])
Forex_month.head()
```

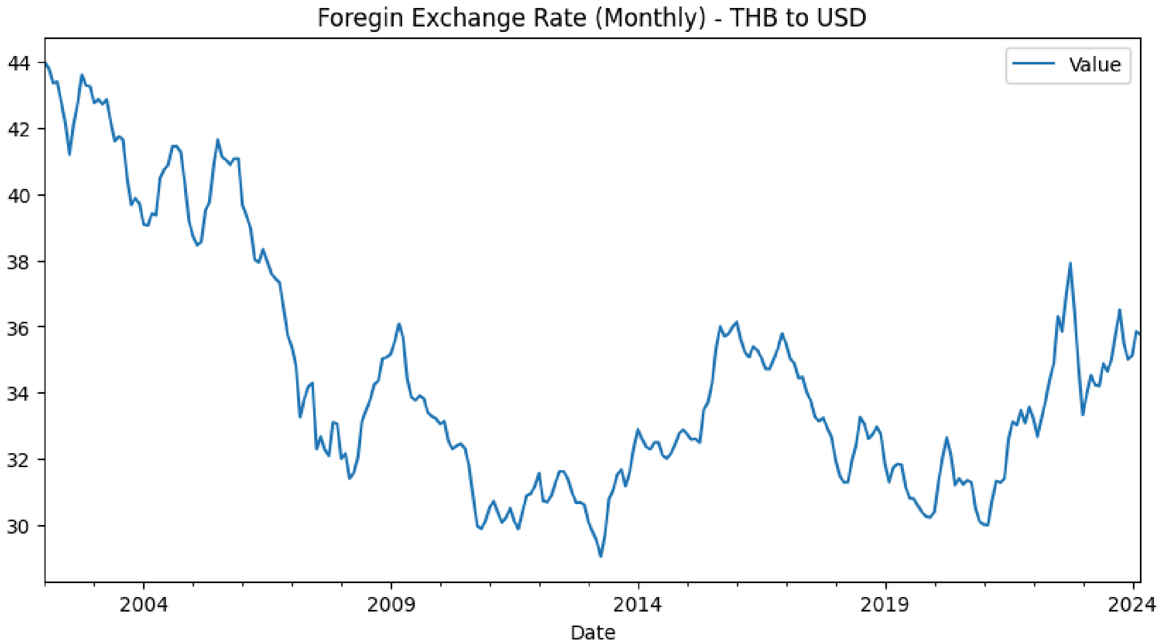
```
Count of The Monthly Data Frame :  267
```

Out[ ]:

Date	Value
2002-01-31	44.004545
2002-02-28	43.809500
2002-03-31	43.370000
2002-04-30	43.407273

2002-05-31	42.807826

In [ ]:
```
Forex_month.plot(figsize = (10,5))
plt.title('Foregin Exchange Rate (Monthly) - THB to USD')
#lt.savefig('Foregin Exchange Rate (Monthly) - THB to USD')
plt.show()
```



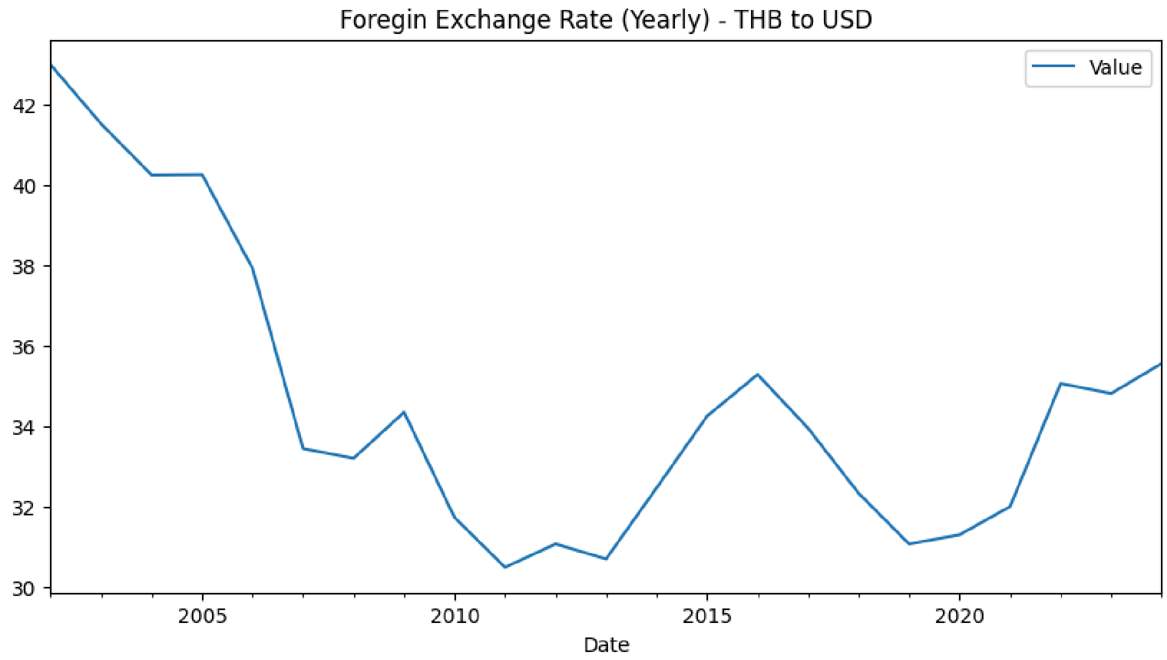
In [ ]:
```
Forex_year = Forex.resample('Y').mean()
print('Count of The Yearly Data Frame : ',Forex_year.shape[0])
Forex_year.head()
```

```
Count of The Yearly Data Frame :  23
```

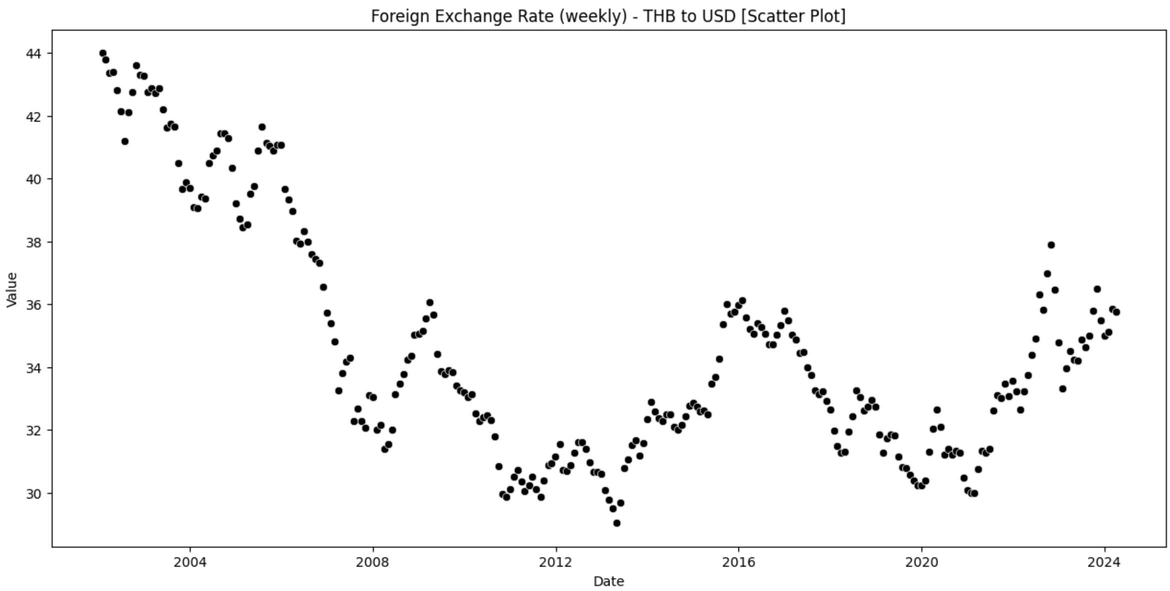
Out[ ]:

Date	Value
2002-12-31	42.977923
2003-12-31	41.508686
2004-12-31	40.236854
2005-12-31	40.245931
2006-12-31	37.911931

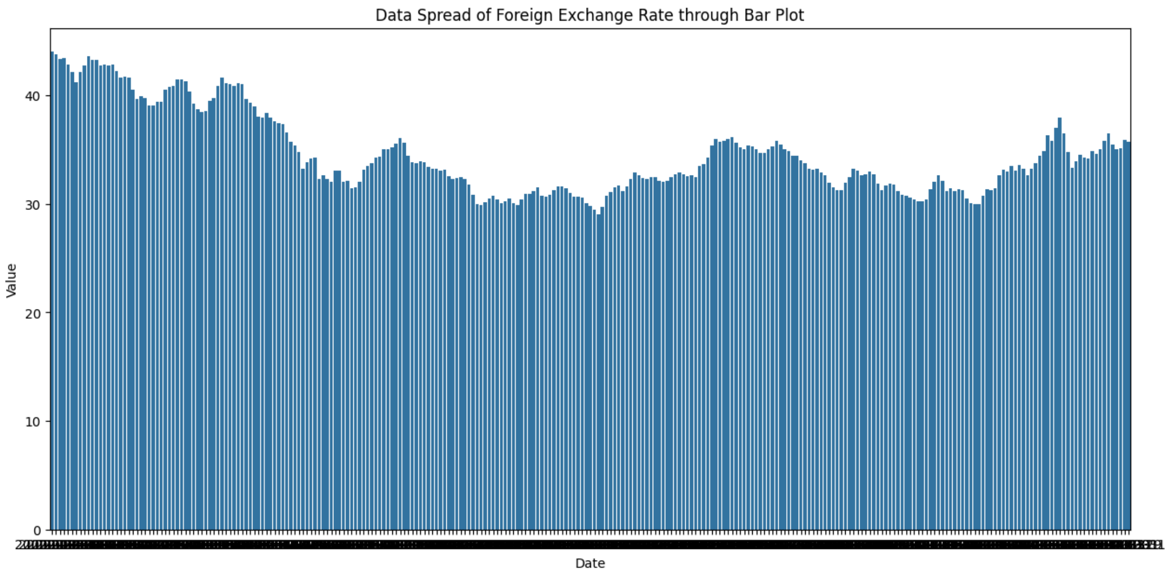
In [ ]:
```
Forex_year.plot(figsize = (10,5))
plt.title('Foregin Exchange Rate (Yearly) - THB to USD')
#plt.savefig('Foregin Exchange Rate (Yearly) - THB to USD.png')
plt.show()
```



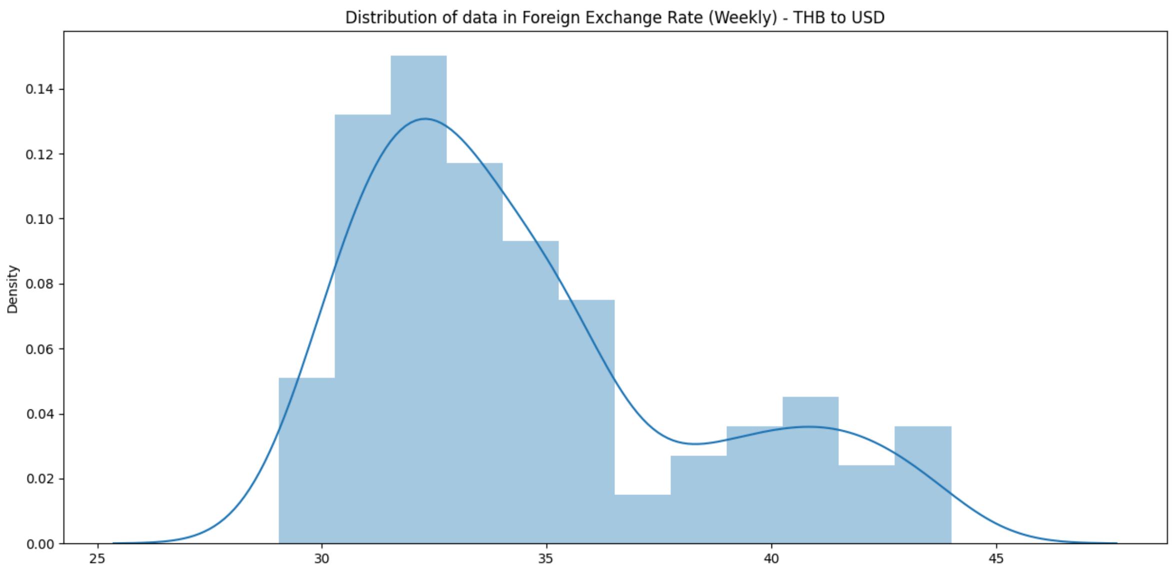

```
In [ ]: plt.rcParams['figure.figsize'] = (15,7)
        sns.scatterplot(x = Forex_month.index , y = Forex_month.Value , color = '
        plt.title('Foreign Exchange Rate (weekly) - THB to USD [Scatter Plot]')
        #plt.savefig('Foreign Exchange Rate (weekly) - THB to USD [Scatter Plot].
        plt.show()
```



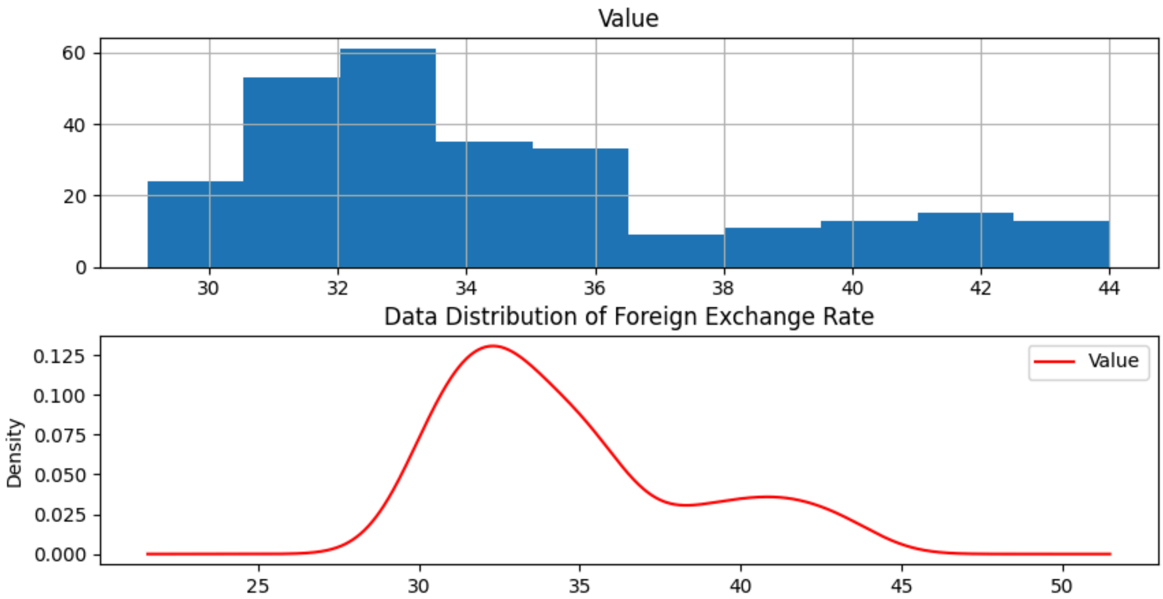

