In [ ]:
```
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 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.metrics import r2_score , mean_absolute_error , mean_absolute_percentage_error , mean_squared_error
import pickle
import warnings
warnings.filterwarnings('ignore')
```

In [ ]:
```
os.chdir('C:\\Users\\santa\\OneDrive\\Documents\\KMUTT-4\\Final_PJ\\Data')
df = pd.read_csv('Policy_rate_data.csv')
df.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 [ ]:
```
df.shape
```

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

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

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

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

Out[ ]:
```
0
```

In [ ]:
```
df.dtypes
```

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

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

Out[ ]:

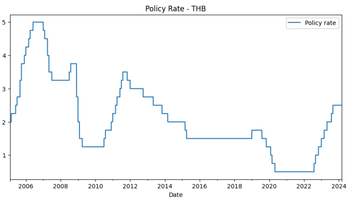
	Policy rate
count	6968.000000
mean	2.092028
std	1.129977
min	0.500000
25%	1.500000
50%	1.750000
75%	2.750000
max	5.000000

Data Processing

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

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

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



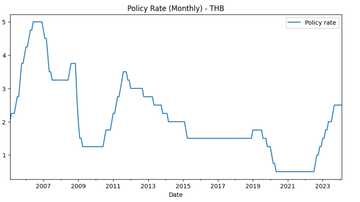
In [ ]:
```
df_month = df.resample('M').mean()
print('Count Of The Monthly Data Frame : ',df_month.shape[0])
df_month.head()
```

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



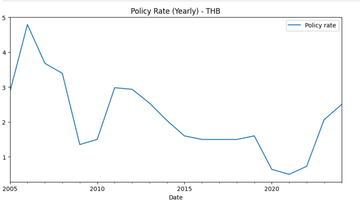
In [ ]:
```
df_year = df.resample('Y').mean()
print('Count Of The Yearly Data Frame : ',df_year.shape[0])
df_year.head()
```

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



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

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[83], line 2
      1 plt.rcParams['figure.figsize'] = (15,7)
----> 2 sns.scatterplot(x = df_month.index , y = df_month.values , color = 'black')
      3 plt.title('Policy Rate (monthly) - THB to USD [Scatter Plot]')
      4 #plt.savefig('Foreign Exchange Rate (weekly) - THB to USD [Scatter Plot].png')

File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\LocalCache\local-packages\Python311\site-packages\seaborn\relational.py:615, in scatterplot(data, x, y, hue, size, style, palette, hue_order, hue_norm, size_norm, size_order, size_norm, markers, style_order, legend, ax, **kwargs)
    606 def scatterplot(
    607     data=None, *,
    608     x=None, y=None, hue=None, size=None, style=None,
   (...)
    612     **kwargs
    613 ):
--> 615     p = _ScatterPlotter(
    616         data=data,
    617         variables=dict(x=x, y=y, hue=hue, size=size, style=style),
    618         legend=legend
    619     )
    621     p.map_hue(palette=palette, order=hue_order, norm=hue_norm)
    622     p.map_size(sizes=sizes, order=size_order, norm=size_norm)

File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\LocalCache\local-packages\Python311\site-packages\seaborn\relational.py:396, in _ScatterPlotter.__init__(self, data, variables, legend)
    387 def __init__(self, *, data=None, variables={}, legend=None):
    388
    389     # TODO this is messy, we want the mapping to be agnostic about
    390     # the kind of plot to draw, but for the time being we need to set
    391     # this information so the SizeMapping can use it
    392     self._default_size_range = (
    393         np.r_[.5, 2] * np.square(mpl.rcParams["lines.markersize"])
    394     )
--> 396     super().__init__(data=data, variables=variables)
    398     self.legend = legend

File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\LocalCache\local-packages\Python311\site-packages\seaborn\_base.py:634, in VectorPlotter.__init__(self, data, variables)
    629 # var_ordered is relevant only for categorical axis variables, and may
    630 # be better handled by an internal axis information object that tracks
    631 # such information and is set up by the scale_* methods. The analogous
    632 # information for numeric axes would be information about log scales.
