```
!pip install pyts
!pip install keras_tuner
```

```
Requirement already satisfied: pyts in /usr/local/lib/python3.10/dist-packages (0.13.0)
Requirement already satisfied: numpy>=1.22.4 in /usr/local/lib/python3.10/dist-packages (from pyts) (1.25.2)
Requirement already satisfied: scipy>=1.8.1 in /usr/local/lib/python3.10/dist-packages (from pyts) (1.11.4)
Requirement already satisfied: scikit-learn>=1.2.0 in /usr/local/lib/python3.10/dist-packages (from pyts) (1.2.2)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from pyts) (1.3.2)
Requirement already satisfied: numba>=0.55.2 in /usr/local/lib/python3.10/dist-packages (from pyts) (0.58.1)
Requirement already satisfied: llvmlite<0.42,>=0.41.0dev0 in /usr/local/lib/python3.10/dist-packages (from numba>=0.55.2->pyts) (0.41.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=1.2.0->pyts) (3.3.0)
Collecting keras_tuner
  Downloading keras_tuner-1.4.6-py3-none-any.whl (128 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 128.9/128.9 kB 1.3 MB/s eta 0:00:00
Requirement already satisfied: keras in /usr/local/lib/python3.10/dist-packages (from keras_tuner) (2.15.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from keras_tuner) (23.2)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from keras_tuner) (2.31.0)
Collecting kt-legacy (from keras_tuner)
  Downloading kt_legacy-1.0.5-py3-none-any.whl (9.6 kB)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->keras_tuner) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->keras_tuner) (3.6)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->keras_tuner) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->keras_tuner) (2024.2.2)
Installing collected packages: kt-legacy, keras_tuner
Successfully installed keras_tuner-1.4.6 kt-legacy-1.0.5
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm
from statsmodels.tsa.stattools import kpss, adfuller
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from sklearn.model_selection import train_test_split
import keras_tuner as kt
import tensorflow as tf
from keras_tuner import HyperModel
from sklearn import metrics
from pyts.image import GramianAngularField
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import mean_squared_error, accuracy_score, classification_report
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
import os
from keras import backend as K
```

Step 1 Data Preparation

Original Time Series

The dataset used in this project contains 1-minute bars of S&P 500 during Feb 15, 2024. downloaded from https://www.histdata.com/download-free-forex-historical-data/. The file is in Excel format and has been zipped alongside this notebook.

```
from google.colab import files

uploaded = files.upload()
```

选择文件 SP500 15.02.2024.xlsx

- **SP500 15.02.2024.xlsx**(application/vnd.openxmlformats-officedocument.spreadsheetml.sheet) - 68876 bytes, last modified: 2024/2/27 - 100% done

Saving SP500 15.02.2024.xlsx to SP500 15.02.2024.xlsx

```
df = pd.read_excel('/content/SP500 15.02.2024.xlsx')
df.columns = ['Date', "Time", "Open", "High", "Low", "Close"]
df['DateTime'] = pd.to_datetime(df['Date'].astype(str) + ' ' + df['Time'].astype(str))
df.set_index('DateTime', inplace=True)
df = df.drop(columns=['Date', "Time", "Open", 'High', 'Low'])
df.head(10)
```

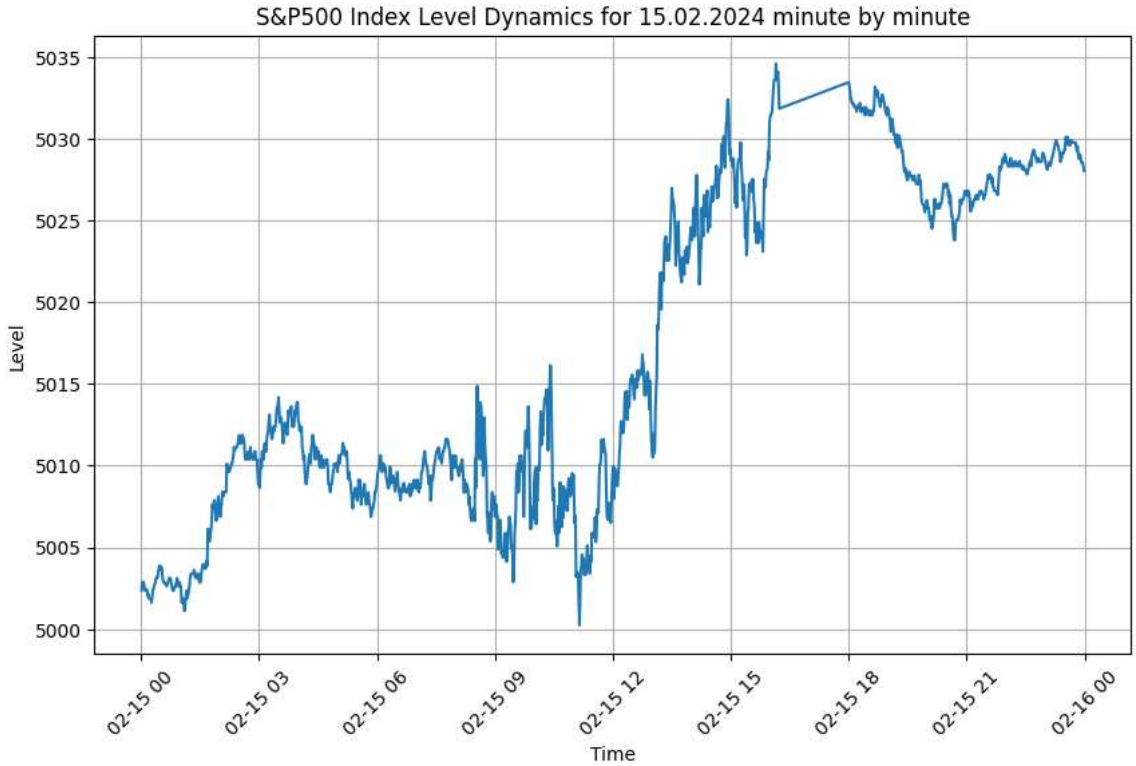
DateTime	Close
2024-02-15 00:01:00	5002.376
2024-02-15 00:02:00	5002.864
2024-02-15 00:03:00	5002.629
2024-02-15 00:04:00	5002.876
2024-02-15 00:05:00	5002.611
2024-02-15 00:06:00	5002.370
2024-02-15 00:07:00	5002.427
2024-02-15 00:09:00	5002.427
2024-02-15 00:10:00	5002.047
2024-02-15 00:11:00	5002.177

There is data missing at 00:08:00.

```
df.shape
```

```
(1271, 1)
```

```
# Plot time series
plt.figure(figsize=(10, 6))
plt.plot(df['Close'])
plt.title('S&P500 Index Level Dynamics for 15.02.2024 minute by minute')
plt.xlabel('Time')
plt.ylabel('Level')
plt.grid(True)
plt.xticks(rotation= 45)
plt.show()
```



The full 1-min price data of a day should contain $24 \times 60 = 1440$ rows. After some inspection, there is no data between 16:15 and 18:00 each day, during which the U.S. stock market is closed.