```
In [ ]: sns.barplot(data = Forex_month,x = Forex_month.index , y = Forex_month.Va
        plt.title('Data Spread of Foreign Exchange Rate through Bar Plot')
        #plt.savefig('Data Spread of Foreign Exchange Rate through Bar Plot.png')
        plt.show()
```



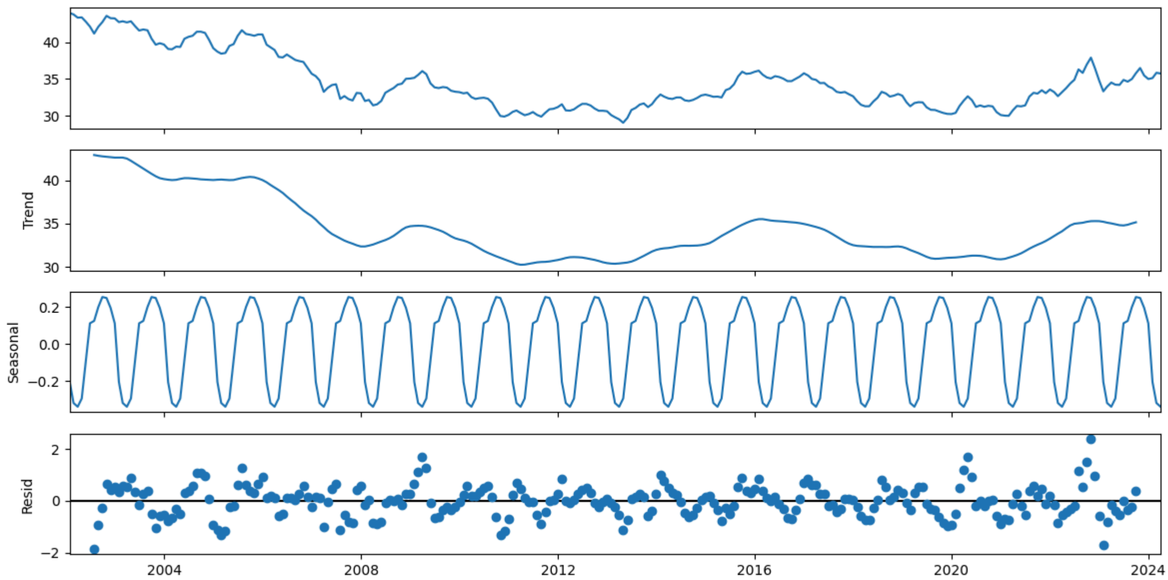
In [ ]:
```
sns.distplot(Forex_month)
plt.title('Distribution of data in Foreign Exchange Rate (Weekly) - THB to
#plt.savefig('Distribution of data in Foreign Exchange Rate (Weekly) - THB
plt.show()
```



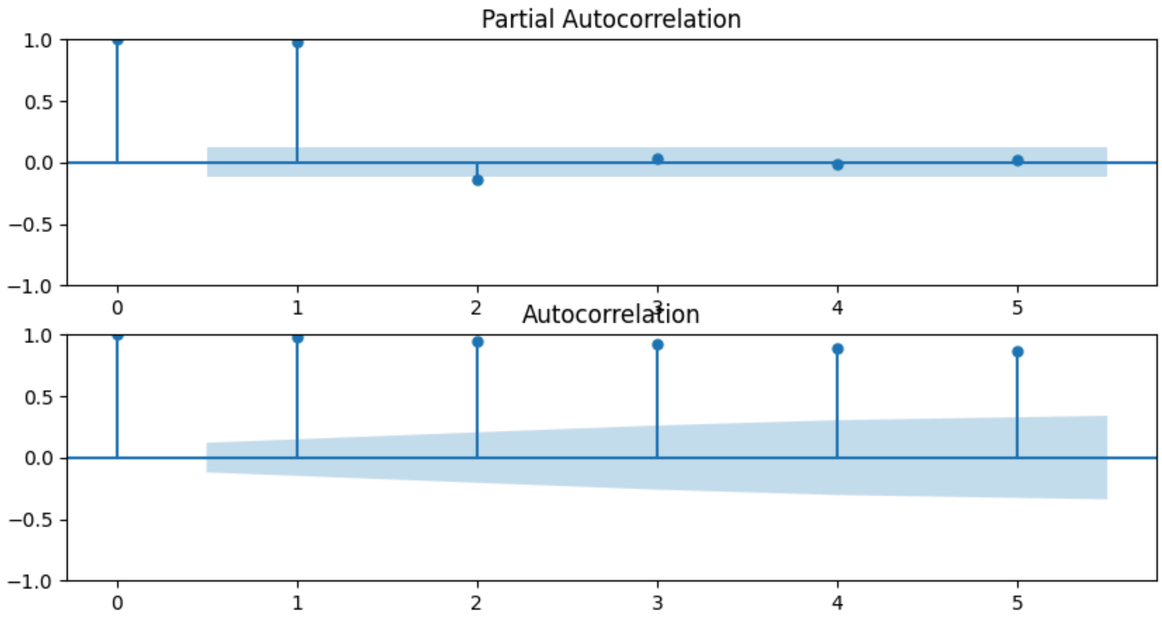
In [ ]:
```
fig , (ax1,ax2) = plt.subplots(nrows = 2 ,ncols = 1,sharex = False , shar
Forex_month.hist(ax = ax1)
Forex_month.plot(kind = 'kde' , ax = ax2,c = 'r')
plt.title('Data Distribution of Foreign Exchange Rate')
#plt.savefig('Data Distribution of Foreign Exchange Rate.png')
plt.show()
```



In [ ]:
```
plt.rcParams['figure.figsize']=(12,6)
decomposition = seasonal_decompose(Forex_month , period = 12 , model = 'ad
decomposition.plot()
#plt.savefig('Discription , trend , seasonal , residuals.png')
plt.show()
```



In [ ]:
```
fig , (ax1,ax2) = plt.subplots(nrows = 2 ,ncols = 1,sharex = False , shar
ax1 = plot_pacf(Forex_month , lags = 5 , ax = ax1)
ax2 = plot_acf(Forex_month , lags = 5 , ax = ax2)
#plt.savefig('Partial Autocorrelation and Autocorrelation.png')
plt.show()
```



Data Tranformation

In [ ]:
```
def adf_check(time_series):
    result = adfuller(time_series , autolag = 'AIC')
    label = pd.Series(result[0:4], index=['Test Statistic','p-value','Num
    for key,value in result[4].items():
        label['Critical Value (%s)'%key] = value
    print(label)
    if result[1] <= 0.05:
        print('Strong evidence against the null hypothesis, hence REJECT 
    else:
        print ('Weak evidence against the null hypothesis, hence ACCEPT n
```

In [ ]:
```
adf_check(Forex_month)
```

```
Test Statistic                  -2.443372
p-value                          0.129867
Number of Lags Used              2.000000
Number of Observations Used    264.000000
Critical Value (1%)             -3.455365
Critical Value (5%)             -2.872551
Critical Value (10%)            -2.572638
dtype: float64
Weak evidence against the null hypothesis, hence ACCEPT null hypothesis an
d the series is Not Stationary
```

In [ ]:
```
Forex1_month = Forex_month.diff().dropna()
print('Count of monthlyly First Difference',Forex1_month.shape[0])
Forex1_month.head()
```

```
Count of monthlyly First Difference 266
```

Out[ ]:

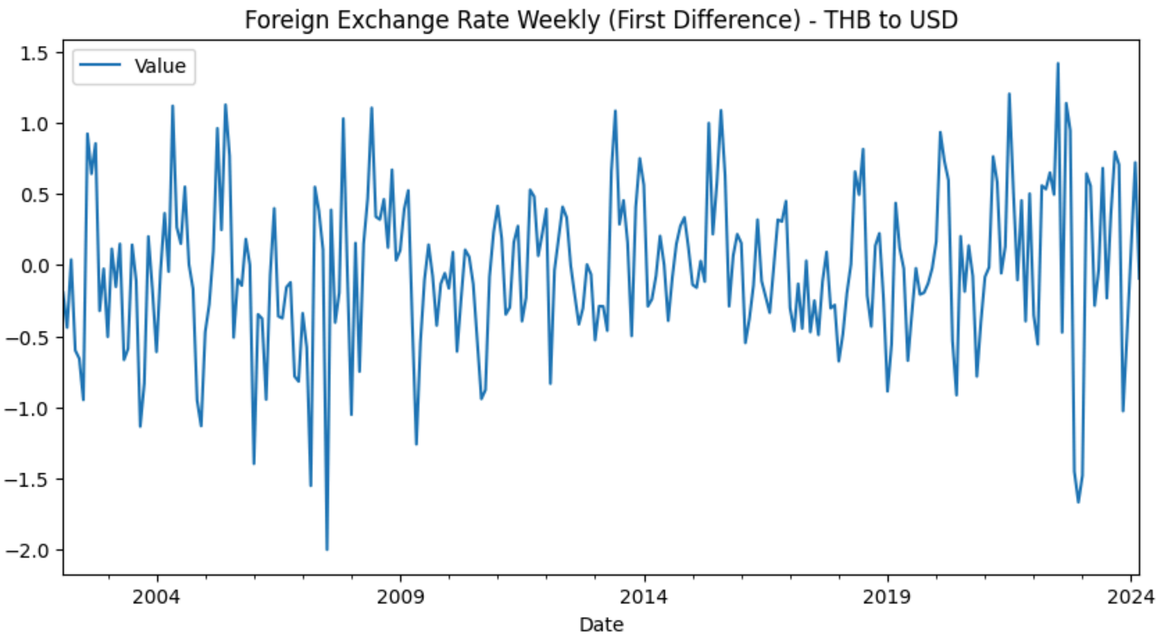
	Value
Date	
2002-02-28	-0.195045
2002-03-31	-0.439500

2002-04-30	0.037273
2002-05-31	-0.599447
2002-06-30	-0.657826

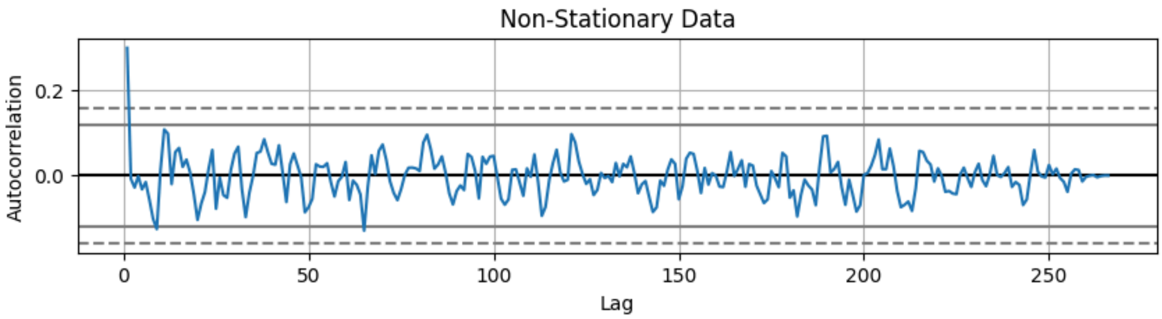
In [ ]:
```
adf_check(Forex1_month)
```

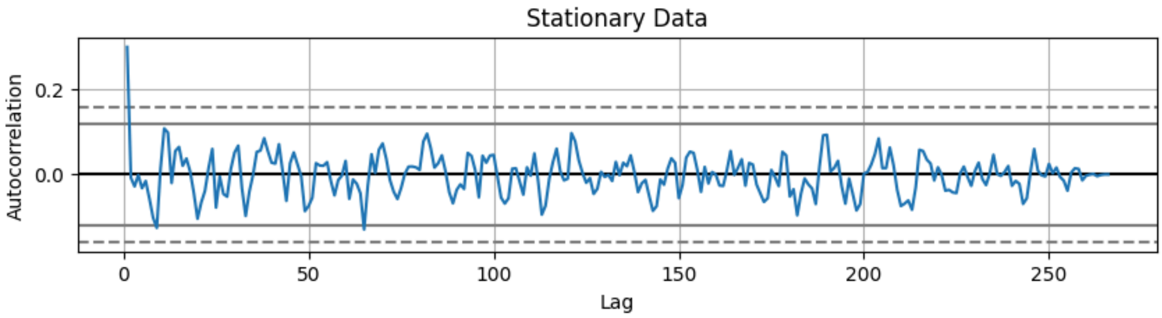
```
Test Statistic                    -1.066318e+01
p-value                            4.354129e-19
Number of Lags Used                1.000000e+00
Number of Observations Used        2.640000e+02
Critical Value (1%)               -3.455365e+00
Critical Value (5%)               -2.872551e+00
Critical Value (10%)              -2.572638e+00
dtype: float64
Strong evidence against the null hypothesis, hence REJECT null hypothesis
and the series is Stationary
```

In [ ]:
```
Forex1_month.plot(figsize = (10,5))
plt.title('Foreign Exchange Rate Weekly (First Difference) - THB to USD')
#plt.savefig('Foreign Exchange Rate Weekly(First Difference) - THB to USD
plt.show()
```



In [ ]:
```
fig , (ax1,ax2) = plt.subplots(nrows = 2 ,ncols = 1,sharex = False , shar
ax1 = autocorrelation_plot(Forex1_month , ax = ax1)
ax1.set_title('Non-Stationary Data')
ax2 = autocorrelation_plot(Forex1_month , ax = ax2)
ax2.set_title('Stationary Data')
plt.subplots_adjust(hspace = 0.5)
#plt.savefig('Stationary data and Non-Stationary data.png')
plt.show()
```





Model Fitting

In [ ]:
```
model = auto_arima(Forex_month , m = 12, d = 1 ,seasonal = False , max_or
```

```
Performing stepwise search to minimize aic
 ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=415.143, Time=0.46 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=436.983, Time=0.05 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=414.134, Time=0.06 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=411.399, Time=0.07 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=435.839, Time=0.02 sec
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=413.351, Time=0.10 sec
 ARIMA(0,1,2)(0,0,0)[0] intercept   : AIC=413.339, Time=0.09 sec
 ARIMA(1,1,2)(0,0,0)[0] intercept   : AIC=413.188, Time=0.33 sec
 ARIMA(0,1,1)(0,0,0)[0]             : AIC=409.953, Time=0.04 sec
 ARIMA(1,1,1)(0,0,0)[0]             : AIC=411.890, Time=0.06 sec
 ARIMA(0,1,2)(0,0,0)[0]             : AIC=411.876, Time=0.06 sec
 ARIMA(1,1,0)(0,0,0)[0]             : AIC=412.609, Time=0.03 sec
 ARIMA(1,1,2)(0,0,0)[0]             : AIC=411.732, Time=0.20 sec

Best model:  ARIMA(0,1,1)(0,0,0)[0]
Total fit time: 1.575 seconds
```

In [ ]:
```
model.summary()
```

Out[ ]:

SARIMAX Results

Dep. Variable:	y	**No. Observations:**	267
Model:	SARIMAX(0, 1, 1)	**Log Likelihood**	-202.977
Date:	Sat, 06 Apr 2024	**AIC**	409.953
Time:	04:37:12	**BIC**	417.120
Sample:	01-31-2002	**HQIC**	412.832
	- 03-31-2024		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ma.L1	0.3266	0.050	6.576	0.000	0.229	0.424
sigma2	0.2692	0.019	13.913	0.000	0.231	0.307

Ljung-Box (L1) (Q):	0.01	**Jarque-Bera (JB):**	10.66
Prob(Q):	0.94	**Prob(JB):**	0.00
Heteroskedasticity (H):	0.82	**Skew:**	-0.17
Prob(H) (two-sided):	0.34	**Kurtosis:**	3.92

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

In [ ]:
```
model = ARIMA(Forex_month , order = (0,1,1))
result = model.fit()
result.summary()
```

Out[ ]:

SARIMAX Results

Dep. Variable:	Value	**No. Observations:**	267
Model:	ARIMA(0, 1, 1)	**Log Likelihood**	-202.977
Date:	Sat, 06 Apr 2024	**AIC**	409.953
Time:	04:37:12	**BIC**	417.120
Sample:	01-31-2002	**HQIC**	412.832
	- 03-31-2024		
Covariance Type:	opg		

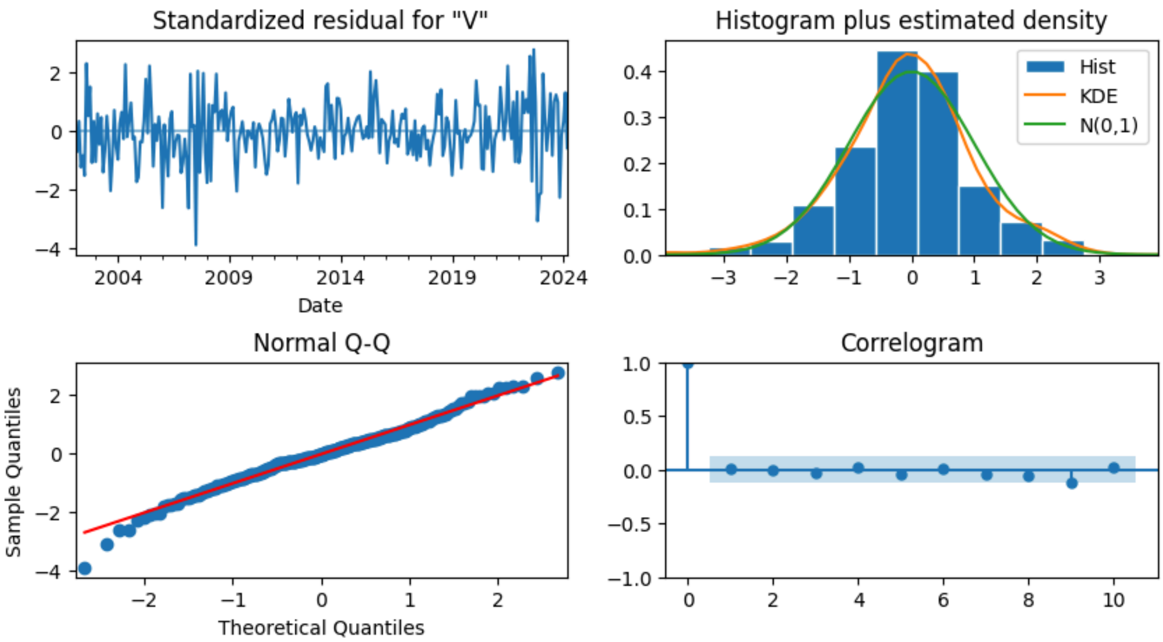
	coef	std err	z	P>|z|	[0.025	0.975]
ma.L1	0.3266	0.050	6.576	0.000	0.229	0.424
sigma2	0.2692	0.019	13.913	0.000	0.231	0.307

Ljung-Box (L1) (Q):	0.01	**Jarque-Bera (JB):**	10.66
Prob(Q):	0.94	**Prob(JB):**	0.00
Heteroskedasticity (H):	0.82	**Skew:**	-0.17
Prob(H) (two-sided):	0.34	**Kurtosis:**	3.92

Warnings:

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

In [ ]:
```
result.plot_diagnostics(figsize = (10,5))
plt.subplots_adjust(hspace = 0.5)
#plt.savefig('Diagnostic plot of best model.png')
plt.show()
```



In [ ]:
```
predictions = result.predict(typ = 'levels')
```

In [ ]:
```
print('Evaluation Result for whole data : ','\n')
print('R2 Score for whole data : {0:.2f} %'.format(100*r2_score(Forex_mon
print('Mean Squared Error : ',mean_squared_error(Forex_month['Value'],pre
print('Mean Absolute Error : ',mean_absolute_error(Forex_month['Value'],p
print('Root Mean Squared Error : ',sqrt(mean_squared_error(Forex_month['V
print('Mean Absolute Percentage Error : {0:.2f} %'.format(100*mean_absolu
```