    633 self._var_ordered = {"x": False, "y": False}  # alt., used DefaultDict
--> 634 self.assign_variables(data, variables)
    636 # TODO Lots of tests assume that these are called to initialize the
    637 # mappings to default values on class initialization. I'd prefer to
    638 # move away from that and only have a mapping when explicitly called.
    639 for var in ["hue", "size", "style"]:

File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\LocalCache\local-packages\Python311\site-packages\seaborn\_base.py:679, in VectorPlotter.assign_variables(self, data, variables)
    674 else:
    675     # When dealing with long-form input, use the newer PlotData
    676     # object (internal but introduced for the objects interface)
    677     # to centralize / standardize data consumption logic.
    678     self.input_format = "long"
--> 679     plot_data = PlotData(data, variables)
    680     frame = plot_data.frame
    681     names = plot_data.names

File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\LocalCache\local-packages\Python311\site-packages\seaborn\_core\data.py:58, in PlotData.__init__(self, data, variables)
     51 def __init__(
     52     self,
     53     data: DataSource,
     54     variables: dict[str, VariableSpec],
     55 ):
     57     data = handle_data_source(data)
---> 58     frame, names, ids = self._assign_variables(data, variables)
     60     self.frame = frame
     61     self.names = names

File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\LocalCache\local-packages\Python311\site-packages\seaborn\_core\data.py:265, in PlotData._assign_variables(self, data, variables)
    260         ids[key] = id(val)
    262 # Construct a tidy plot DataFrame. This will convert a number of
    263 # types automatically, aligning on index in case of pandas objects
    264 # TODO note: this fails when variable specs *only* have scalars!
--> 265 frame = pd.DataFrame(plot_data)
    267 return frame, names, ids

File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\LocalCache\local-packages\Python311\site-packages\pandas\core\frame.py:767, in DataFrame.__init__(self, data, index, columns, dtype, copy)
    761     mgr = self._init_mgr(
    762         data, axes={"index": index, "columns": columns}, dtype=dtype, copy=copy
    763     )
    765 elif isinstance(data, dict):
    766     # GH#38939 de facto copy defaults to False only in non-dict cases
--> 767     mgr = dict_to_mgr(data, index, columns, dtype=dtype, copy=copy, typ=manager)
    768 elif isinstance(data, ma.MaskedArray):
    769     from numpy.ma import mrecords

File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\LocalCache\local-packages\Python311\site-packages\pandas\core\internals\construction.py:503, in dict_to_mgr(data, index, columns, dtype, typ, copy)
    499     else:
    500         # dtype check to exclude e.g. range objects, scalars
    501         arrays = [x.copy() if hasattr(x, "dtype") else x for x in arrays]
--> 503 return arrays_to_mgr(arrays, columns, index, dtype=dtype, typ=typ, consolidate=copy)

File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\LocalCache\local-packages\Python311\site-packages\pandas\core\internals\construction.py:114, in arrays_to_mgr(arrays, columns, index, dtype, verify_integrity, typ, consolidate)
    111 if verify_integrity:
    112     # figure out the index, if necessary
    113     if index is None:
--> 114         index = _extract_index(arrays)
    115     else:
    116         index = ensure_index(index)

File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\LocalCache\local-packages\Python311\site-packages\pandas\core\internals\construction.py:664, in _extract_index(data)
    662         raw_lengths.append(len(val))
    663     elif isinstance(val, np.ndarray) and val.ndim > 1:
--> 664         raise ValueError("Per-column arrays must each be 1-dimensional")
    666 if not indexes and not raw_lengths:
    667     raise ValueError("If using all scalar values, you must pass an index")

ValueError: Per-column arrays must each be 1-dimensional
```