To avoid any implication of this discontinuation, we decide to separate this data into two time series, one for training and the other for testing.

```
df1 = df[df.index <= '2024-02-15 16:15:00']
df1 = df1.reindex(pd.date_range(start=df1.index[0], end=df1.index[-1], freq='min'))
df1 = df1.fillna(method='pad')

print(df1.shape)
df1.tail(10)
```

(974, 1)

	Close
2024-02-15 16:05:00	5033.061
2024-02-15 16:06:00	5033.582
2024-02-15 16:07:00	5033.573
2024-02-15 16:08:00	5033.561
2024-02-15 16:09:00	5034.567
2024-02-15 16:10:00	5033.567
2024-02-15 16:11:00	5033.826
2024-02-15 16:12:00	5034.079
2024-02-15 16:13:00	5032.842
2024-02-15 16:14:00	5031.824

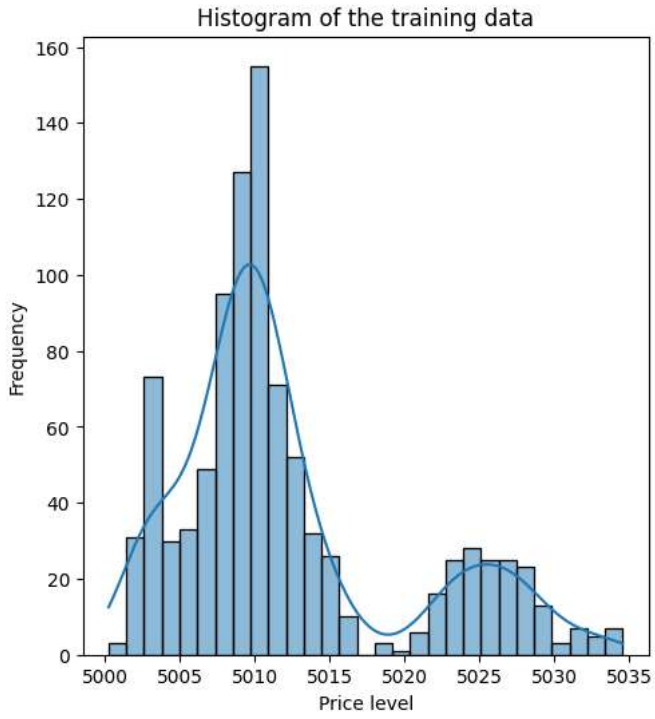
```
df2 = df[df.index > '2024-02-15 16:15:00']
df2 = df2.reindex(pd.date_range(start=df2.index[0], end=df2.index[-1], freq='min'))
df2 = df2.fillna(method='pad')

print(df2.shape)
df2.head(10)
```

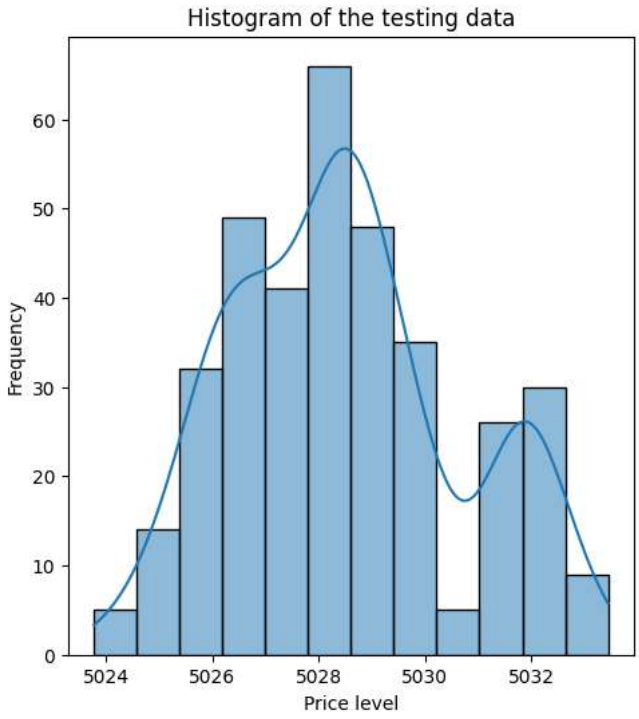
(360, 1)

	Close
2024-02-15 18:00:00	5033.437
2024-02-15 18:01:00	5033.143
2024-02-15 18:02:00	5032.887
2024-02-15 18:03:00	5032.381
2024-02-15 18:04:00	5032.390
2024-02-15 18:05:00	5032.134
2024-02-15 18:06:00	5032.146
2024-02-15 18:07:00	5032.137
2024-02-15 18:08:00	5031.947
2024-02-15 18:09:00	5032.029

```
# Create the histogram
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
sns.histplot(df1['Close'], kde=True)
plt.title('Histogram of the training data')
plt.xlabel('Price level')
plt.ylabel('Frequency')

plt.subplot(1, 2, 2)
sns.histplot(df2['Close'], kde=True)
plt.title('Histogram of the testing data')
plt.xlabel('Price level')
plt.ylabel('Frequency')
```

```
Text(0, 0.5, 'Frequency')
```






```
# Perform ADF test
print('Results of ADF Test:')
dftest = adfuller(df1['Close'])
dfoutput = pd.Series(dftest[0:4], index=['Test Statistic','P-value','#Lags Used','Number of Observations Used'])
for key,value in dftest[4].items():
    dfoutput['Critical Value (%s)'%key] = value
print(dfoutput)
```

```
Results of ADF Test:
Test Statistic                  -0.625607
P-value                          0.865090
#Lags Used                       4.000000
Number of Observations Used    969.000000
Critical Value (1%)             -3.437116
Critical Value (5%)             -2.864527
Critical Value (10%)            -2.568361
dtype: float64
```

```
# Perform ADF test
print('Results of ADF Test:')
dftest = adfuller(df2['Close'])
dfoutput = pd.Series(dftest[0:4], index=['Test Statistic','P-value','#Lags Used','Number of Observations Used'])
for key,value in dftest[4].items():
    dfoutput['Critical Value (%s)'%key] = value
print(dfoutput)
```

```
Results of ADF Test:
Test Statistic                  -2.326153
P-value                          0.163672
#Lags Used                       0.000000
Number of Observations Used    359.000000
Critical Value (1%)             -3.448697
Critical Value (5%)             -2.869625
Critical Value (10%)            -2.571077
dtype: float64
```

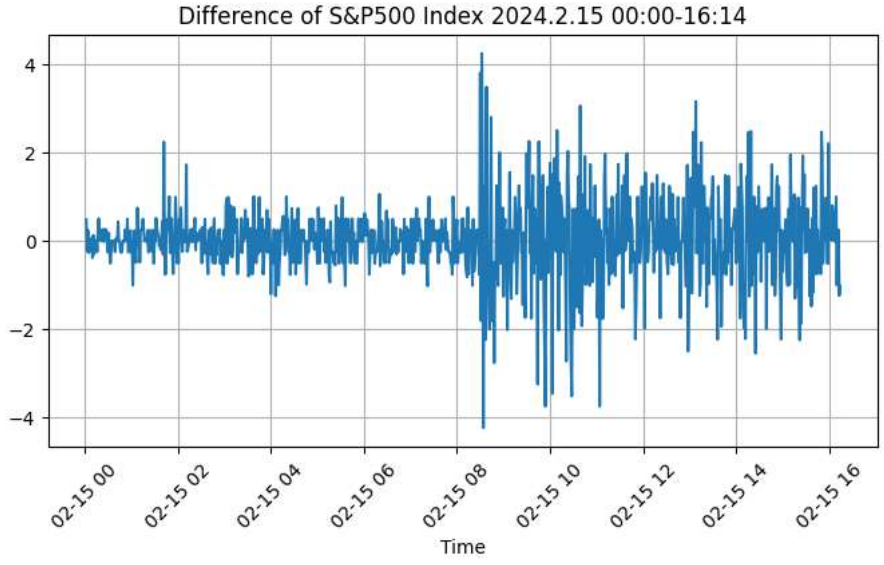
Both p-values are greater than 0.05, so we fail to reject the null hypothesis and this suggests that the time series is not stationary.