```
Evaluation Result for whole data :

R2 Score for whole data : 46.28 %

Mean Squared Error :  7.520683218886425

Mean Absolute Error :  0.5571019632377429

Root Mean Squared Error :  2.7423864094774144

Mean Absolute Percentage Error : 1.51 %
```

In [ ]:
```
Final_data = pd.concat([Forex_month,Forex1_month,predictions],axis=1)
Final_data.columns = ['Foreign Exchange Rate (monthly)','Monthly First Di
#Final_data.to_csv('Foreign Exchange Rate with Prediction (THB To USD).cs
Final_data.head()
```

Out[ ]:

	Foreign Exchange Rate (monthly)	Monthly First Difference	Predicted Exchange Rate
Date			

2002-01-31	44.004545	NaN	0.000000
2002-02-28	43.809500	-0.195045	44.004549
2002-03-31	43.370000	-0.439500	43.751939
2002-04-30	43.407273	0.037273	43.246534
2002-05-31	42.807826	-0.599447	43.459710

Model Testing

In [ ]:
```
size = int(len(Forex_month)*0.80)
train , test = Forex_month[0:size]['Value'] , Forex_month[size:(len(Forex_
print('Counts of Train Data : ',train.shape[0])
print('Counts of Test Data : ',test.shape[0])
```

```
Counts of Train Data :  213
Counts of Test Data :  54
```

In [ ]:
```
train_values = [x for x in train]
prediction = []
print('Printing Predictied vs Expected Values....')
print('\n')
for t in range(len(test)):
    model = ARIMA(train_values , order = (0,1,1))
    model_fit = model.fit()
    output = model_fit.forecast()
    pred_out = output[0]
    prediction.append(float(pred_out))
    test_in = test[t]
    train_values.append(test_in)
    print('Predicted = %f , Actual = %f' % (pred_out , test_in))
```

```
Printing Predictied vs Expected Values....


Predicted = 30.506989 , Actual = 30.379783
Predicted = 30.339360 , Actual = 30.250238
Predicted = 30.221891 , Actual = 30.226818
Predicted = 30.228385 , Actual = 30.390435
Predicted = 30.441992 , Actual = 31.322250
Predicted = 31.604082 , Actual = 32.052727
Predicted = 32.197451 , Actual = 32.647727
Predicted = 32.795013 , Actual = 32.115714
Predicted = 31.896694 , Actual = 31.201818
Predicted = 30.973650 , Actual = 31.403043
Predicted = 31.543891 , Actual = 31.216190
Predicted = 31.109853 , Actual = 31.351818
Predicted = 31.429960 , Actual = 31.272955
Predicted = 31.222325 , Actual = 30.490476
Predicted = 30.251861 , Actual = 30.090870
Predicted = 30.038386 , Actual = 30.007143
```

```
Predicted = 29.996896 , Actual = 29.990000
Predicted = 29.987738 , Actual = 30.751087
Predicted = 31.001524 , Actual = 31.334091
Predicted = 31.443499 , Actual = 31.275714
Predicted = 31.220077 , Actual = 31.405909
Predicted = 31.467405 , Actual = 32.607273
Predicted = 32.989099 , Actual = 33.122955
Predicted = 33.167383 , Actual = 33.016818
Predicted = 32.965994 , Actual = 33.469762
Predicted = 33.639651 , Actual = 33.075000
Predicted = 32.887790 , Actual = 33.575217
Predicted = 33.797231 , Actual = 33.222619
Predicted = 33.042147 , Actual = 32.665500
Predicted = 32.545295 , Actual = 33.222391
Predicted = 33.440154 , Actual = 33.754762
Predicted = 33.856233 , Actual = 34.402727
Predicted = 34.580623 , Actual = 34.897273
Predicted = 35.000703 , Actual = 36.312381
Predicted = 36.745785 , Actual = 35.838913
Predicted = 35.559820 , Actual = 36.974091
Predicted = 37.375799 , Actual = 37.918095
Predicted = 38.075651 , Actual = 36.468636
Predicted = 35.981307 , Actual = 34.802500
Predicted = 34.428489 , Actual = 33.323864
Predicted = 32.953579 , Actual = 33.965000
Predicted = 34.291172 , Actual = 34.519130
Predicted = 34.592660 , Actual = 34.233750
Predicted = 34.115300 , Actual = 34.201957
Predicted = 34.230516 , Actual = 34.881591
Predicted = 35.097238 , Actual = 34.648333
Predicted = 34.501679 , Actual = 35.005217
Predicted = 35.167627 , Actual = 35.799524
Predicted = 36.006535 , Actual = 36.503409
Predicted = 36.668057 , Actual = 35.477500
Predicted = 35.089194 , Actual = 35.004286
Predicted = 34.977003 , Actual = 35.133043
Predicted = 35.184254 , Actual = 35.852381
Predicted = 36.072176 , Actual = 35.758067
```

In [ ]:

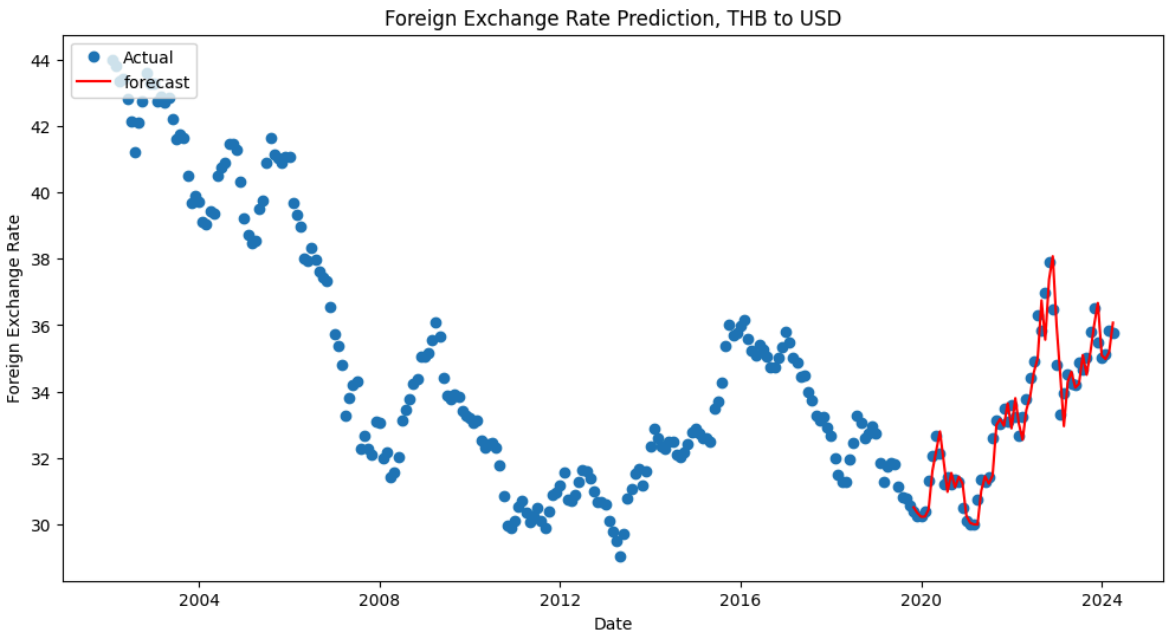
```
print('Evaluation Result for Test data : ','\n')
print('R2 Score for Test data : {0:.2f} %'.format(100*r2_score(test,predi
print('Mean Squared Error : ',mean_squared_error(test,prediction),'\n')
print('Mean Absolute Error : ',mean_absolute_error(test,prediction),'\n')
print('Root Mean Squared Error : ',sqrt(mean_squared_error(test,predictio
print('Mean Absolute Percentage Error : {0:.2f} %'.format(100*mean_absolu
```

```
Evaluation Result for Test data :

R2 Score for Test data : 90.64 %

Mean Squared Error :  0.4155636210391755

Mean Absolute Error :  0.5177040196061622

Root Mean Squared Error :  0.6446422426735433

Mean Absolute Percentage Error : 1.53 %
```

In [ ]:

```
predictions_df = pd.Series(prediction, index = test.index)
```

In [ ]:
```
plt.rcParams['figure.figsize'] = (12,6)
fig, ax = plt.subplots()
ax.set(title='Foreign Exchange Rate Prediction, THB to USD', xlabel='Date
ax.plot(Forex_month, 'o', label='Actual')
ax.plot(predictions_df, 'r', label='forecast')
legend = ax.legend(loc='upper left')
legend.get_frame().set_facecolor('w')
#plt.savefig('Foreign Exchange Rate Prediction - THB to USD.png')
```



Policy Rate

In [ ]:
```
Pr = pd.read_csv('Policy_rate_data.csv')
Pr.head()
```

Out[ ]:

	Date	Policy rate
0	29/2/2024	2.5
1	28/2/2024	2.5
2	27/2/2024	2.5
3	26/2/2024	2.5
4	25/2/2024	2.5

In [ ]:
```
Pr.shape
```

Out[ ]:
```
(6968, 2)
```

In [ ]:
```
Pr.isnull().sum()
```

Out[ ]:
```
Date           0
Policy rate    0
dtype: int64
```

In [ ]:
```
Pr.duplicated().sum()
```

Out[ ]: 0

In [ ]:
```
Pr.dtypes
```

Out[ ]:
```
Date              object
Policy rate      float64
dtype: object
```

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

Out[ ]:

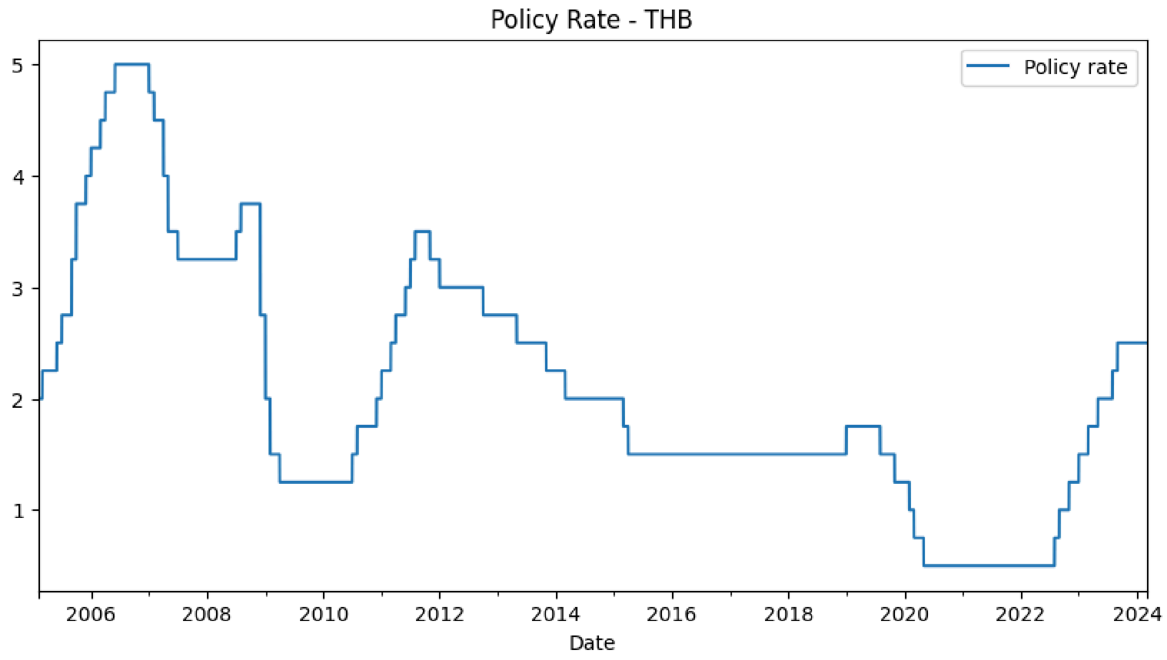
	Policy rate
count	6968.000000
mean	2.092028
std	1.129977
min	0.500000
25%	1.500000
50%	1.750000
75%	2.750000
max	5.000000

Data Processing

In [ ]:
```
Pr['Date'] = pd.to_datetime(Pr['Date'])
```

In [ ]:
```
Pr.set_index('Date',inplace = True)
```

In [ ]:
```
Pr.plot(figsize = (10,5))
plt.title('Policy Rate - THB')
#plt.savefig('Policy Rate - THB to USD.png')
plt.show()
```



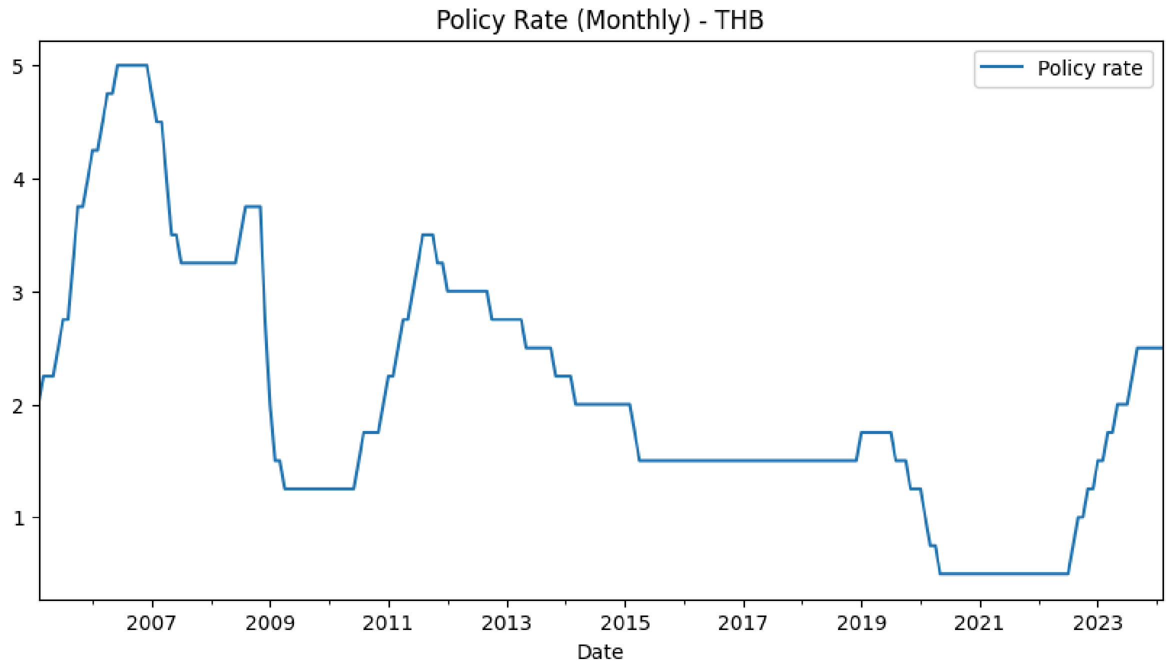
In [ ]:
```
Pr_month = Pr.resample('M').mean()
print('Count of The Monthly Data Frame : ',Pr_month.shape[0])
Pr_month.head()
```

```
Count of The Monthly Data Frame :  229
```

Out[ ]:

Date	Policy rate
2005-02-28	2.00
2005-03-31	2.25
2005-04-30	2.25
2005-05-31	2.25
2005-06-30	2.50

In [ ]:
```
Pr_month.plot(figsize = (10,5))
plt.title('Policy Rate (Monthly) - THB')
#lt.savefig('Policy Rate (Monthly) - THB to USD')
plt.show()
```



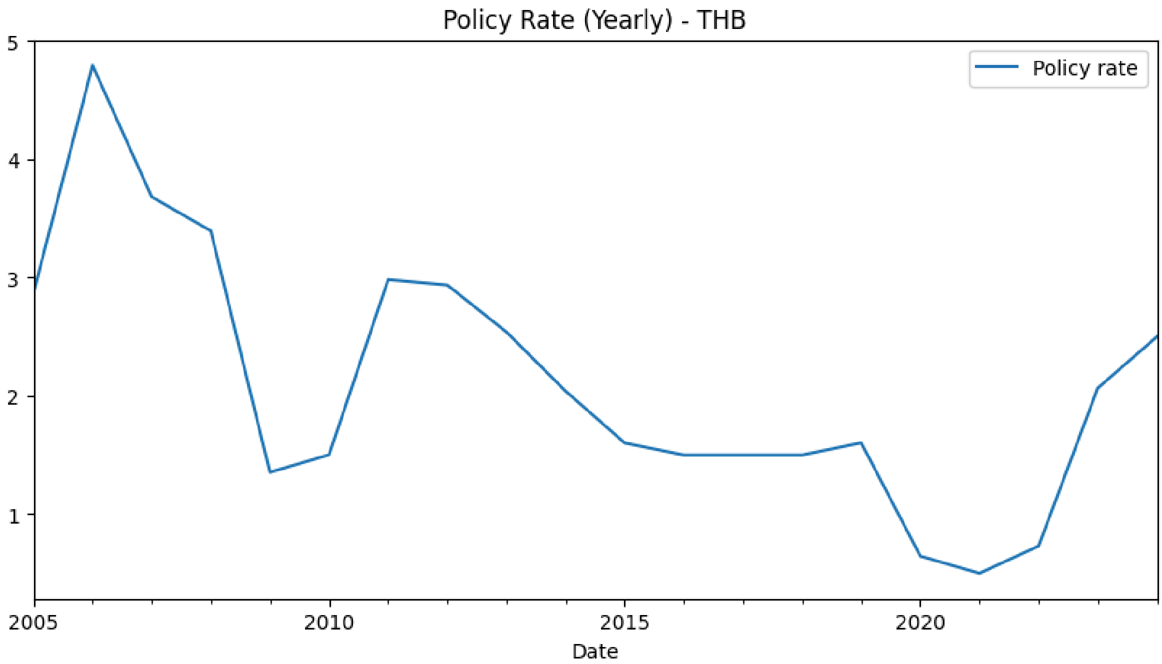
In [ ]:
```
Pr_year = Pr.resample('Y').mean()
print('Count of The Yearly Data Frame : ',Pr_year.shape[0])
Pr_year.head()
```

```
Count of The Yearly Data Frame :  20
```

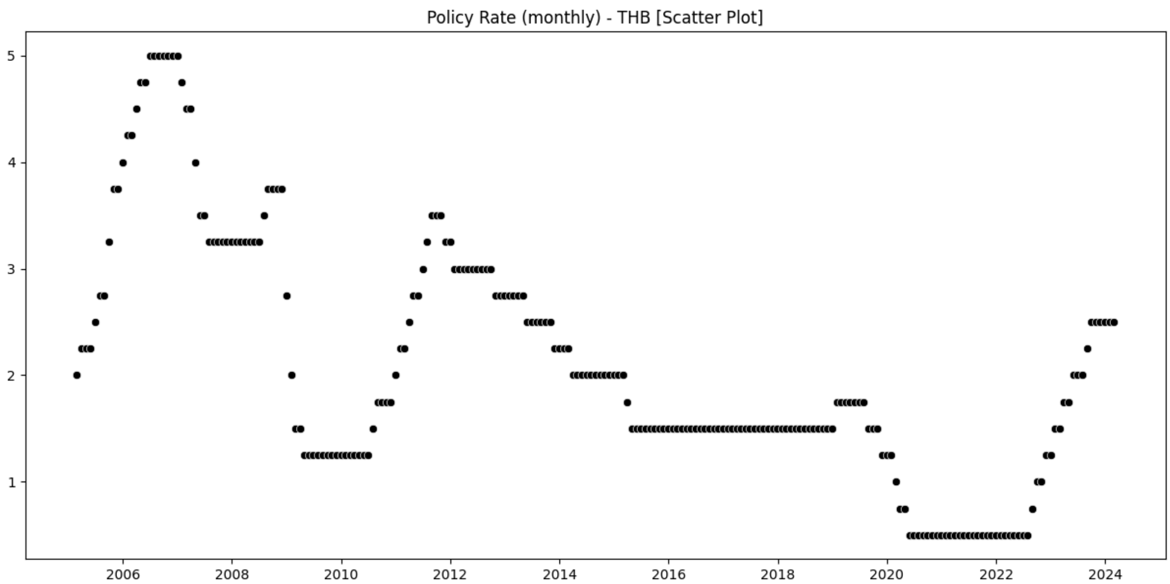
Out[ ]:

Date	Policy rate
2005-12-31	2.870509
2006-12-31	4.794521
2007-12-31	3.682877
2008-12-31	3.395492
2009-12-31	1.354110

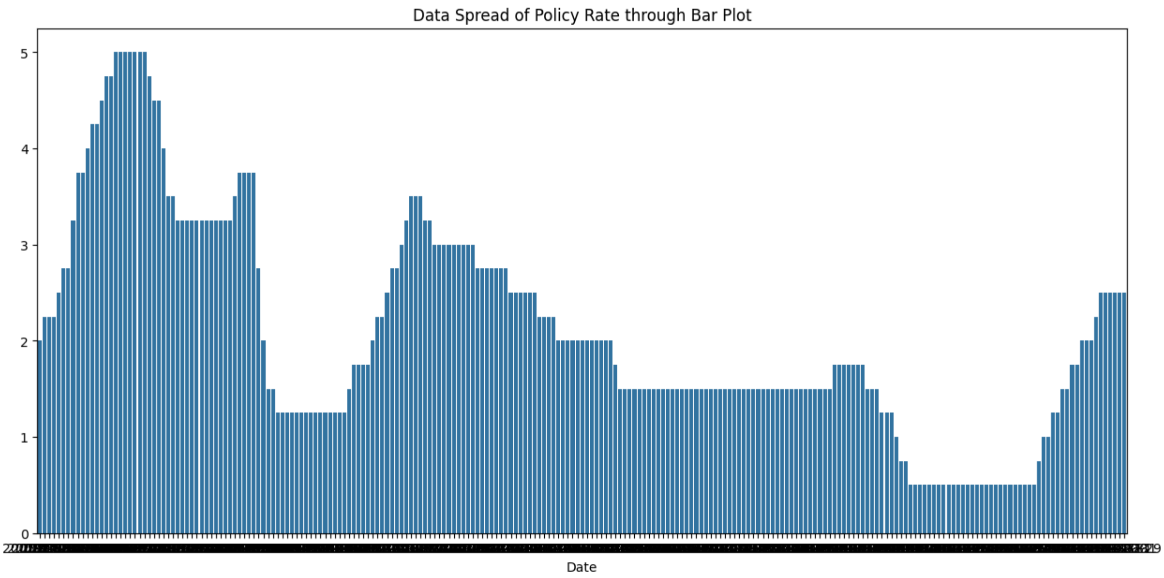
In [ ]:
```
Pr_year.plot(figsize = (10,5))
plt.title('Policy Rate (Yearly) - THB')
#plt.savefig('Policy Rate (Yearly) - THB to USD.png')
plt.show()
```



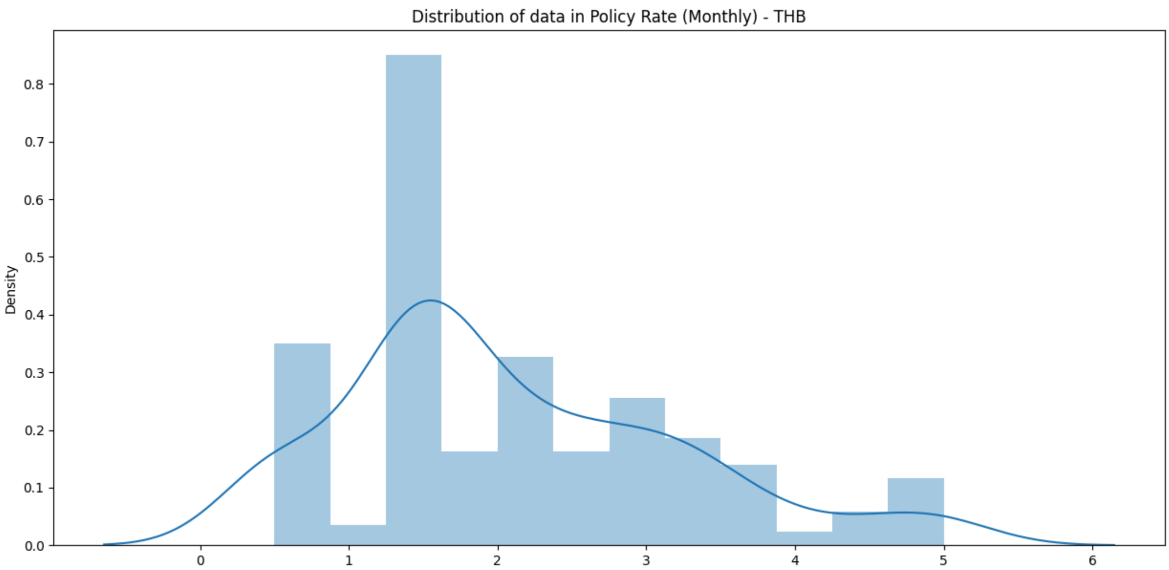
In [ ]:
```
plt.rcParams['figure.figsize'] = (15,7)
sns.scatterplot(x = Pr_month.index.to_numpy().ravel() , y = Pr_month.valu
plt.title('Policy Rate (monthly) - THB [Scatter Plot]')
#plt.savefig('Policy Rate (monthly) - THB to USD [Scatter Plot].png')
plt.show()
```



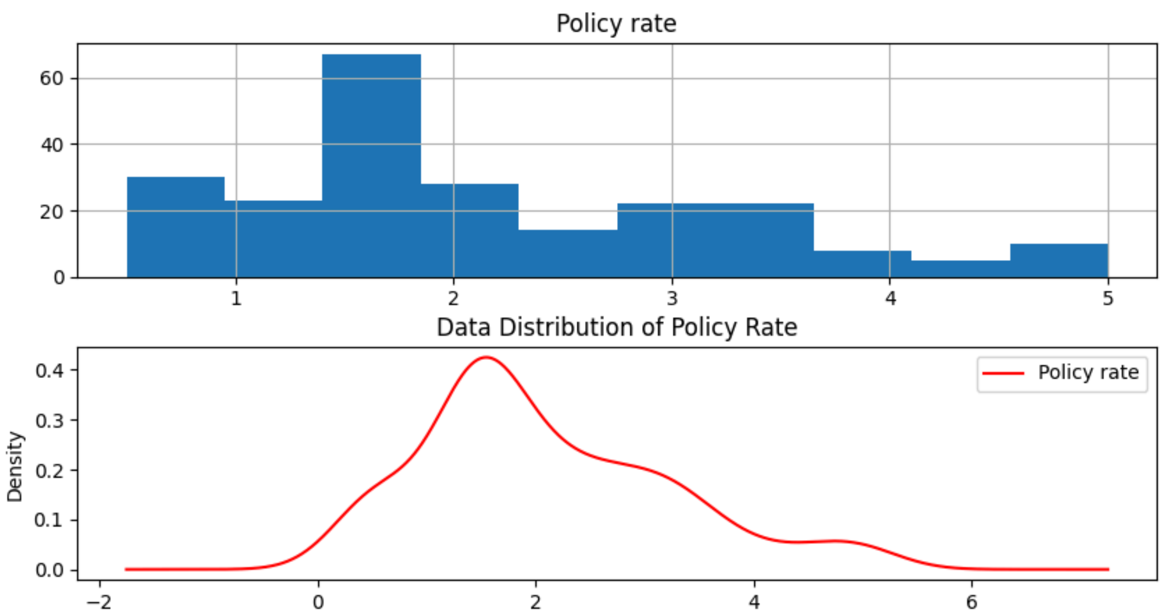
In [ ]:
```
sns.barplot(data = Pr_month,x = Pr_month.index , y = Pr_month.values.rave
plt.title('Data Spread of Policy Rate through Bar Plot')
#plt.savefig('Data Spread of Policy Rate through Bar Plot.png')
plt.show()
```



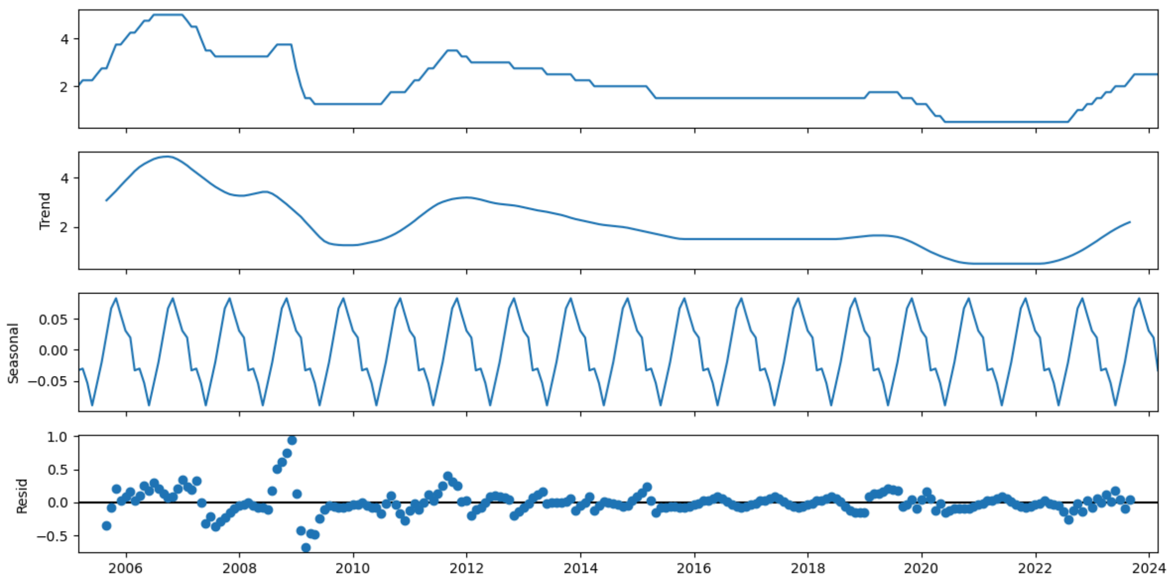
In [ ]:
```
sns.distplot(Pr_month)
plt.title('Distribution of data in Policy Rate (Monthly) - THB')
#plt.savefig('Distribution of data in Policy Rate (Monthly) - THB to USD.p
plt.show()
```



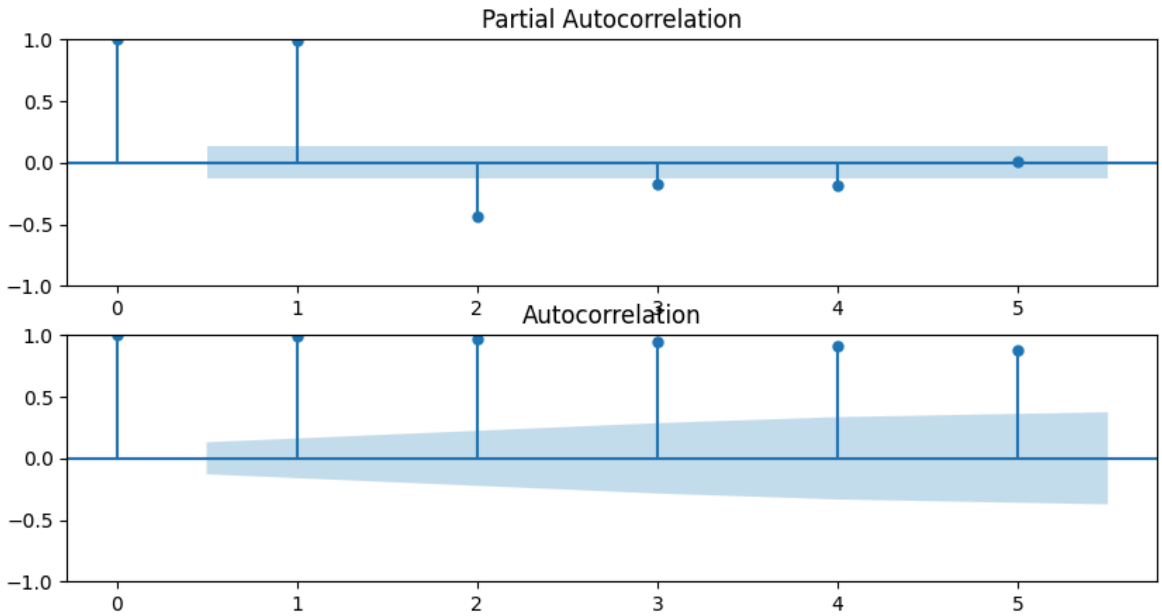
In [ ]:
```
fig , (ax1,ax2) = plt.subplots(nrows = 2 ,ncols = 1,sharex = False , share
Pr_month.hist(ax = ax1)
Pr_month.plot(kind = 'kde' , ax = ax2,c = 'r')
plt.title('Data Distribution of Policy Rate')
#plt.savefig('Data Distribution of Policy Rate.png')
plt.show()
```



In [ ]:
```
plt.rcParams['figure.figsize']=(12,6)
decomposition = seasonal_decompose(Pr_month , period = 12 , model = 'addi
decomposition.plot()
#plt.savefig('Discription , trend , seasonal , residuals.png')
plt.show()
```



In [ ]:
```
fig , (ax1,ax2) = plt.subplots(nrows = 2 ,ncols = 1,sharex = False , shar
ax1 = plot_pacf(Pr_month , lags = 5 , ax = ax1)
ax2 = plot_acf(Pr_month , lags = 5 , ax = ax2)
#plt.savefig('Partial Autocorrelation and Autocorrelation.png')
plt.show()
```



Data Tranformaion

In [ ]:
```
def adf_check(time_series):
    result = adfuller(time_series , autolag = 'AIC')
    label = pd.Series(result[0:4], index=['Test Statistic','p-value','Num
    for key,value in result[4].items():
        label['Critical Value (%s)'%key] = value
    print(label)
    if result[1] <= 0.05:
        print('Strong evidence against the null hypothesis, hence REJECT n
    else:
        print ('Weak evidence against the null hypothesis, hence ACCEPT nu
```

In [ ]:
```
adf_check(Pr_month)
```