Stationary Time Series

```
df1_diff = df1['Close'].diff()[1:].to_frame()
```

```
plt.figure(figsize=(8,  4))
plt.plot(df1_diff['Close'])
plt.title('Difference  of  S&P500  Index  2024.2.15  00:00-16:14')
plt.xlabel('Time')
plt.grid(True)
plt.xticks(rotation=  45)
```

```
(array([19768.        , 19768.08333333, 19768.16666667, 19768.25      ,
        19768.33333333, 19768.41666667, 19768.5       , 19768.58333333,
        19768.66666667]),
 [Text(19768.0, 0, '02-15 00'),
  Text(19768.083333333332, 0, '02-15 02'),
  Text(19768.166666666668, 0, '02-15 04'),
  Text(19768.25, 0, '02-15 06'),
  Text(19768.333333333332, 0, '02-15 08'),
  Text(19768.416666666668, 0, '02-15 10'),
  Text(19768.5, 0, '02-15 12'),
  Text(19768.583333333332, 0, '02-15 14'),
  Text(19768.666666666668, 0, '02-15 16')])
```




```
#  Perform  ADF  test
print('Results  of  ADF  Test:')
dftest  =  adfuller(df1_diff['Close'])
dfoutput  =  pd.Series(dftest[0:4],  index=['Test  Statistic','P-value','#Lags  Used','Number  of  Observations  Used'])
for  key,value  in  dftest[4].items():
    dfoutput['Critical  Value  (%s)'%key]  =  value
print(dfoutput)
```

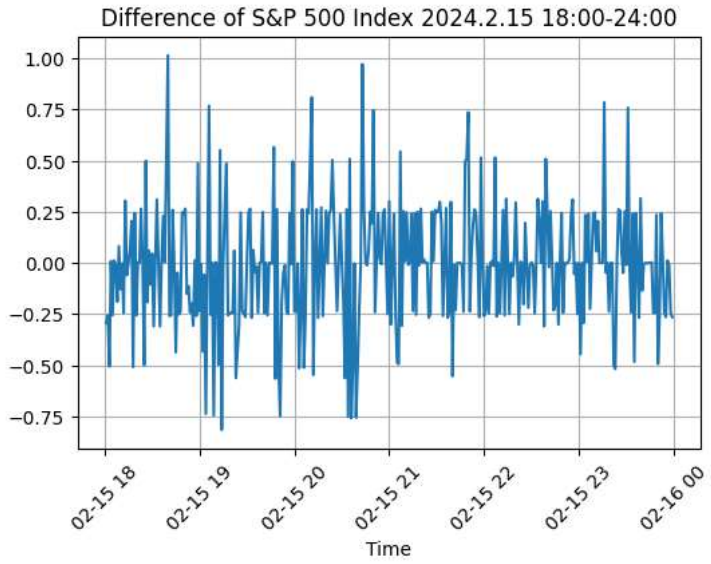
```
Results of ADF Test:
Test Statistic                -1.755825e+01
P-value                        4.109165e-30
#Lags Used                     3.000000e+00
Number of Observations Used    9.690000e+02
Critical Value (1%)           -3.437116e+00
Critical Value (5%)           -2.864527e+00
Critical Value (10%)          -2.568361e+00
dtype: float64
```

The p-values is smaller than 0.05, so we reject the null hypothesis, meaning that the new series has no unit root and thus is stationary.

```
df2_diff  =  df2['Close'].diff()[1:].to_frame()
```

```
plt.figure(figsize=(6,  4))
plt.plot(df2_diff['Close'])
plt.title('Difference  of  S&P  500  Index  2024.2.15  18:00-24:00')
plt.xlabel('Time')
plt.grid(True)
plt.xticks(rotation=  45)
```

```
(array([19768.75       , 19768.79166667, 19768.83333333, 19768.875      ,
        19768.91666667, 19768.95833333, 19769.         ]),
 [Text(19768.75, 0, '02-15 18'),
  Text(19768.791666666668, 0, '02-15 19'),
  Text(19768.833333333332, 0, '02-15 20'),
  Text(19768.875, 0, '02-15 21'),
  Text(19768.916666666668, 0, '02-15 22'),
  Text(19768.958333333332, 0, '02-15 23'),
  Text(19769.0, 0, '02-16 00')])
```




```
# Perform ADF test
print('Results of ADF Test:')
dftest = adfuller(df2_diff['Close'])
dfoutput = pd.Series(dftest[0:4], index=['Test Statistic','P-value','#Lags Used','Number of Observations Used'])
for key,value in dftest[4].items():
    dfoutput['Critical Value (%s)'%key] = value
print(dfoutput)
```

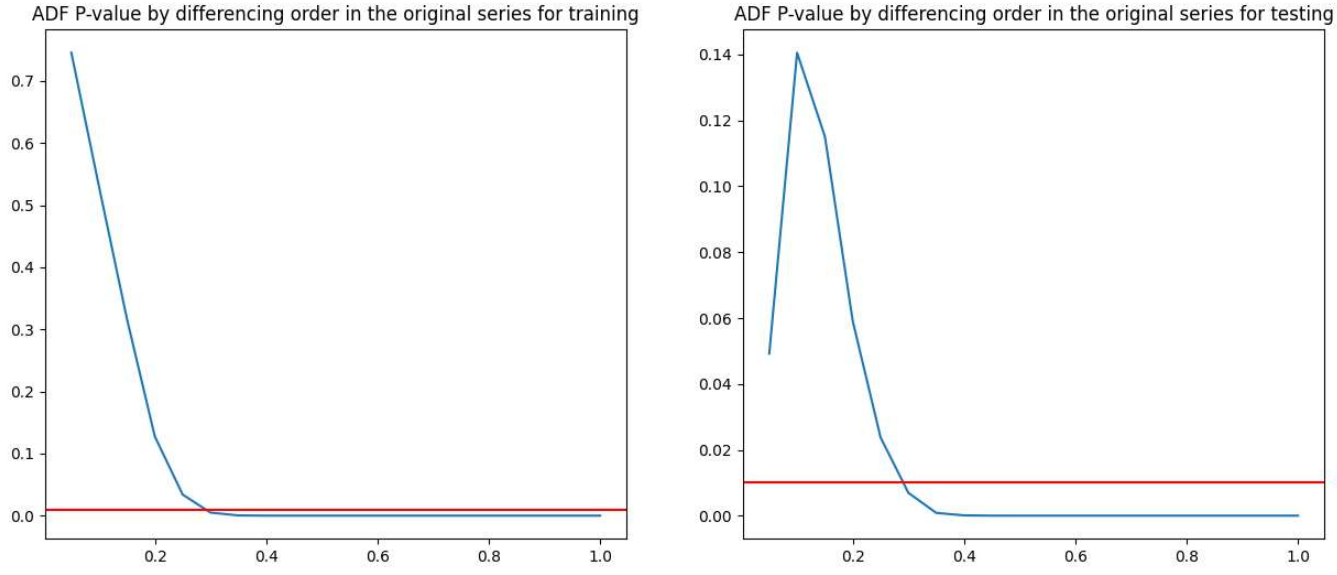
```
Results of ADF Test:
Test Statistic                -19.582866
P-value                         0.000000
#Lags Used                      0.000000
Number of Observations Used   358.000000
Critical Value (1%)            -3.448749
Critical Value (5%)            -2.869647
Critical Value (10%)           -2.571089
dtype: float64
```

The p-values is zero, so we reject the null hypothesis, meaning that the new series has no unit root and thus is stationary.