```
Test Statistic                 -2.347868
p-value                         0.156994
Number of Lags Used             6.000000
Number of Observations Used   222.000000
Critical Value (1%)            -3.460154
Critical Value (5%)            -2.874649
Critical Value (10%)           -2.573757
dtype: float64
Weak evidence against the null hypothesis, hence ACCEPT null hypothesis an
d the series is Not Stationary
```

In [ ]:
```
adf_check(Pr_month)
```


```
Test Statistic                 -2.347868
p-value                         0.156994
Number of Lags Used             6.000000
Number of Observations Used   222.000000
Critical Value (1%)            -3.460154
Critical Value (5%)            -2.874649
Critical Value (10%)           -2.573757
dtype: float64
Weak evidence against the null hypothesis, hence ACCEPT null hypothesis an
d the series is Not Stationary
```

In [ ]:
```
Pr1_month = Pr_month.diff().dropna()
print('Count of monthly Frist Diference', Pr1_month.shape[0])
Pr1_month.head()
```

```
Count of monthly Frist Diference 228
```

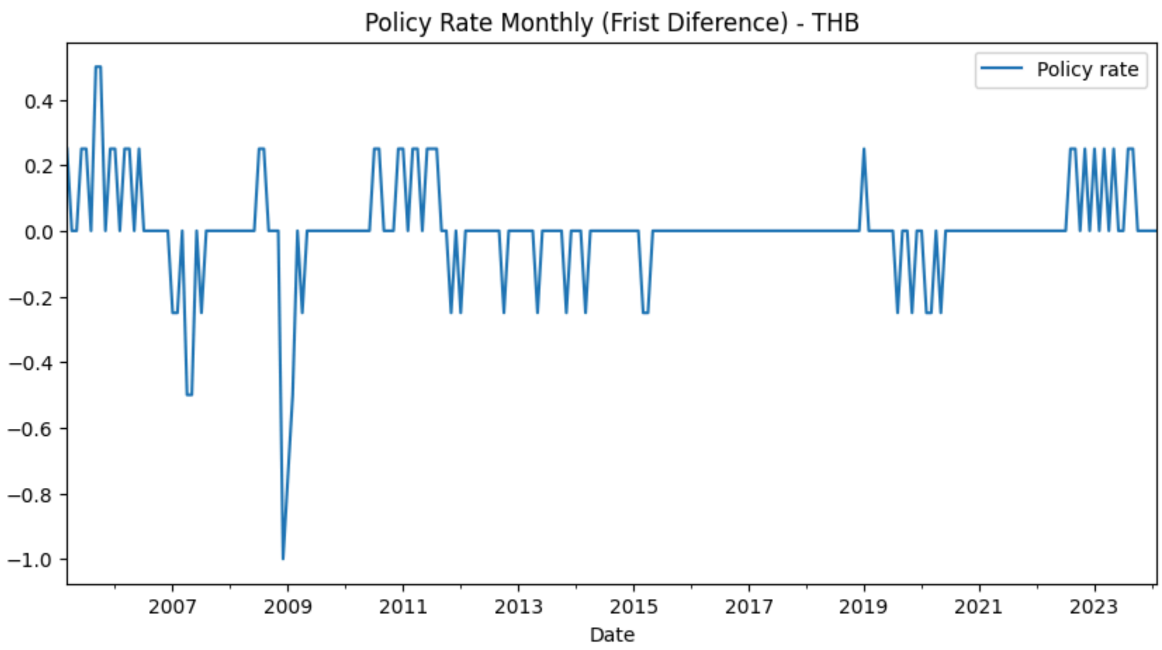
Out[ ]:

Date	Policy rate
2005-03-31	0.25
2005-04-30	0.00
2005-05-31	0.00
2005-06-30	0.25
2005-07-31	0.25

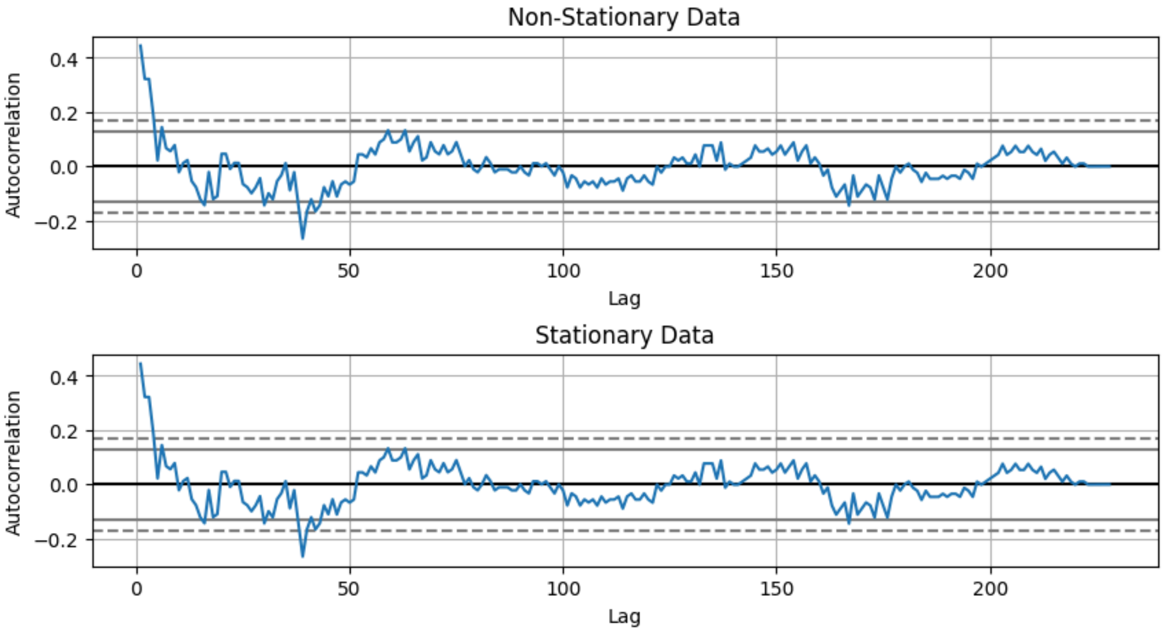
In [ ]:
```
adf_check(Pr1_month)
```

```
Test Statistic                  -5.696197e+00
p-value                          7.863958e-07
Number of Lags Used              4.000000e+00
Number of Observations Used      2.230000e+02
Critical Value (1%)             -3.460019e+00
Critical Value (5%)             -2.874590e+00
Critical Value (10%)            -2.573725e+00
dtype: float64
Strong evidence against the null hypothesis, hence REJECT null hypothesis
and the series is Stationary
```

In [ ]:
```
Pr1_month.plot(figsize = (10,5))
plt.title('Policy Rate Monthly (Frist Diference) - THB')
#plt.savefig('Policy Rate Monthly (Frist Diference) - THB.png')
plt.show()
```



In [ ]:
```
fig , (ax1,ax2) = plt.subplots(nrows = 2 ,ncols = 1,sharex = False , shar
ax1 = autocorrelation_plot(Pr1_month , ax = ax1)
ax1.set_title('Non-Stationary Data')
ax2 = autocorrelation_plot(Pr1_month , ax = ax2)
ax2.set_title('Stationary Data')
plt.subplots_adjust(hspace = 0.5)
#plt.savefig('Stationary data and Non-Stationary data.png')
plt.show()
```



Model Fitting

In [ ]:
```
model = auto_arima(Pr_month, m = 12, d = 1, seasonal = False, max_order =
```

```
Performing stepwise search to minimize aic
 ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=-244.445, Time=0.45 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=-193.093, Time=0.06 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=-241.663, Time=0.06 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=-228.356, Time=0.07 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=-195.048, Time=0.03 sec
 ARIMA(1,1,2)(0,0,0)[0] intercept   : AIC=-246.162, Time=0.24 sec
 ARIMA(0,1,2)(0,0,0)[0] intercept   : AIC=-232.265, Time=0.14 sec
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=-248.157, Time=0.13 sec
 ARIMA(2,1,1)(0,0,0)[0] intercept   : AIC=-246.160, Time=0.20 sec
 ARIMA(2,1,0)(0,0,0)[0] intercept   : AIC=-245.093, Time=0.12 sec
 ARIMA(1,1,1)(0,0,0)[0]             : AIC=-250.126, Time=0.07 sec
 ARIMA(0,1,1)(0,0,0)[0]             : AIC=-230.317, Time=0.04 sec
 ARIMA(1,1,0)(0,0,0)[0]             : AIC=-243.630, Time=0.03 sec
 ARIMA(2,1,1)(0,0,0)[0]             : AIC=-248.129, Time=0.12 sec
 ARIMA(1,1,2)(0,0,0)[0]             : AIC=-248.131, Time=0.14 sec
 ARIMA(0,1,2)(0,0,0)[0]             : AIC=-234.227, Time=0.09 sec
 ARIMA(2,1,0)(0,0,0)[0]             : AIC=-247.062, Time=0.06 sec
 ARIMA(2,1,2)(0,0,0)[0]             : AIC=-246.419, Time=0.24 sec

Best model:  ARIMA(1,1,1)(0,0,0)[0]
Total fit time: 2.313 seconds
```

In [ ]:
```
model.summary()
```

Out[ ]:

SARIMAX Results

Dep. Variable:	y	**No. Observations:**	229
Model:	SARIMAX(1, 1, 1)	**Log Likelihood**	128.063
Date:	Sat, 06 Apr 2024	**AIC**	-250.126
Time:	04:37:24	**BIC**	-239.838
Sample:	02-28-2005	**HQIC**	-245.975
	- 02-29-2024		
Covariance Type:	opg		

	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
ar.L1	0.7627	0.064	11.979	0.000	0.638	0.887
ma.L1	-0.4105	0.083	-4.933	0.000	-0.574	-0.247
sigma2	0.0190	0.001	27.787	0.000	0.018	0.020

Ljung-Box (L1) (Q):	0.00	**Jarque-Bera (JB):**	1608.90
Prob(Q):	0.98	**Prob(JB):**	0.00
Heteroskedasticity (H):	0.26	**Skew:**	-1.75
Prob(H) (two-sided):	0.00	**Kurtosis:**	15.53

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

In [ ]:

```
model = ARIMA(Pr_month, order = (1,1,1))
result = model.fit()
result.summary()
```

Out[ ]:

SARIMAX Results

Dep. Variable:	Policy rate	**No. Observations:**	229
Model:	ARIMA(1, 1, 1)	**Log Likelihood**	128.063
Date:	Sat, 06 Apr 2024	**AIC**	-250.126
Time:	04:37:24	**BIC**	-239.838
Sample:	02-28-2005	**HQIC**	-245.975
	- 02-29-2024		
Covariance Type:	opg		

	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**

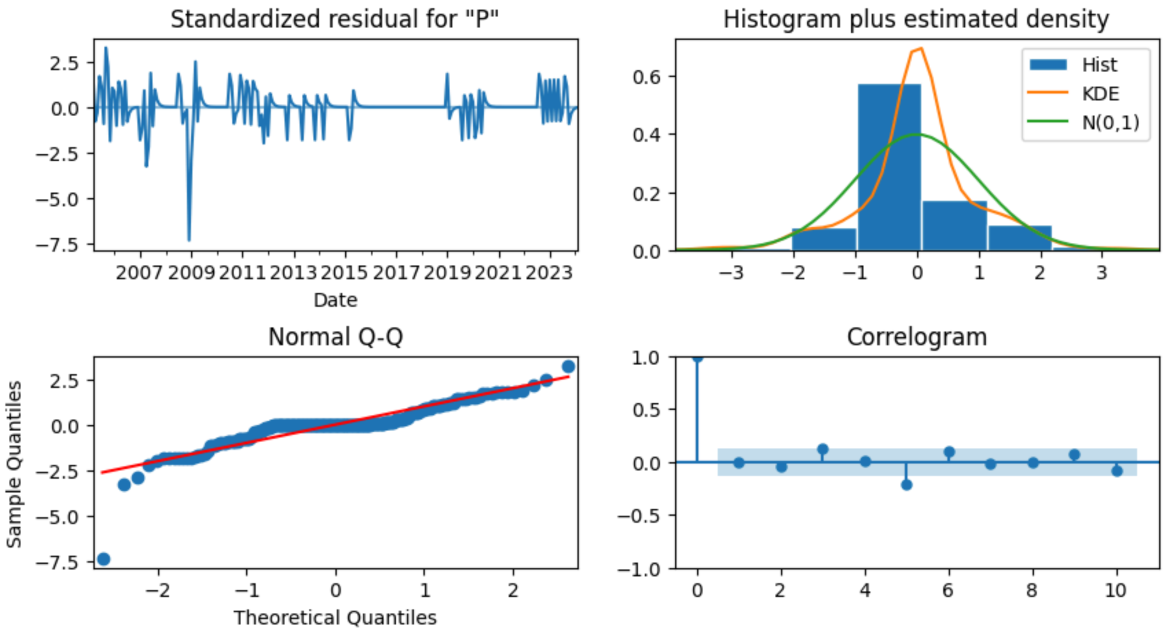
ar.L1	0.7627	0.064	11.979	0.000	0.638	0.887
ma.L1	-0.4105	0.083	-4.933	0.000	-0.574	-0.247
sigma2	0.0190	0.001	27.787	0.000	0.018	0.020

Ljung-Box (L1) (Q):	0.00	Jarque-Bera (JB):	1608.90
Prob(Q):	0.98	Prob(JB):	0.00
Heteroskedasticity (H):	0.26	Skew:	-1.75
Prob(H) (two-sided):	0.00	Kurtosis:	15.53

Warnings:

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

In [ ]:
```
result.plot_diagnostics(figsize = (10,5))
plt.subplots_adjust(hspace = 0.5)
#plt.savefig('Diagnostic Policy Rate plot of best model.png')
plt.show()
```



In [ ]:
```
predictions = result.predict(typ = 'levels')
```

In [ ]:
```
print('Evaluation Result for whole data : ','\n')
print('R2 Score for whole data : {0:.2f} %'.format(100*r2_score(Pr_month[
print('Mean Squared Error : ',mean_squared_error(Pr_month['Policy rate'],
print('Mean Absolute Error : ',mean_absolute_error(Pr_month['Policy rate'
print('Root Mean Squared Error : ',sqrt(mean_squared_error(Pr_month['Poli
print('Mean Absolute Percentage Error : {0:.2f} %'.format(100*mean_absolu
```

```
Evaluation Result for whole data :

R2 Score for whole data : 97.14 %
```

```
Mean Squared Error :  0.03646305323365609

Mean Absolute Error :  0.08546435912336092

Root Mean Squared Error :  0.19095301315678706

Mean Absolute Percentage Error : 4.31 %
```

In [ ]:
```
Final_data = pd.concat([Pr_month,Pr1_month,predictions],axis=1)
Final_data.columns = ['Policy Rate (monthly)','Monthly First Difference',
#Final_data.to_csv('FPolicy Rate with Prediction (THB To USD).csv')
Final_data.head()
```

Out[ ]:

	Policy Rate (monthly)	Monthly First Difference	Predicted Policy Rate
Date			
2005-02-28	2.00	NaN	0.000000
2005-03-31	2.25	0.25	2.000000
2005-04-30	2.25	0.00	2.361527
2005-05-31	2.25	0.00	2.294082
2005-06-30	2.50	0.25	2.267983

Model Testing

In [ ]:
```
size = int(len(Pr_month)*0.80)
train , test = Pr_month[0:size]['Policy rate'] , Pr_month[size:(len(Pr_mo
print('Counts of Train Data : ',train.shape[0])
print('Counts of Train Data : ',test.shape[0])
```

```
Counts of Train Data :  183
Counts of Train Data :  46
```

In [ ]:
```
train_values = [x for x in train]
prediction = []
print('Printing Predictied vs Expected Values....')
print('\n')
for t in range(len(test)):
    model = ARIMA(train_values , order = (1,1,1))
    model_fit = model.fit()
    output = model_fit.forecast()
    pred_out = output[0]
    prediction.append(float(pred_out))
    test_in = test[t]
    train_values.append(test_in)
    print('Predicted = %f , Actual = %f' % (pred_out , test_in))
```

```
Printing Predictied vs Expected Values....


Predicted = 0.706916 , Actual = 0.500000
Predicted = 0.388000 , Actual = 0.500000
Predicted = 0.460768 , Actual = 0.500000
Predicted = 0.486411 , Actual = 0.500000
Predicted = 0.495275 , Actual = 0.500000
```

```
Predicted = 0.498352 , Actual = 0.500000
Predicted = 0.499424 , Actual = 0.500000
Predicted = 0.499799 , Actual = 0.500000
Predicted = 0.499930 , Actual = 0.500000
Predicted = 0.499975 , Actual = 0.500000
Predicted = 0.499991 , Actual = 0.500000
Predicted = 0.499997 , Actual = 0.500000
Predicted = 0.499999 , Actual = 0.500000
Predicted = 0.500000 , Actual = 0.500000
Predicted = 0.500000 , Actual = 0.500000
Predicted = 0.500000 , Actual = 0.500000
Predicted = 0.500000 , Actual = 0.500000
Predicted = 0.500000 , Actual = 0.500000
Predicted = 0.500000 , Actual = 0.500000
Predicted = 0.500000 , Actual = 0.500000
Predicted = 0.500000 , Actual = 0.500000
Predicted = 0.500000 , Actual = 0.500000
Predicted = 0.500000 , Actual = 0.500000
Predicted = 0.500000 , Actual = 0.500000
Predicted = 0.500000 , Actual = 0.500000
Predicted = 0.500000 , Actual = 0.500000
Predicted = 0.500000 , Actual = 0.500000
Predicted = 0.500000 , Actual = 0.750000
Predicted = 0.844988 , Actual = 1.000000
Predicted = 1.130073 , Actual = 1.000000
Predicted = 1.043665 , Actual = 1.250000
Predicted = 1.361238 , Actual = 1.250000
Predicted = 1.289770 , Actual = 1.500000
Predicted = 1.608411 , Actual = 1.500000
Predicted = 1.540656 , Actual = 1.750000
Predicted = 1.857575 , Actual = 1.750000
Predicted = 1.792054 , Actual = 2.000000
Predicted = 2.107086 , Actual = 2.000000
Predicted = 2.043425 , Actual = 2.000000
Predicted = 2.017511 , Actual = 2.250000
Predicted = 2.345347 , Actual = 2.500000
Predicted = 2.629044 , Actual = 2.500000
Predicted = 2.553052 , Actual = 2.500000
Predicted = 2.521631 , Actual = 2.500000
Predicted = 2.508827 , Actual = 2.500000
Predicted = 2.503614 , Actual = 2.500000
```

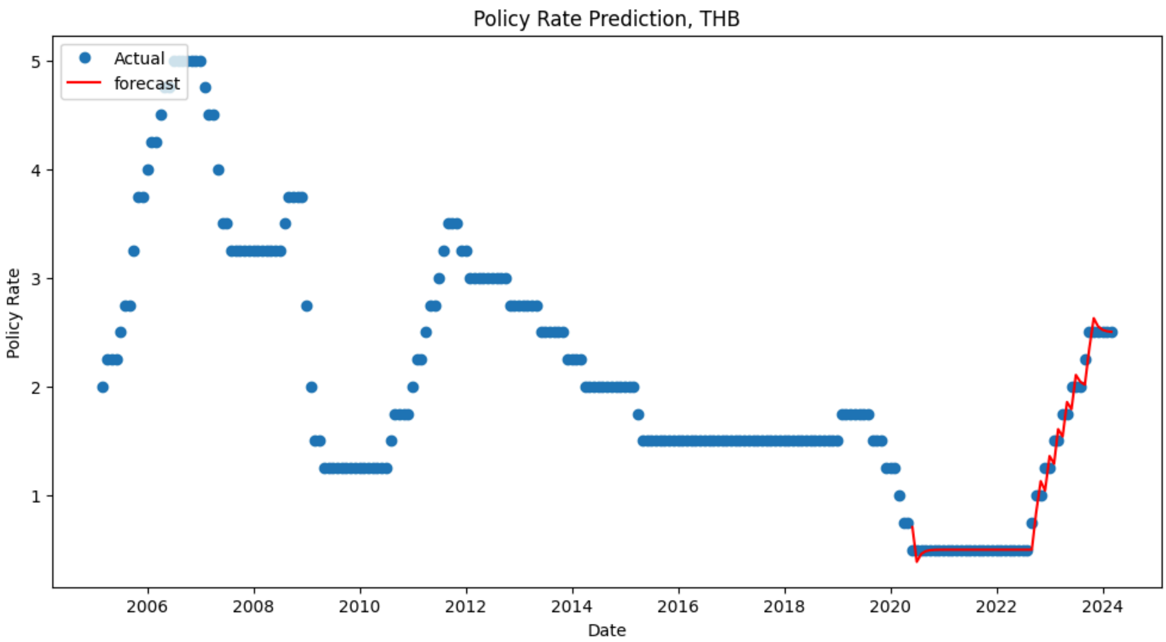
In [ ]:

```
print('Evaluation Result for Test data : ','\n')
print('R2 Score for Test data : {0:.2f} %'.format(100*r2_score(test,predi
print('Mean Squared Error : ',mean_squared_error(test,prediction),'\n')
print('Mean Absolute Error : ',mean_absolute_error(test,prediction),'\n')
print('Root Mean Squared Error : ',sqrt(mean_squared_error(test,predictio
print('Mean Absolute Percentage Error : {0:.2f} %'.format(100*mean_absolu
```

```
Evaluation Result for Test data :

R2 Score for Test data : 98.18 %

Mean Squared Error :  0.010466082828162254

Mean Absolute Error :  0.061499594342554385

Root Mean Squared Error :  0.10230387494206783

Mean Absolute Percentage Error : 5.34 %
```

In [ ]:
```
predictions_df = pd.Series(prediction, index = test.index)
```

In [ ]:
```
plt.rcParams['figure.figsize'] = (12,6)
fig, ax = plt.subplots()
ax.set(title='Policy Rate Prediction, THB', xlabel='Date', ylabel='Policy
ax.plot(Pr_month, 'o', label='Actual')
ax.plot(predictions_df, 'r', label='forecast')
legend = ax.legend(loc='upper left')
legend.get_frame().set_facecolor('w')
#plt.savefig('Foreign Exchange Rate Prediction - THB to USD.png')
```



Merge Data

In [ ]:
```
print(Forex.columns)
print(Pr.columns)
```

```
Index(['Value'], dtype='object')
Index(['Policy rate'], dtype='object')
```

In [ ]:
```
merged_df = Forex_month.merge(Pr_month, how='inner', on='Date')
merged_df.head()
```

Out[ ]:

	Value	Policy rate
Date		
2005-02-28	38.459500	2.00
2005-03-31	38.556522	2.25
2005-04-30	39.515952	2.25
2005-05-31	39.762045	2.25
2005-06-30	40.886818	2.50

In [ ]:
```
merged_df.shape
```

Out[ ]: (229, 2)

In [ ]:
```
merged_df.isnull().sum()
```

Out[ ]:
```
Value          0
Policy rate    0
dtype: int64
```

In [ ]:
```
merged_df.duplicated().sum()
```

Out[ ]: 0

In [ ]:
```
merged_df.dtypes
```

Out[ ]:
```
Value          float64
Policy rate    float64
dtype: object
```

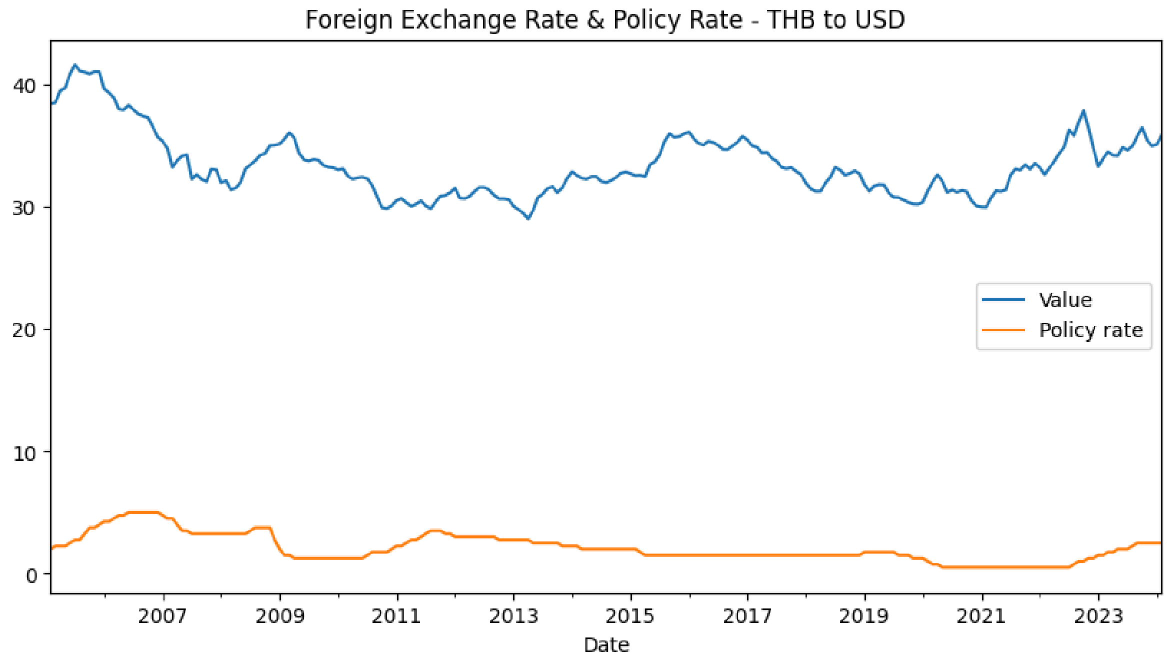
In [ ]:
```
merged_df.describe()
```

Out[ ]:

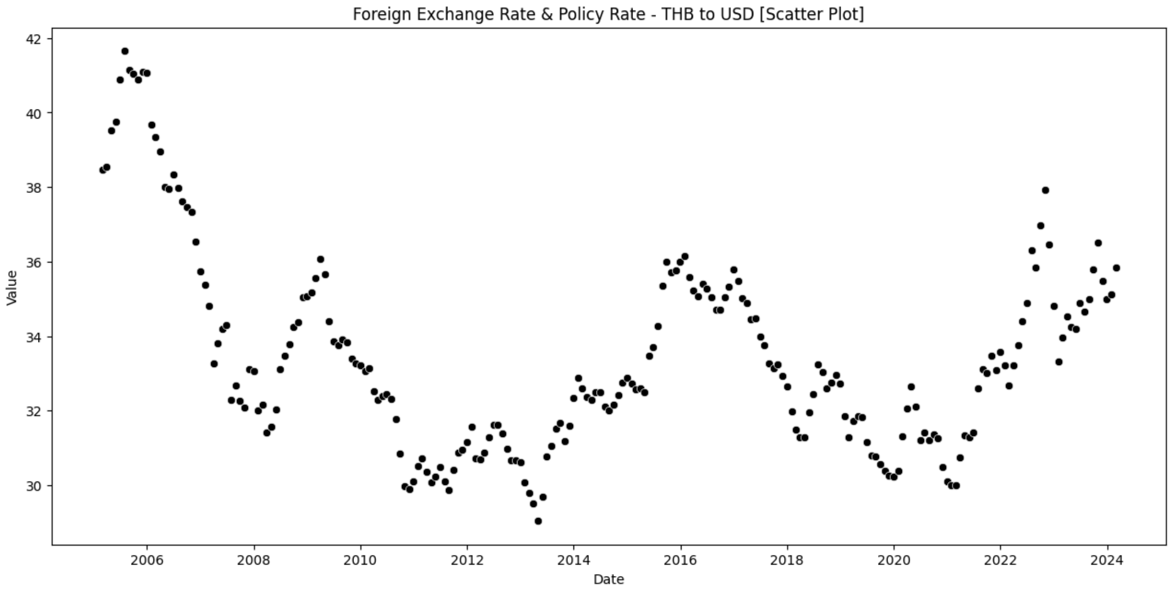
	Value	Policy rate
count	229.000000	229.000000
mean	33.439016	2.091703
std	2.629871	1.132067
min	29.040909	0.500000
25%	31.405909	1.500000
50%	32.964773	1.750000
75%	35.005217	2.750000
max	41.653476	5.000000

Data Processing

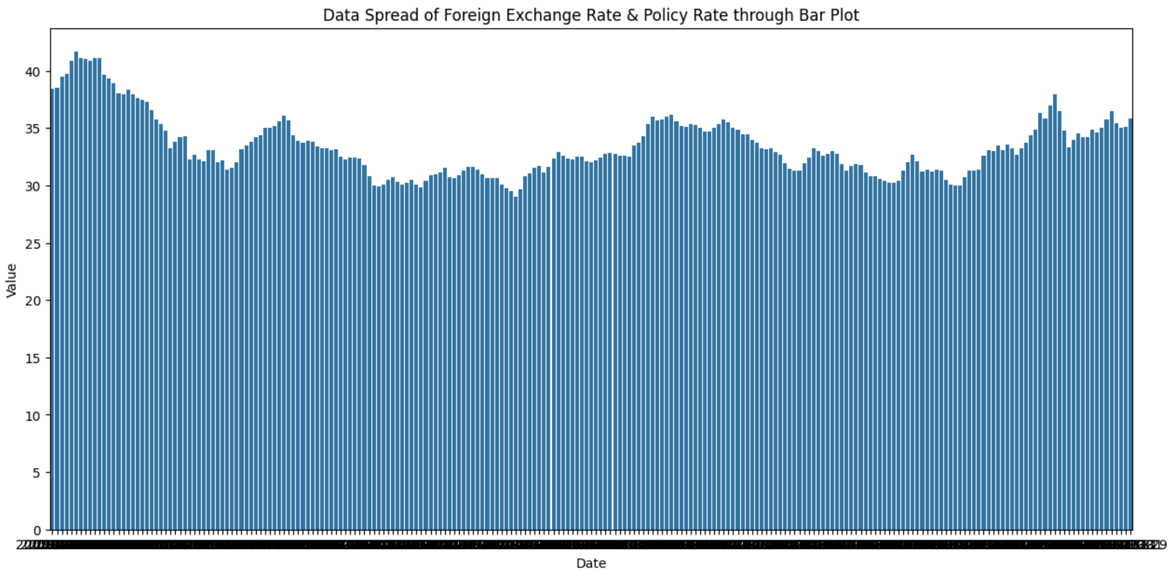
In [ ]:
```
merged_df.plot(figsize = (10,5))
plt.title('Foreign Exchange Rate & Policy Rate - THB to USD')
# plt.savefig('Foreign Exchange Rate & Policy Rate - THB to USD')
plt.show()
```



In [ ]:
```
plt.rcParams['figure.figsize'] = (15,7)
sns.scatterplot(x = merged_df.index , y = merged_df.Value , color = 'blac
plt.title('Foreign Exchange Rate & Policy Rate - THB to USD [Scatter Plot
#plt.savefig('Foreign Exchange Rate & Policy Rate - THB to USD [Scatter P.
plt.show()
```



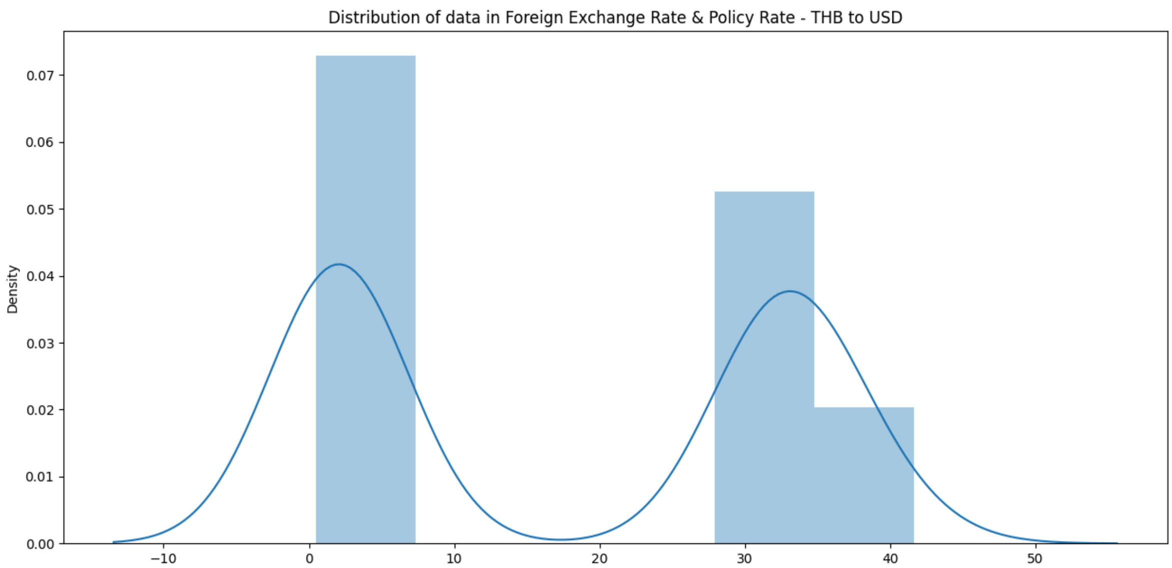
In [ ]:
```
sns.barplot(data = merged_df, x = merged_df.index , y = merged_df.Value)
plt.title('Data Spread of Foreign Exchange Rate & Policy Rate through Bar
#plt.savefig('Data Spread of Foreign Exchange Rate & Policy Rate through I
plt.show()
```



In [ ]:
```
print(merged_df.Value)
```

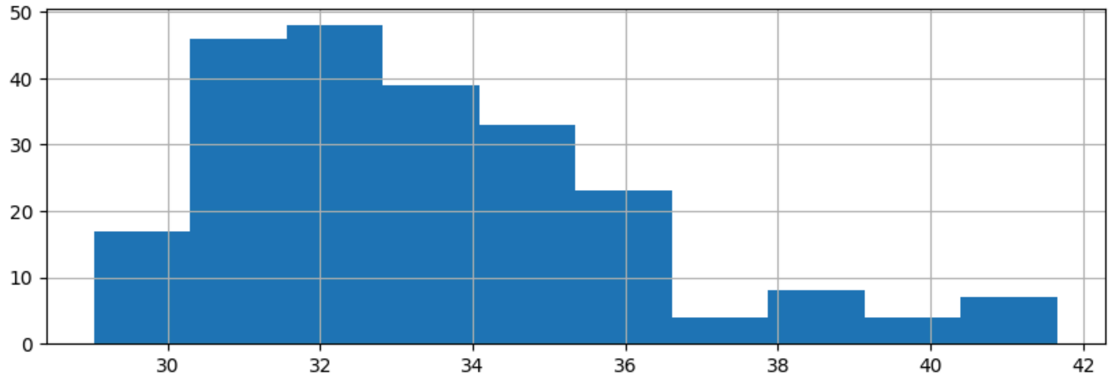
```
Date
2005-02-28    38.459500
2005-03-31    38.556522
2005-04-30    39.515952
2005-05-31    39.762045
2005-06-30    40.886818
                ...
2023-10-31    36.503409
2023-11-30    35.477500
2023-12-31    35.004286
2024-01-31    35.133043
2024-02-29    35.852381
Name: Value, Length: 229, dtype: float64
```

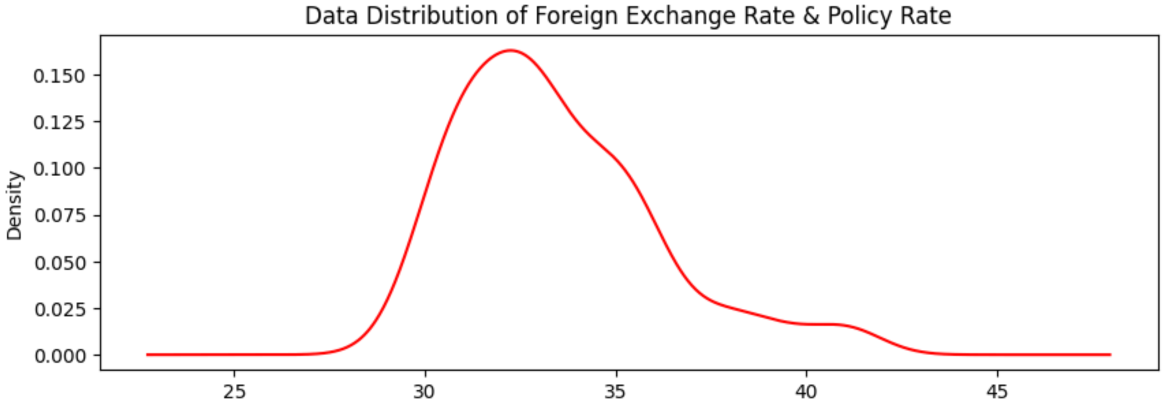
In [ ]:
```
sns.distplot(merged_df)
plt.title('Distribution of data in Foreign Exchange Rate & Policy Rate - T
#plt.savefig('Distribution of data in Foreign Exchange Rate & Policy Rate
plt.show()
```



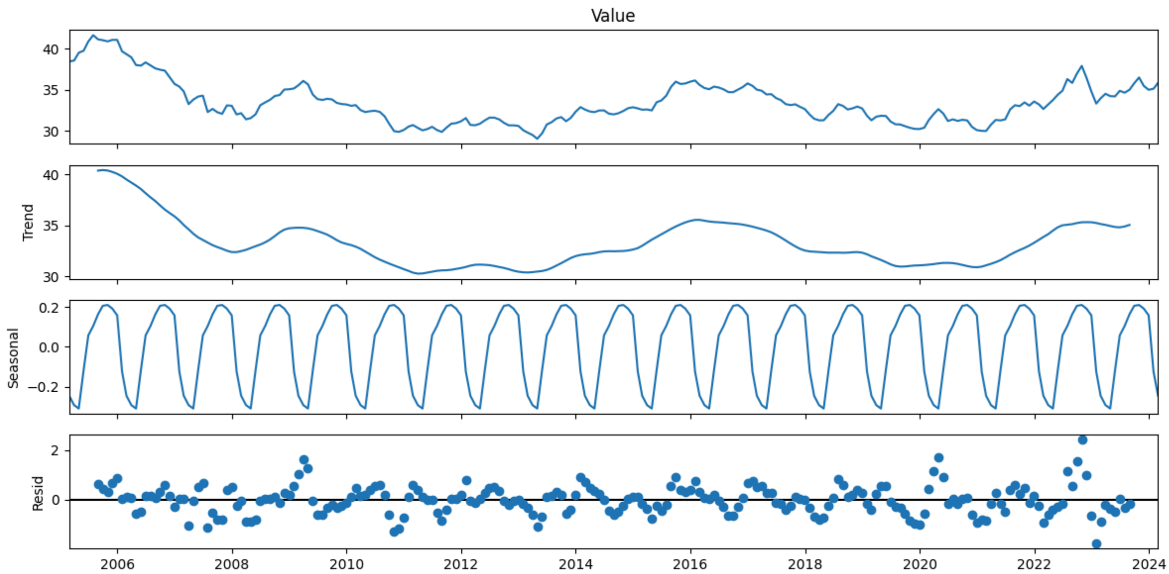
In [ ]:
```
fig , (ax1,ax2) = plt.subplots(nrows = 2, ncols = 1, sharex = False, shar
merged_df.Value.hist(ax = ax1)
merged_df.Value.plot(kind = 'kde' , ax = ax2,c = 'r')
```