Fractional Differencing

```
def getWeights(d,lags):
        # return the weights from the series expansion of the differencing operator
        # for real orders d and up to lags coefficients
        w=[1]
        for k in range(1,lags):
                w.append(-w[-1]*((d-k+1))/k)
        w=np.array(w).reshape(-1,1)
        return w

def cutoff_find(order,cutoff,start_lags):
        # order is our dearest d, cutoff is 1e-4 for us, and start
        # lags is an initial amount of lags in which the loop will start,
        # this can be set to high values in order to speed up the algo
        val=np.inf
        lags=start_lags
        while abs(val)>cutoff:
                w=getWeights(order, lags)
                val=w[len(w)-1]
                lags+=1
        return lags

def ts_differencing_tau(series, order, tau):
        # return the time series resulting from (fractional) differencing
        lag_cutoff=(cutoff_find(order,tau,1)) #finding lag cutoff with tau
        # print(lag_cutoff)
        weights=getWeights(order, lag_cutoff)
        res=0
        for k in range(lag_cutoff):
                res += weights[k]*series.shift(k).fillna(0)
        return res[lag_cutoff:]


possible_d=np.divide(range(5,101,5),100)
tau = 1e-3
original_adf_stat_holder1=[None]*len(possible_d)
original_adf_stat_holder2=[None]*len(possible_d)

for i in range(len(possible_d)):
        original_adf_stat_holder1[i]=adfuller(ts_differencing_tau(df1['Close'],possible_d[i],tau))[1]
        original_adf_stat_holder2[i]=adfuller(ts_differencing_tau(df2['Close'],possible_d[i],tau))[1]


#now the plots of the ADF p-values
fig, axs = plt.subplots(1,2,figsize=(15,6))
axs[0].plot(possible_d,original_adf_stat_holder1)
axs[0].axhline(y=0.01,color='r')
axs[0].set_title('ADF P-value by differencing order in the original series for training')
axs[1].plot(possible_d,original_adf_stat_holder2)
axs[1].axhline(y=0.01,color='r')
axs[1].set_title('ADF P-value by differencing order in the original series for testing')
```

```
Text(0.5, 1.0, 'ADF P-value by differencing order in the original series for testing')
```



"Larger time series would allow smaller cutoff values in order to preserve more memory, it ends up being a trade-off between memory conservation and computational efficiency. In short time series however, you might want to stay away from low threshold values, since they can (and will) greatly diminish the size of your training set." [https://www.kaggle.com/code/elvisesp/time-series-analysis-using-fractional-differencing/notebook]

Since our data set is small, we choose the cutoff value a bit higher at 0.001. From the graph we can see that 0.4 seems to be a good choice for the order of differencing.

```
cutoff_find(0.4, tau, 1)
```

```
57
```

```
cutoff_find(0.6, tau, 1)
```

```
36
```

```
cutoff_find(0.8, tau, 1)
```

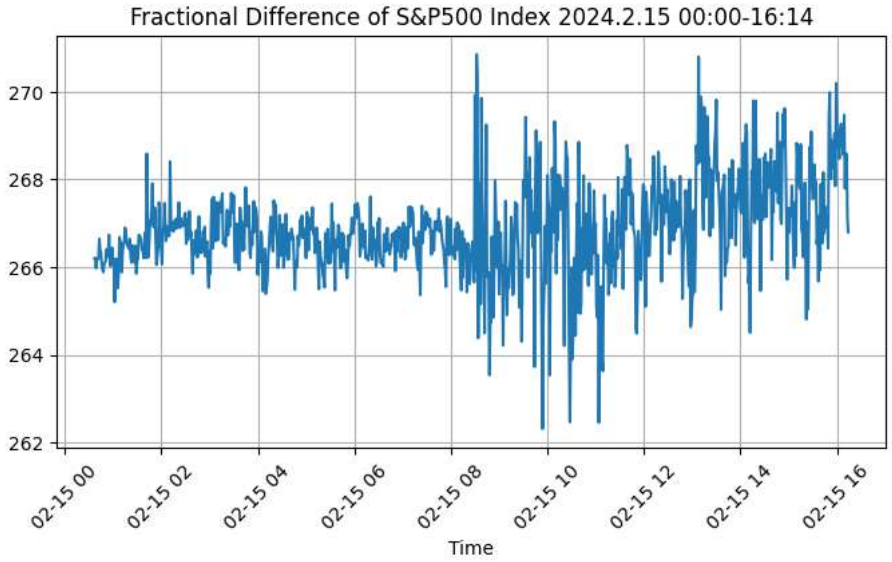
```
20
```

For analysis purpose, we still choose 0.6 as the order so that the cutoff lag would be 36.

```
df1_fdiff = ts_differencing_tau(df1, 0.6, tau)
```

```
plt.figure(figsize=(8, 4))
plt.plot(df1_fdiff)
plt.title('Fractional Difference of S&P500 Index 2024.2.15 00:00-16:14')
plt.xlabel('Time')
plt.grid(True)
plt.xticks(rotation= 45)
```

```
(array([19768.        , 19768.08333333, 19768.16666667, 19768.25      ,
        19768.33333333, 19768.41666667, 19768.5       , 19768.58333333,
        19768.66666667]),
 [Text(19768.0, 0, '02-15 00'),
  Text(19768.083333333332, 0, '02-15 02'),
  Text(19768.166666666668, 0, '02-15 04'),
  Text(19768.25, 0, '02-15 06'),
  Text(19768.333333333332, 0, '02-15 08'),
  Text(19768.416666666668, 0, '02-15 10'),
  Text(19768.5, 0, '02-15 12'),
  Text(19768.583333333332, 0, '02-15 14'),
  Text(19768.666666666668, 0, '02-15 16')])
```



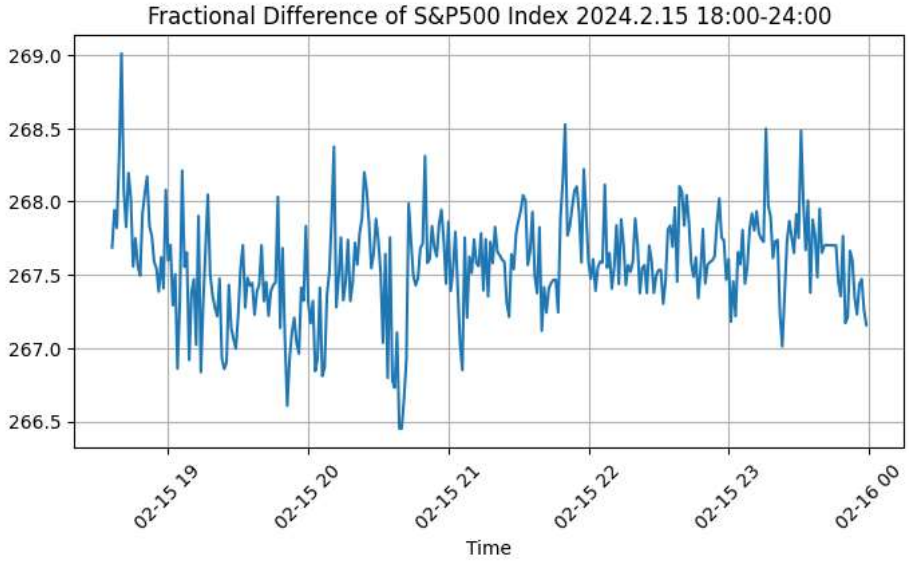

```
# Perform ADF test
print('Results of ADF Test:')
dftest = adfuller(df1_fdiff)
dfoutput = pd.Series(dftest[0:4], index=['Test Statistic','P-value','#Lags Used','Number of Observations Used'])
for key,value in dftest[4].items():
    dfoutput['Critical Value (%s)'%key] = value
print(dfoutput)
```

```
Results of ADF Test:
Test Statistic                -8.024246e+00
P-value                        2.037448e-12
#Lags Used                     4.000000e+00
Number of Observations Used    9.330000e+02
Critical Value (1%)           -3.437378e+00
Critical Value (5%)           -2.864643e+00
Critical Value (10%)          -2.568422e+00
dtype: float64
```

```
df2_fdiff = ts_differencing_tau(df2, 0.6, tau)
```

```
plt.figure(figsize=(8, 4))
plt.plot(df2_fdiff)
plt.title('Fractional Difference of S&P500 Index 2024.2.15 18:00-24:00')
plt.xlabel('Time')
plt.grid(True)
plt.xticks(rotation= 45)
```

```
(array([19768.79166667, 19768.83333333, 19768.875     , 19768.91666667,
        19768.95833333, 19769.        ]),
 [Text(19768.791666666668, 0, '02-15 19'),
  Text(19768.833333333332, 0, '02-15 20'),
  Text(19768.875, 0, '02-15 21'),
  Text(19768.916666666668, 0, '02-15 22'),
  Text(19768.958333333332, 0, '02-15 23'),
  Text(19769.0, 0, '02-16 00')])
```




```
# Perform ADF test
print('Results of ADF Test:')
dftest = adfuller(df2_fdiff)
dfoutput = pd.Series(dftest[0:4], index=['Test Statistic','P-value','#Lags Used','Number of Observations Used'])
for key,value in dftest[4].items():
    dfoutput['Critical Value (%s)'%key] = value
print(dfoutput)
```

```
Results of ADF Test:
Test Statistic                -7.063537e+00
P-value                        5.149908e-10
#Lags Used                     1.000000e+00
Number of Observations Used    3.220000e+02
Critical Value (1%)           -3.450823e+00
Critical Value (5%)           -2.870558e+00
Critical Value (10%)          -2.571575e+00
dtype: float64
```

Step 2 MLP

Setup

We define the label as whether the 15 min return in the future is positive.

```
def add_label(df_in):
    df = df_in.copy()
    df["Ret"] = df["Close"].pct_change()
    df["Ret15_i"] = df["Ret"].rolling(15).apply(lambda x: 100 * (np.prod(1 + x / 100) - 1))
    df["Ret15_i"] = df["Ret15_i"].shift(-15)
    df["Label"] = df["Ret15_i"] > 0
    df["Label"] = df["Label"].astype(int)
    df.drop(columns=['Ret'], inplace=True)
    return df.dropna()

df1_label = add_label(df1)

df1_label.head()
```

	Close	Ret15_i	Label
2024-02-15 00:01:00	5002.376	-1.517235e-04	0
2024-02-15 00:02:00	5002.864	-1.986933e-04	0
2024-02-15 00:03:00	5002.629	-4.995853e-05	0
2024-02-15 00:04:00	5002.876	-8.973710e-05	0
2024-02-15 00:05:00	5002.611	6.142571e-07	1

Next steps: Generate code with df1_label | View recommended plots

```
df1_label.tail()
```

	Close	Ret15_i	Label
2024-02-15 15:55:00	5028.027	0.001101	1
2024-02-15 15:56:00	5028.197	0.001119	1
2024-02-15 15:57:00	5029.177	0.000974	1
2024-02-15 15:58:00	5028.661	0.000831	1
2024-02-15 15:59:00	5030.871	0.000190	1

```
df1_label.Label.value_counts()
```

```
1    543
0    416
Name: Label, dtype: int64
```

```
df2_label = add_label(df2)
df2_label.Label.value_counts()
```

```
0    185
1    160
Name: Label, dtype: int64
```

We can see this label setup is quite balanced.

As we need to compare MLP with CNN on GAF later, we will use the same length of past information.

For demonstration purpose, if we want to use window size 20 in the GAP representation, then we can use 20 data points as length of past information.

Therefore, we simply use the past 19 minutes data plut the current minute data as features.

```
feature_lst = ['Close'] + ["Past_{}min".format(i) for i in range(1, 20)]
print(feature_lst)
```

```
['Close', 'Past_1min', 'Past_2min', 'Past_3min', 'Past_4min', 'Past_5min', 'Past_6min', 'Past_7min', 'Past_8min', 'Past_9min', 'Past_10min', 'Pa
```

```
def add_features(df_in, name='Close', set_zero=False):
    df = df_in.copy()
    for i in range(1, 20):
        feat = feature_lst[i]
        if set_zero:
            df[feat] = df[name].shift(i) - df[name]
        else:
            df[feat] = df[name].shift(i)
    return df.dropna()

df1_feature = add_features(df1_label)

df1_feature.head()
```

	Close	Ret15_i	Label	Past_1min	Past_2min	Past_3min	Past_4min	Past_5min	Past_6min	Past_7min	...	Past_10min
2024-02-15 00:20:00	5002.614	0.000052	1	5002.427	5002.379	5001.870	5001.617	5001.864	5001.873	5001.873	...	5002.047
2024-02-15 00:21:00	5002.614	0.000051	1	5002.614	5002.427	5002.379	5001.870	5001.617	5001.864	5001.873	...	5002.177
2024-02-15 00:22:00	5002.861	0.000002	1	5002.614	5002.614	5002.427	5002.379	5001.870	5001.617	5001.864	...	5001.873
2024-02-15 00:23:00	5003.126	-0.000051	0	5002.861	5002.614	5002.614	5002.427	5002.379	5001.870	5001.617	...	5001.873
2024-02-15 00:24:00	5003.126	-0.000101	0	5003.126	5002.861	5002.614	5002.614	5002.427	5002.379	5001.870	...	5001.873

5 rows × 22 columns

Notice that for the original index level time series, the absolute value is quite large. Given limited training data, we can substract the "Close" from all past values so that neural networks can be trained more easily.

```
df1_feature = add_features(df1_label, set_zero = True)
df1_feature.head()
```

	Close	Ret15_i	Label	Past_1min	Past_2min	Past_3min	Past_4min	Past_5min	Past_6min	Past_7min	...	Past_10min
2024-02-15 00:20:00	5002.614	0.000052	1	-0.187	-0.235	-0.744	-0.997	-0.750	-0.741	-0.741	...	-0.567
2024-02-15 00:21:00	5002.614	0.000051	1	0.000	-0.187	-0.235	-0.744	-0.997	-0.750	-0.741	...	-0.437
2024-02-15 00:22:00	5002.861	0.000002	1	-0.247	-0.247	-0.434	-0.482	-0.991	-1.244	-0.997	...	-0.988
2024-02-15 00:23:00	5003.126	-0.000051	0	-0.265	-0.512	-0.512	-0.699	-0.747	-1.256	-1.509	...	-1.253
2024-02-15 00:24:00	5003.126	-0.000101	0	0.000	-0.265	-0.512	-0.512	-0.699	-0.747	-1.256	...	-1.253

5 rows × 22 columns

For the other two stationary time series, we may choose to keep as it is.

Original Time Series

For the original time series, we have substracted "Close" from the past information and therefore we can exclude it from the features.

```
df1_feature = add_features(df1_label, set_zero=True)
X_train = df1_feature[feature_lst[1:]].astype("float32")
y_train = df1_feature['Label'].astype("float32")

df2_feature = add_features(df2_label, set_zero=True)
X_test = df2_feature[feature_lst[1:]].astype("float32")
y_test = df2_feature['Label'].astype("float32")
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

```
(940, 19) (326, 19) (940,) (326,)
```

We use Keras Tuner for hyperparameter tuning.

```
class MLP_model(HyperModel):
    def build(self, hp):
        # We define a constant activation function of ReLU form. We will not be tuning the activation functions
        act_fun = "relu"

        # We do ask the Keras Tuner to choose whether is best to have a dropout rate after each hidden layer of 0, 0.1
        n_dropout = hp.Choice("n_dropout", values=[0.0, 0.20, 0.30])

        model = tf.keras.models.Sequential()

        # Now, we will use a loop to let the tuner choose the number of layers that is best for the model between 1 a
        for i in range(1, hp.Int("num_layers", 1, 5)):
            # Within this loop, we will also ask the tuner to decide the optimal number of units that each of the se
            model.add(
                tf.keras.layers.Dense(
                    units=hp.Int(
                        "units_dense_" + str(i), min_value=10, max_value=50, step=10
                    ),
                    activation=act_fun,
                )
            )
            model.add(tf.keras.layers.Dropout(n_dropout))

        model.add(tf.keras.layers.Dense(units=1, activation="sigmoid"))

        # As was the case with the activation function, there is no tuning on the learning rate, nor the optimizer, altho
        hp_lr = 1e-5
        adam = tf.keras.optimizers.Adam(learning_rate=hp_lr)
        model.compile(optimizer=adam, loss="binary_crossentropy", metrics=["accuracy"])

        return model
```

We use 30% of the training set as validation set during hyperparameter tuning, and the best set of hyperparameters is selected according to the minimum validation loss.