```
plt.title('Data Distribution of Foreign Exchange Rate & Policy Rate')
#plt.savefig('Data Distribution of Foreign Exchange Rate & Policy Rate.pn
plt.show()
```

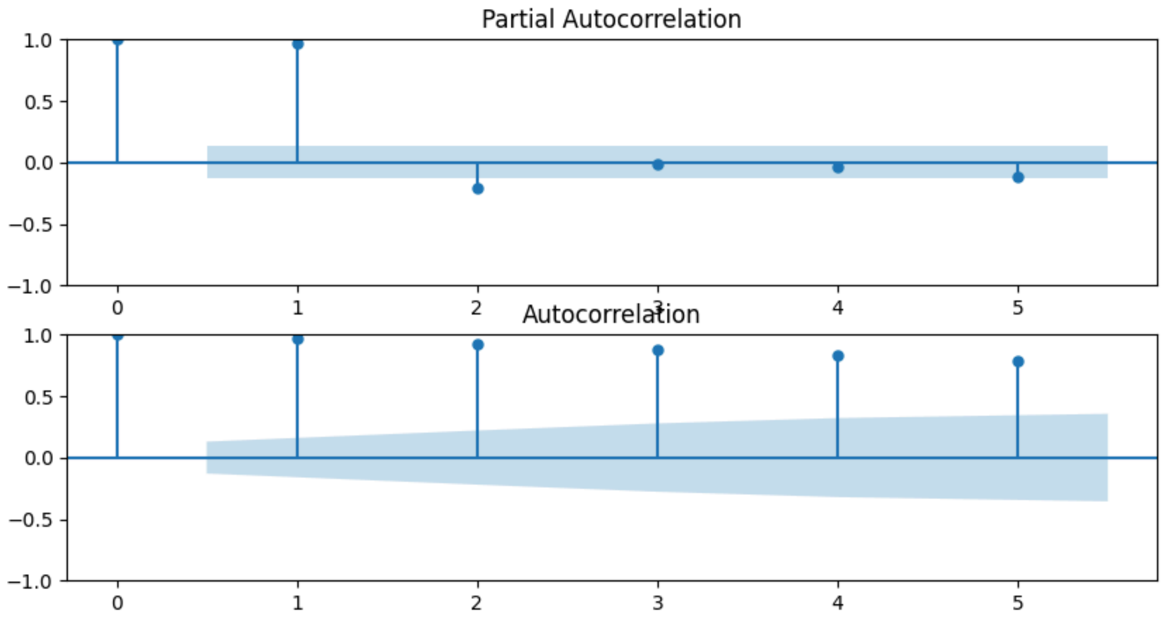




In [ ]:
```
plt.rcParams['figure.figsize']=(12,6)
decomposition = seasonal_decompose(merged_df.Value , period = 12 , model
decomposition.plot()
plt.savefig('Discription , trend , seasonal , residuals.png')
plt.show()
```



In [ ]:
```
fig , (ax1,ax2) = plt.subplots(nrows = 2 ,ncols = 1,sharex = False , shar
ax1 = plot_pacf(merged_df.Value , lags = 5 , ax = ax1)
ax2 = plot_acf(merged_df.Value , lags = 5 , ax = ax2)
#plt.savefig('Partial Autocorrelation and Autocorrelation.png')
plt.show()
```



Data Tranformation For ARIMAX

In [ ]:
```
def adf_check(time_series):
    result = adfuller(time_series , autolag = 'AIC')
    label = pd.Series(result[0:4], index=['Test Statistic','p-value','Num
    for key,value in result[4].items():
        label['Critical Value (%s)'%key] = value
    print(label)
    if result[1] <= 0.05:
        print('Strong evidence against the null hypothesis, hence REJECT n
    else:
        print ('Weak evidence against the null hypothesis, hence ACCEPT nu
```

In [ ]:
```
adf_check(merged_df.Value)
```

```
Test Statistic                   -2.589813
p-value                           0.095121
Number of Lags Used               2.000000
Number of Observations Used     226.000000
Critical Value (1%)              -3.459620
Critical Value (5%)              -2.874415
Critical Value (10%)             -2.573632
dtype: float64
Weak evidence against the null hypothesis, hence ACCEPT null hypothesis an
d the series is Not Stationary
```

In [ ]:
```
merged_df1 = merged_df.diff().dropna()
print('Count of merged policy rate', merged_df1.shape[0])
merged_df1.head()
```

```
Count of merged policy rate 228
```

Out[ ]:

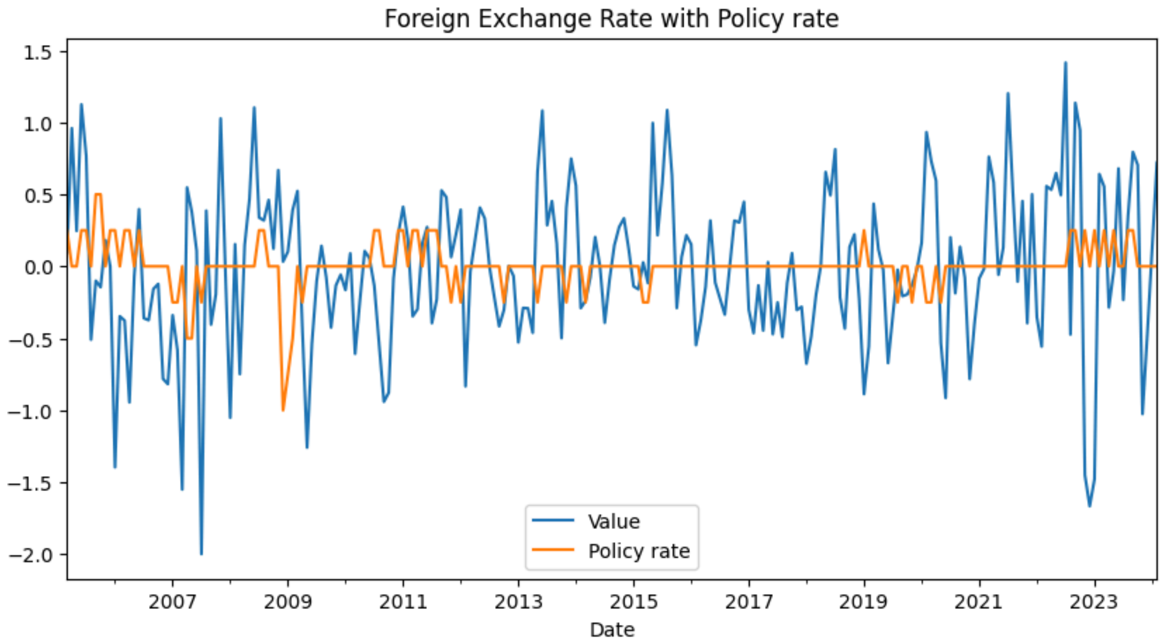
	Value	Policy rate
Date		
2005-03-31	0.097022	0.25
2005-04-30	0.959431	0.00

2005-05-31	0.246093	0.00
2005-06-30	1.124773	0.25
2005-07-31	0.766658	0.25

In [ ]:
```
adf_check(merged_df1['Value'])
adf_check(merged_df1['Policy rate'])
```

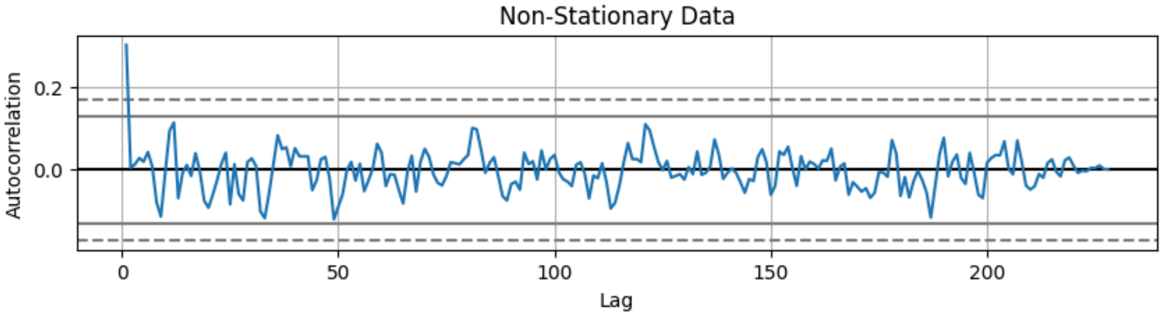
```
Test Statistic                  -1.004295e+01
p-value                          1.479677e-17
Number of Lags Used              1.000000e+00
Number of Observations Used      2.260000e+02
Critical Value (1%)             -3.459620e+00
Critical Value (5%)             -2.874415e+00
Critical Value (10%)            -2.573632e+00
dtype: float64
Strong evidence against the null hypothesis, hence REJECT null hypothesis
and the series is Stationary
Test Statistic                  -5.696197e+00
p-value                          7.863958e-07
Number of Lags Used              4.000000e+00
Number of Observations Used      2.230000e+02
Critical Value (1%)             -3.460019e+00
Critical Value (5%)             -2.874590e+00
Critical Value (10%)            -2.573725e+00
dtype: float64
Strong evidence against the null hypothesis, hence REJECT null hypothesis
and the series is Stationary
```

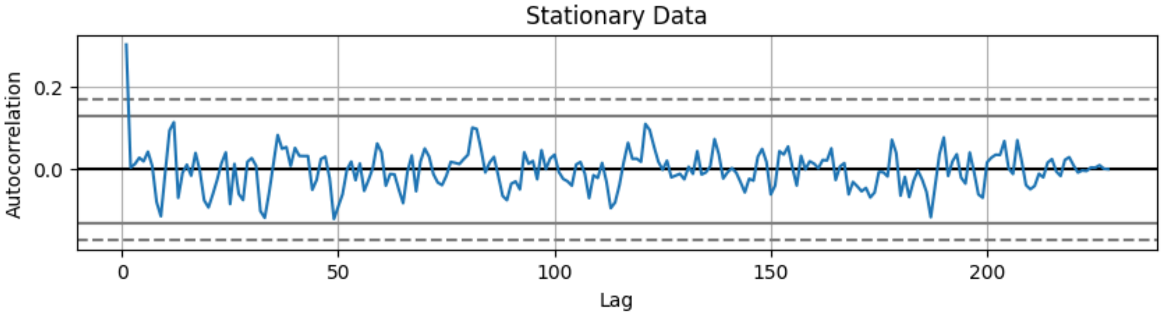
In [ ]:
```
merged_df1.plot(figsize = (10,5))
plt.title('Foreign Exchange Rate with Policy rate')
plt.show()
```



In [ ]:
```
fig, (ax1,ax2) = plt.subplots(nrows = 2, ncols = 1, sharex = False, share
ax1 = autocorrelation_plot(merged_df1, ax = ax1)
ax1.set_title('Non-Stationary Data')
ax2 = autocorrelation_plot(merged_df1 , ax = ax2)
ax2.set_title('Stationary Data')
```