```
# First, we clear the session just to make sure our seeds are correctly working and replicability is achieved
K.clear_session()
tf.random.set_seed(1234)

# Then, we call the model and perform the tuning:
hypermodel = MLP_model()
tuner = kt.Hyperband(
    hypermodel,
    objective=kt.Objective("val_loss", direction="min"),
    overwrite=True,
    max_epochs=30,
    seed=1234,
    directory=os.path.normpath("/content/"),
)

# Let's run the tuner!
tuner.search(X_train, y_train, validation_split=0.3)
best_hps = tuner.get_best_hyperparameters(num_trials=1)[0]
```

```
Trial 90 Complete [00h 00m 07s]
val_loss: 0.6406893730163574

Best val_loss So Far: 0.6406893730163574
Total elapsed time: 00h 04m 46s
```

```
best_hps.values
```

```
{'n_dropout': 0.0,
 'num_layers': 4,
 'units_dense_1': 50,
 'units_dense_2': 30,
```

```
        'units_dense_3': 10,
        'units_dense_4': 20,
        'tuner/epochs': 30,
        'tuner/initial_epoch': 0,
        'tuner/bracket': 0,
        'tuner/round': 0}


K.clear_session()
tf.random.set_seed(1234)
model  =  tuner.hypermodel.build(best_hps)
es  =  tf.keras.callbacks.EarlyStopping(
        monitor="val_loss",  mode="min",  verbose=1,  patience=20,  restore_best_weights=True
)

#  fit  the  model
model.fit(
        X_train,
        y_train,
        validation_split=0.2,
        epochs=500,
        batch_size=32,
        verbose=2,
        callbacks=[es],
)
```

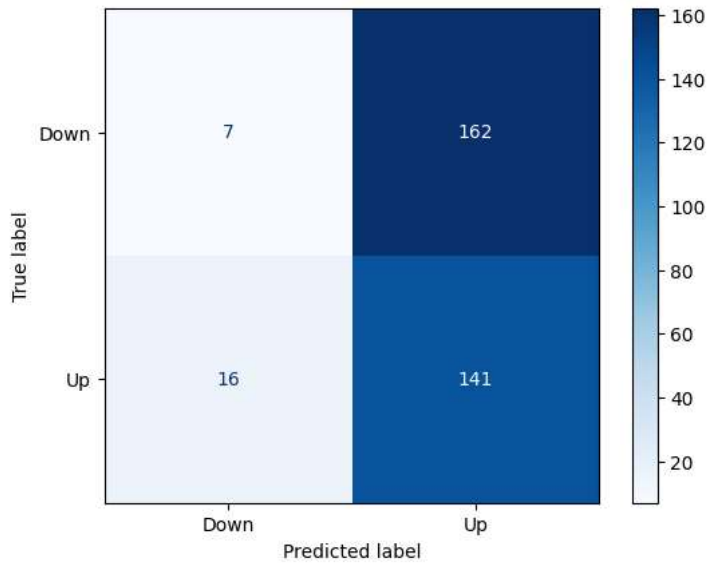
```
24/24 - 0s - loss: 0.6832 - accuracy: 0.5532 - val_loss: 0.6465 - val_accuracy: 0.6383 - 90ms/epoch - 4ms/step
Epoch 34/500
24/24 - 0s - loss: 0.6821 - accuracy: 0.5519 - val_loss: 0.6462 - val_accuracy: 0.6383 - 81ms/epoch - 3ms/step
Epoch 35/500
24/24 - 0s - loss: 0.6809 - accuracy: 0.5545 - val_loss: 0.6459 - val_accuracy: 0.6383 - 85ms/epoch - 4ms/step
Epoch 36/500
24/24 - 0s - loss: 0.6799 - accuracy: 0.5545 - val_loss: 0.6456 - val_accuracy: 0.6383 - 83ms/epoch - 3ms/step
Epoch 37/500
24/24 - 0s - loss: 0.6789 - accuracy: 0.5519 - val_loss: 0.6455 - val_accuracy: 0.6383 - 82ms/epoch - 3ms/step
Epoch 38/500
24/24 - 0s - loss: 0.6779 - accuracy: 0.5519 - val_loss: 0.6454 - val_accuracy: 0.6383 - 91ms/epoch - 4ms/step
Epoch 39/500
24/24 - 0s - loss: 0.6769 - accuracy: 0.5519 - val_loss: 0.6453 - val_accuracy: 0.6383 - 80ms/epoch - 3ms/step
Epoch 40/500
24/24 - 0s - loss: 0.6761 - accuracy: 0.5519 - val_loss: 0.6453 - val_accuracy: 0.6330 - 78ms/epoch - 3ms/step
Epoch 41/500
24/24 - 0s - loss: 0.6752 - accuracy: 0.5505 - val_loss: 0.6453 - val_accuracy: 0.6277 - 91ms/epoch - 4ms/step
Epoch 42/500
24/24 - 0s - loss: 0.6743 - accuracy: 0.5545 - val_loss: 0.6455 - val_accuracy: 0.6383 - 87ms/epoch - 4ms/step
Epoch 43/500
24/24 - 0s - loss: 0.6734 - accuracy: 0.5532 - val_loss: 0.6457 - val_accuracy: 0.6330 - 88ms/epoch - 4ms/step
Epoch 44/500
24/24 - 0s - loss: 0.6727 - accuracy: 0.5585 - val_loss: 0.6459 - val_accuracy: 0.6383 - 80ms/epoch - 3ms/step
Epoch 45/500
24/24 - 0s - loss: 0.6719 - accuracy: 0.5598 - val_loss: 0.6463 - val_accuracy: 0.6383 - 80ms/epoch - 3ms/step
Epoch 46/500
24/24 - 0s - loss: 0.6711 - accuracy: 0.5612 - val_loss: 0.6467 - val_accuracy: 0.6436 - 77ms/epoch - 3ms/step
Epoch 47/500
24/24 - 0s - loss: 0.6703 - accuracy: 0.5612 - val_loss: 0.6470 - val_accuracy: 0.6543 - 77ms/epoch - 3ms/step
Epoch 48/500
24/24 - 0s - loss: 0.6696 - accuracy: 0.5625 - val_loss: 0.6475 - val_accuracy: 0.6383 - 78ms/epoch - 3ms/step
Epoch 49/500
24/24 - 0s - loss: 0.6689 - accuracy: 0.5612 - val_loss: 0.6481 - val_accuracy: 0.6330 - 88ms/epoch - 4ms/step
Epoch 50/500
24/24 - 0s - loss: 0.6683 - accuracy: 0.5598 - val_loss: 0.6487 - val_accuracy: 0.6330 - 79ms/epoch - 3ms/step
Epoch 51/500
24/24 - 0s - loss: 0.6676 - accuracy: 0.5585 - val_loss: 0.6492 - val_accuracy: 0.6277 - 93ms/epoch - 4ms/step
Epoch 52/500
24/24 - 0s - loss: 0.6670 - accuracy: 0.5572 - val_loss: 0.6496 - val_accuracy: 0.6170 - 82ms/epoch - 3ms/step
Epoch 53/500
24/24 - 0s - loss: 0.6665 - accuracy: 0.5585 - val_loss: 0.6504 - val_accuracy: 0.6117 - 99ms/epoch - 4ms/step
Epoch 54/500
24/24 - 0s - loss: 0.6659 - accuracy: 0.5585 - val_loss: 0.6511 - val_accuracy: 0.6064 - 80ms/epoch - 3ms/step
Epoch 55/500
24/24 - 0s - loss: 0.6654 - accuracy: 0.5585 - val_loss: 0.6520 - val_accuracy: 0.6011 - 94ms/epoch - 4ms/step
Epoch 56/500
24/24 - 0s - loss: 0.6649 - accuracy: 0.5612 - val_loss: 0.6525 - val_accuracy: 0.5957 - 83ms/epoch - 3ms/step
Epoch 57/500
24/24 - 0s - loss: 0.6644 - accuracy: 0.5598 - val_loss: 0.6531 - val_accuracy: 0.5904 - 93ms/epoch - 4ms/step
Epoch 58/500
24/24 - 0s - loss: 0.6640 - accuracy: 0.5625 - val_loss: 0.6539 - val_accuracy: 0.5904 - 86ms/epoch - 4ms/step
Epoch 59/500
24/24 - 0s - loss: 0.6636 - accuracy: 0.5652 - val_loss: 0.6549 - val_accuracy: 0.5904 - 93ms/epoch - 4ms/step
Epoch 60/500
Restoring model weights from the end of the best epoch: 40.
24/24 - 0s - loss: 0.6631 - accuracy: 0.5665 - val_loss: 0.6554 - val_accuracy: 0.5957 - 95ms/epoch - 4ms/step
Epoch 60: early stopping
<keras.src.callbacks.History at 0x7ab6402661d0>
```

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 50)                1000

 dropout (Dropout)           (None, 50)                0

 dense_1 (Dense)             (None, 30)                1530

 dropout_1 (Dropout)         (None, 30)                0

 dense_2 (Dense)             (None, 10)                310

 dropout_2 (Dropout)         (None, 10)                0

 dense_3 (Dense)             (None, 1)                 11

=================================================================
Total params: 2851 (11.14 KB)
Trainable params: 2851 (11.14 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
acc = model.evaluate(x=X_test, y=y_test)
print("Accuracy over test: {:.2f}%".format(acc[1] * 100))

y_prob = model.predict(X_test)
y_pred = np.round(y_prob)
cm = metrics.confusion_matrix(y_test, y_pred)
cm = metrics.ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=["Down", "Up"])
cm.plot(values_format="", cmap="Blues")
print(classification_report(y_test, y_pred))
```

```
11/11 [==============================] - 0s 3ms/step - loss: 0.7067 - accuracy: 0.4540
Accuracy over test: 45.40%
11/11 [==============================] - 0s 2ms/step
              precision    recall  f1-score   support

         0.0       0.30      0.04      0.07       169
         1.0       0.47      0.90      0.61       157

    accuracy                           0.45       326
   macro avg       0.38      0.47      0.34       326
weighted avg       0.38      0.45      0.33       326
```



Due to the nature of the problem setup, this hyperparameter tuning process is very unstable. We found that the following network structure can produce good results. Therefore, we will stick with this structure, and will try it for the other time series as well.

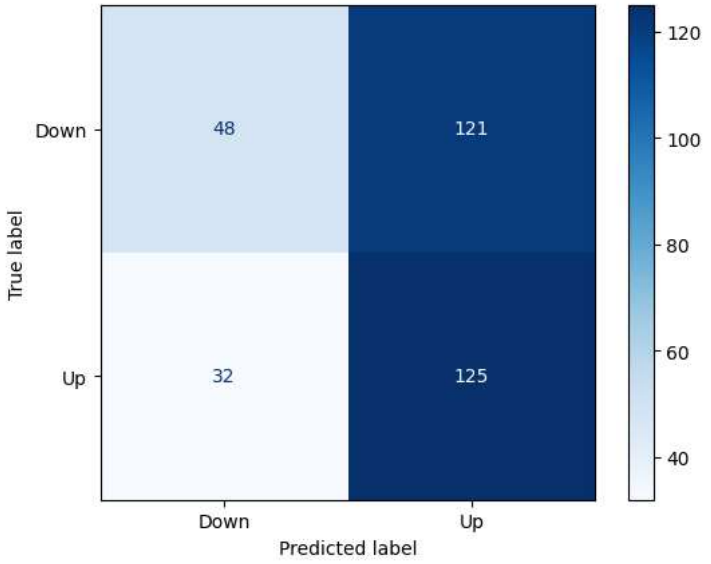
```
best_hps_bk = {'n_dropout': 0.0,
 'num_layers': 5,
 'units_dense_1': 10,
 'units_dense_2': 50,
 'units_dense_3': 50,
 'units_dense_4': 10,
 'tuner/epochs': 30,
 'tuner/initial_epoch': 10,
 'tuner/bracket': 3,
 'tuner/round': 3
}


def build_MLP_model():
        tf.keras.backend.clear_session()
        tf.random.set_seed(1234)
        model = tf.keras.Sequential()
        model.add(tf.keras.layers.Dense(units=10, activation='relu'))
        model.add(tf.keras.layers.Dense(units=50, activation='relu'))
        model.add(tf.keras.layers.Dense(units=50, activation='relu'))
        model.add(tf.keras.layers.Dense(units=10, activation='relu'))
        model.add(tf.keras.layers.Dense(units=1, activation="sigmoid"))
        hp_lr = 1e-5
        model.compile(loss="binary_crossentropy", optimizer="Adam", metrics=["accuracy"])
        return model


model = build_MLP_model()
es = tf.keras.callbacks.EarlyStopping(
        monitor="val_loss", mode="min", verbose=1, patience=20, restore_best_weights=True
)

# fit the model
model.fit(
        X_train,
        y_train,
        validation_split=0.3,
        epochs=500,
        batch_size=32,
        verbose=2,
        callbacks=[es],
)
```

```
Epoch 1/500
21/21 - 1s - loss: 0.7066 - accuracy: 0.5015 - val_loss: 0.7200 - val_accuracy: 0.5248 - 1s/epoch - 66ms/step
Epoch 2/500
21/21 - 0s - loss: 0.6617 - accuracy: 0.5866 - val_loss: 0.7552 - val_accuracy: 0.5177 - 86ms/epoch - 4ms/step
Epoch 3/500
21/21 - 0s - loss: 0.6522 - accuracy: 0.6018 - val_loss: 0.6904 - val_accuracy: 0.5780 - 90ms/epoch - 4ms/step
Epoch 4/500
21/21 - 0s - loss: 0.6499 - accuracy: 0.5942 - val_loss: 0.7769 - val_accuracy: 0.5284 - 91ms/epoch - 4ms/step
Epoch 5/500
21/21 - 0s - loss: 0.6389 - accuracy: 0.6292 - val_loss: 0.6961 - val_accuracy: 0.5674 - 82ms/epoch - 4ms/step
Epoch 6/500
21/21 - 0s - loss: 0.6366 - accuracy: 0.6277 - val_loss: 0.7759 - val_accuracy: 0.5177 - 89ms/epoch - 4ms/step
Epoch 7/500
21/21 - 0s - loss: 0.6370 - accuracy: 0.6277 - val_loss: 0.7660 - val_accuracy: 0.5248 - 94ms/epoch - 4ms/step
Epoch 8/500
21/21 - 0s - loss: 0.6292 - accuracy: 0.6550 - val_loss: 0.7696 - val_accuracy: 0.5248 - 80ms/epoch - 4ms/step
Epoch 9/500
21/21 - 0s - loss: 0.6266 - accuracy: 0.6413 - val_loss: 0.8624 - val_accuracy: 0.5000 - 141ms/epoch - 7ms/step
Epoch 10/500
21/21 - 0s - loss: 0.6289 - accuracy: 0.6353 - val_loss: 0.7355 - val_accuracy: 0.5355 - 134ms/epoch - 6ms/step
Epoch 11/500
21/21 - 0s - loss: 0.6181 - accuracy: 0.6581 - val_loss: 0.7896 - val_accuracy: 0.5213 - 130ms/epoch - 6ms/step
Epoch 12/500
21/21 - 0s - loss: 0.6126 - accuracy: 0.6641 - val_loss: 0.7597 - val_accuracy: 0.5213 - 119ms/epoch - 6ms/step
Epoch 13/500
21/21 - 0s - loss: 0.6106 - accuracy: 0.6535 - val_loss: 0.7713 - val_accuracy: 0.5284 - 151ms/epoch - 7ms/step
Epoch 14/500
21/21 - 0s - loss: 0.6058 - accuracy: 0.6672 - val_loss: 0.7916 - val_accuracy: 0.5142 - 109ms/epoch - 5ms/step
Epoch 15/500
21/21 - 0s - loss: 0.6049 - accuracy: 0.6702 - val_loss: 0.7509 - val_accuracy: 0.5213 - 108ms/epoch - 5ms/step
Epoch 16/500
21/21 - 0s - loss: 0.5999 - accuracy: 0.6611 - val_loss: 0.7864 - val_accuracy: 0.5248 - 112ms/epoch - 5ms/step
Epoch 17/500
21/21 - 0s - loss: 0.5959 - accuracy: 0.6733 - val_loss: 0.8336 - val_accuracy: 0.5142 - 178ms/epoch - 8ms/step
Epoch 18/500
21/21 - 0s - loss: 0.5972 - accuracy: 0.6641 - val_loss: 0.7850 - val_accuracy: 0.5355 - 123ms/epoch - 6ms/step
Epoch 19/500
21/21 - 0s - loss: 0.5889 - accuracy: 0.6793 - val_loss: 0.7667 - val_accuracy: 0.5248 - 171ms/epoch - 8ms/step
Epoch 20/500
21/21 - 0s - loss: 0.5806 - accuracy: 0.6930 - val_loss: 0.8211 - val_accuracy: 0.5284 - 180ms/epoch - 9ms/step
Epoch 21/500
21/21 - 0s - loss: 0.5764 - accuracy: 0.7097 - val_loss: 0.7843 - val_accuracy: 0.4858 - 140ms/epoch - 7ms/step
Epoch 22/500
21/21 - 0s - loss: 0.5740 - accuracy: 0.6915 - val_loss: 0.8826 - val_accuracy: 0.5177 - 168ms/epoch - 8ms/step
```

```
Epoch 23/500
Restoring model weights from the end of the best epoch: 3.
21/21 - 0s - loss: 0.5722 - accuracy: 0.6976 - val_loss: 0.8454 - val_accuracy: 0.5071 - 183ms/epoch - 9ms/step
Epoch 23: early stopping
<keras.src.callbacks.History at 0x7ab6399d1990>
```

```
acc = model.evaluate(x=X_test, y=y_test)
print("Accuracy over test: {:.2f}%".format(acc[1] * 100))

y_prob = model.predict(X_test)
y_pred = np.round(y_prob)
cm = metrics.confusion_matrix(y_test, y_pred)
cm = metrics.ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=["Down", "Up"])
cm.plot(values_format="", cmap="Blues")
print(classification_report(y_test, y_pred))
```

```
11/11 [==============================] - 0s 3ms/step - loss: 0.7098 - accuracy: 0.5307
Accuracy over test: 53.07%
11/11 [==============================] - 0s 2ms/step
              precision    recall  f1-score   support

         0.0       0.60      0.28      0.39       169
         1.0       0.51      0.80      0.62       157

    accuracy                           0.53       326
   macro avg       0.55      0.54      0.50       326
weighted avg       0.56      0.53      0.50       326
```