```
plt.subplots_adjust(hspace = 0.5)
plt.savefig('Stationary data and Non-Stationary data.png')
plt.show()
```





Model Fitting

In [ ]:
```
model_sarimax = auto_arima(merged_df['Value'],
                           exog = merged_df['Policy rate'],
                           m = 12, d = 1,
                           seasonal = True,
                           max_order = 8,
                           test = 'adf',
                           trace = True)
```

```
Performing stepwise search to minimize aic
 ARIMA(2,1,2)(1,0,1)[12] intercept   : AIC=354.458, Time=1.13 sec
 ARIMA(0,1,0)(0,0,0)[12] intercept   : AIC=374.116, Time=0.05 sec
 ARIMA(1,1,0)(1,0,0)[12] intercept   : AIC=353.372, Time=0.16 sec
 ARIMA(0,1,1)(0,0,1)[12] intercept   : AIC=350.377, Time=0.15 sec
 ARIMA(0,1,0)(0,0,0)[12]             : AIC=372.216, Time=0.03 sec
 ARIMA(0,1,1)(0,0,0)[12] intercept   : AIC=352.793, Time=0.06 sec
 ARIMA(0,1,1)(1,0,1)[12] intercept   : AIC=348.831, Time=0.33 sec
 ARIMA(0,1,1)(1,0,0)[12] intercept   : AIC=349.696, Time=0.15 sec
 ARIMA(0,1,1)(2,0,1)[12] intercept   : AIC=350.528, Time=0.72 sec
 ARIMA(0,1,1)(1,0,2)[12] intercept   : AIC=350.423, Time=0.87 sec
 ARIMA(0,1,1)(0,0,2)[12] intercept   : AIC=350.986, Time=0.34 sec
 ARIMA(0,1,1)(2,0,0)[12] intercept   : AIC=349.923, Time=0.32 sec
 ARIMA(0,1,1)(2,0,2)[12] intercept   : AIC=352.375, Time=1.27 sec
 ARIMA(0,1,0)(1,0,1)[12] intercept   : AIC=373.094, Time=0.27 sec
 ARIMA(1,1,1)(1,0,1)[12] intercept   : AIC=350.771, Time=0.46 sec
 ARIMA(0,1,2)(1,0,1)[12] intercept   : AIC=350.765, Time=0.42 sec
 ARIMA(1,1,0)(1,0,1)[12] intercept   : AIC=352.983, Time=0.32 sec
 ARIMA(1,1,2)(1,0,1)[12] intercept   : AIC=352.755, Time=1.03 sec
 ARIMA(0,1,1)(1,0,1)[12]             : AIC=346.844, Time=0.20 sec
 ARIMA(0,1,1)(0,0,1)[12]             : AIC=348.407, Time=0.09 sec
 ARIMA(0,1,1)(1,0,0)[12]             : AIC=347.722, Time=0.08 sec
 ARIMA(0,1,1)(2,0,1)[12]             : AIC=348.540, Time=0.44 sec
 ARIMA(0,1,1)(1,0,2)[12]             : AIC=348.435, Time=0.47 sec
 ARIMA(0,1,1)(0,0,0)[12]             : AIC=350.848, Time=0.04 sec
 ARIMA(0,1,1)(0,0,2)[12]             : AIC=349.011, Time=0.22 sec
```

```
ARIMA(0,1,1)(2,0,0)[12]             : AIC=347.941, Time=0.17 sec
ARIMA(0,1,1)(2,0,2)[12]             : AIC=350.387, Time=1.14 sec
ARIMA(0,1,0)(1,0,1)[12]             : AIC=371.131, Time=0.25 sec
ARIMA(1,1,1)(1,0,1)[12]             : AIC=348.784, Time=0.32 sec
ARIMA(0,1,2)(1,0,1)[12]             : AIC=348.779, Time=0.26 sec
ARIMA(1,1,0)(1,0,1)[12]             : AIC=350.991, Time=0.23 sec
ARIMA(1,1,2)(1,0,1)[12]             : AIC=350.769, Time=0.61 sec

Best model:  ARIMA(0,1,1)(1,0,1)[12]
Total fit time: 12.628 seconds
```

In [ ]:
```
model_sarimax.summary()
```

Out[ ]:

SARIMAX Results

Dep. Variable:	y	**No. Observations:**	229
Model:	SARIMAX(0, 1, 1)x(1, 0, 1, 12)	**Log Likelihood**	-169.422
Date:	Sat, 06 Apr 2024	**AIC**	346.844
Time:	04:37:48	**BIC**	360.562
Sample:	02-28-2005	**HQIC**	352.379
	- 02-29-2024		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ma.L1	0.3578	0.049	7.288	0.000	0.262	0.454
ar.S.L12	0.7065	0.300	2.354	0.019	0.118	1.295
ma.S.L12	-0.5551	0.340	-1.633	0.102	-1.221	0.111
sigma2	0.2578	0.020	12.831	0.000	0.218	0.297

Ljung-Box (L1) (Q):	0.01	**Jarque-Bera (JB):**	12.64
Prob(Q):	0.94	**Prob(JB):**	0.00
Heteroskedasticity (H):	0.94	**Skew:**	-0.25
Prob(H) (two-sided):	0.80	**Kurtosis:**	4.04

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

In [ ]:
```
model_sarimax1 = SARIMAX(merged_df['Value'],
                         order = (0, 1, 1),
                         seasonal_order = (1, 0, 1, 12),
                         exog = merged_df['Policy rate'],
                         freq = 'M',
                         enforce_stationarity=False,
```

```
                           enforce_invertibility=False)
result_SARIMAX = model_sarimax1.fit(disp = False)
result_SARIMAX.summary()
```

Out[ ]:

SARIMAX Results

Dep. Variable:	Value	**No. Observations:**	229
Model:	SARIMAX(0, 1, 1)x(1, 0, 1, 12)	**Log Likelihood**	-155.304
Date:	Sat, 06 Apr 2024	**AIC**	320.608
Time:	04:37:48	**BIC**	337.438
Sample:	02-28-2005	**HQIC**	327.409
	- 02-29-2024		
Covariance Type:	opg		

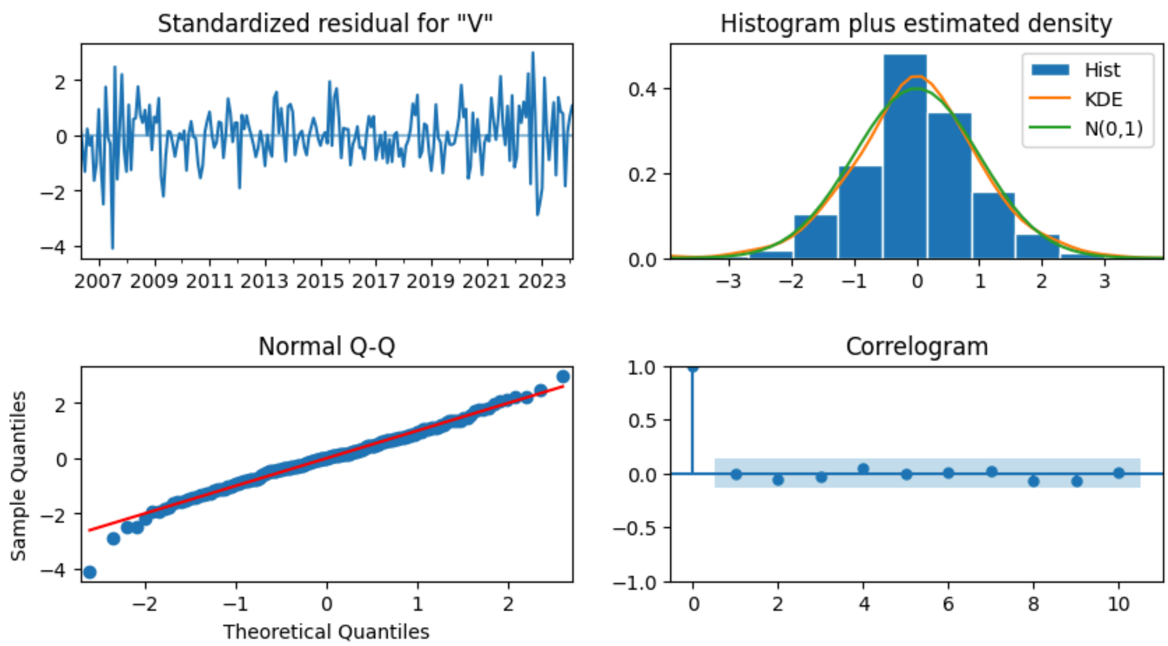
	coef	std err	z	P>|z|	[0.025	0.975]
Policy rate	-0.1691	0.216	-0.784	0.433	-0.592	0.254
ma.L1	0.3421	0.050	6.904	0.000	0.245	0.439
ar.S.L12	0.5139	0.162	3.164	0.002	0.196	0.832
ma.S.L12	-0.3652	0.180	-2.024	0.043	-0.719	-0.011
sigma2	0.2481	0.020	12.298	0.000	0.209	0.288

Ljung-Box (L1) (Q):	0.00	**Jarque-Bera (JB):**	14.92
Prob(Q):	0.97	**Prob(JB):**	0.00
Heteroskedasticity (H):	1.07	**Skew:**	-0.26
Prob(H) (two-sided):	0.77	**Kurtosis:**	4.19

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

In [ ]:

```
result_SARIMAX.plot_diagnostics(figsize = (10,5))
plt.subplots_adjust(hspace = 0.5)
# plt.savefig('Diagnostic plot of best SARIMAX model.png')
plt.show()
```



In [ ]:
```
predictions_sarimax = result_SARIMAX.predict(typ = 'levels')
```

In [ ]:
```
print('Evaluation Result for whole data : ','\n')
print('R2 Score for whole data : {0:.2f} %'.format(100*r2_score(merged_df
print('Mean Squared Error : ',mean_squared_error(merged_df['Value'],predi
print('Mean Absolute Error : ',mean_absolute_error(merged_df['Value'],pre
print('Root Mean Squared Error : ',sqrt(mean_squared_error(merged_df['Val
print('Mean Absolute Percentage Error : {0:.2f} %'.format(100*mean_absolu
```

```
Evaluation Result for whole data :

R2 Score for whole data : -6.07 %

Mean Squared Error :  7.3042039878750336

Mean Absolute Error :  0.6011954278484114

Root Mean Squared Error :  2.7026290881056974

Mean Absolute Percentage Error : 1.71 %
```

In [ ]:
```
Final_data = pd.concat([merged_df, merged_df1,
                        predictions_sarimax],axis=1)
Final_data.columns = ['Foreign Exchange Rate (monthly)',
                      'Policr Rate (monthly)',
                      'Monthly First Difference',
                      'Predicted Policy Rate',
                      'Predicted Exchange Rate']
#Final_data.to_csv('Foreign Exchange Rate with Prediction (THB To USD).cs
Final_data.head()
```

Out[ ]:

	Foreign Exchange Rate (monthly)	Policr Rate (monthly)	Monthly First Difference	Predicted Policy Rate	Predicted Exchange Rate
2005-02-28	38.459500	2.00	NaN	NaN	-0.338118

2005-03-31	38.556522	2.25	0.097022	0.25	38.417235
2005-04-30	39.515952	2.25	0.959431	0.00	38.556522
2005-05-31	39.762045	2.25	0.246093	0.00	39.515952
2005-06-30	40.886818	2.50	1.124773	0.25	39.719781

Model Testing

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

train_set = merged_df[:train_size]
test_set = merged_df[train_size:]

print('Counts of Train Data : ',train.shape[0])
print('Counts of Test Data : ',test.shape[0])

print(train_set)
print(test_set)
```

```
Counts of Train Data :  183
Counts of Test Data :  46
                Value  Policy rate
Date
2005-02-28  38.459500         2.00
2005-03-31  38.556522         2.25
2005-04-30  39.515952         2.25
2005-05-31  39.762045         2.25
2005-06-30  40.886818         2.50
...               ...          ...
2019-12-31  30.226818         1.25
2020-01-31  30.390435         1.25
2020-02-29  31.322250         1.00
2020-03-31  32.052727         0.75
2020-04-30  32.647727         0.75

[183 rows x 2 columns]
                Value  Policy rate
Date
2020-05-31  32.115714         0.50
2020-06-30  31.201818         0.50
2020-07-31  31.403043         0.50
2020-08-31  31.216190         0.50
2020-09-30  31.351818         0.50
2020-10-31  31.272955         0.50
2020-11-30  30.490476         0.50
2020-12-31  30.090870         0.50
2021-01-31  30.007143         0.50
2021-02-28  29.990000         0.50
2021-03-31  30.751087         0.50
2021-04-30  31.334091         0.50
2021-05-31  31.275714         0.50
```

```
2021-06-30  31.405909         0.50
2021-07-31  32.607273         0.50
2021-08-31  33.122955         0.50
2021-09-30  33.016818         0.50
2021-10-31  33.469762         0.50
2021-11-30  33.075000         0.50
2021-12-31  33.575217         0.50
2022-01-31  33.222619         0.50
2022-02-28  32.665500         0.50
2022-03-31  33.222391         0.50
2022-04-30  33.754762         0.50
2022-05-31  34.402727         0.50
2022-06-30  34.897273         0.50
2022-07-31  36.312381         0.50
2022-08-31  35.838913         0.75
2022-09-30  36.974091         1.00
2022-10-31  37.918095         1.00
2022-11-30  36.468636         1.25
2022-12-31  34.802500         1.25
2023-01-31  33.323864         1.50
2023-02-28  33.965000         1.50
2023-03-31  34.519130         1.75
2023-04-30  34.233750         1.75
2023-05-31  34.201957         2.00
2023-06-30  34.881591         2.00
2023-07-31  34.648333         2.00
2023-08-31  35.005217         2.25
2023-09-30  35.799524         2.50
2023-10-31  36.503409         2.50
2023-11-30  35.477500         2.50
2023-12-31  35.004286         2.50
2024-01-31  35.133043         2.50
2024-02-29  35.852381         2.50
```

In [ ]:
```
train_values_Poli_add_Date = train_set.loc[train_set.index]
print(train_values_Poli_add_Date)
```

```
                Value  Policy rate
Date
2005-02-28  38.459500         2.00
2005-03-31  38.556522         2.25
2005-04-30  39.515952         2.25
2005-05-31  39.762045         2.25
2005-06-30  40.886818         2.50
...               ...          ...
2019-12-31  30.226818         1.25
2020-01-31  30.390435         1.25
2020-02-29  31.322250         1.00
2020-03-31  32.052727         0.75
2020-04-30  32.647727         0.75

[183 rows x 2 columns]
```

In [ ]:
```
import pandas as pd
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.statespace.sarimax import SARIMAX
from sklearn.metrics import mean_squared_error
```

In [ ]:
```
train_values = train_set['Value']
train_values
```

```python
train_values_Poli = [y for y in train_set['Policy rate']]
train_values_Poli_copy = train_set.copy()
train_values_Poli = train_set['Policy rate'].to_frame()
prediction = []
print('Printing Predictied vs Expected Values....')
print('\n')

# for t in range(len(test)):
for t, value in enumerate(test):
    model = SARIMAX(endog = train_values,
                    order = (0, 1, 1),
                    seasonal_order = (1, 0, 1, 12),
                    exog = train_values_Poli,
                    freq = 'M',
                    enforce_stationarity=False,
                    enforce_invertibility=False)
    model_fit = model.fit()
    policy_model_arima = ARIMA(train_values_Poli['Policy rate'],
                               order = (1,1,1))
    policy_model_arima_fit = policy_model_arima.fit()
    future_policy_rates_arima = policy_model_arima_fit.forecast()
    output = model_fit.forecast(exog = future_policy_rates_arima)
    pred_out = output[0]
    prediction.append(float(pred_out))
    train_values = []
    train_values.append(value)
    print('Predicted = %f, Actual = %f' % (pred_out, train_values[-1]))
```

```
Printing Predictied vs Expected Values....


Predicted = 32.811551, Actual = 2.500000
```