Stationary Time Series

```
df1_label_diff = df1_diff.copy()
df1_label_diff['Label'] = df1_label[df1_label.index.isin(df1_diff.index)]['Label']
df1_label_diff.dropna(inplace=True)

df2_label_diff = df2_diff.copy()
df2_label_diff['Label'] = df2_label[df2_label.index.isin(df2_diff.index)]['Label']
df2_label_diff.dropna(inplace=True)
```

```
df1_feature_diff = add_features(df1_label_diff)
X_train_diff = df1_feature_diff[feature_lst].astype("float32")
y_train_diff = df1_feature_diff['Label'].astype("float32")

df2_feature_diff = add_features(df2_label_diff)
X_test_diff = df2_feature_diff[feature_lst].astype("float32")
y_test_diff = df2_feature_diff['Label'].astype("float32")
print(X_train_diff.shape, X_test_diff.shape, y_train_diff.shape, y_test_diff.shape)
```

```
(939, 20) (325, 20) (939,) (325,)
```

```
# First, we clear the session just to make sure our seeds are correctly working and replicability is achieved
K.clear_session()

# Then, we call the model and perform the tuning:
hypermodel = MLP_model()
tuner = kt.Hyperband(
        hypermodel,
        objective=kt.Objective("val_loss", direction="min"),
        overwrite=True,
        max_epochs=30,
        seed=1234,
        directory=os.path.normpath("/content/"),
)

# Let's run the tuner! (Warning: this could take time)
tuner.search(X_train_diff, y_train_diff, validation_split=0.3)
best_hps_diff = tuner.get_best_hyperparameters(num_trials=1)[0]
```

```
Trial 90 Complete [00h 00m 06s]
val_loss: 0.6689453721046448

Best val_loss So Far: 0.6381919384002686
Total elapsed time: 00h 05m 17s
```

```
best_hps_diff.values
```

```
{'n_dropout': 0.0,
 'num_layers': 5,
 'units_dense_1': 10,
 'units_dense_2': 50,
 'units_dense_3': 50,
 'units_dense_4': 10,
 'tuner/epochs': 2,
 'tuner/initial_epoch': 0,
 'tuner/bracket': 3,
 'tuner/round': 0}
```

```
model_diff = tuner.hypermodel.build(best_hps_diff)
es = tf.keras.callbacks.EarlyStopping(
        monitor="val_loss", mode="min", verbose=1, patience=20, restore_best_weights=True
)

# fit the model
model_diff.fit(
        X_train_diff,
        y_train_diff,
        validation_split=0.2,
        epochs=500,
        batch_size=32,
        verbose=2,
        callbacks=[es],
)
```

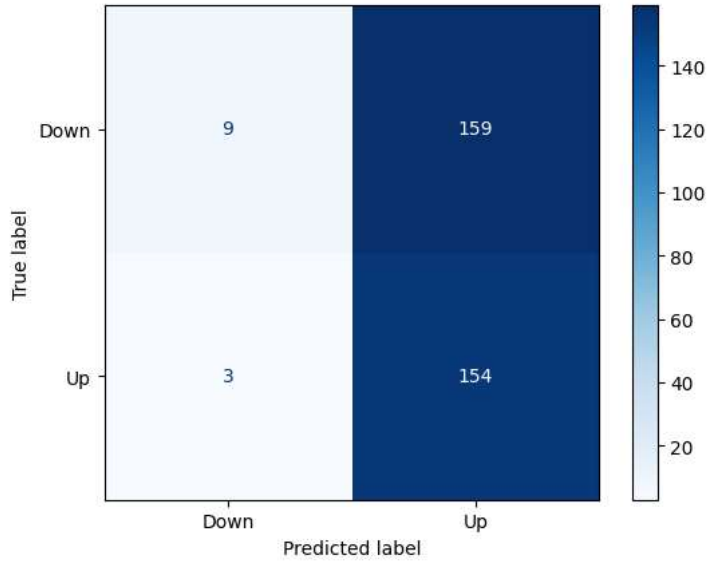
```
Epoch 37/500
24/24 - 0s - loss: 0.6948 - accuracy: 0.5020 - val_loss: 0.7248 - val_accuracy: 0.4362 - 79ms/epoch - 3ms/step
Epoch 38/500
24/24 - 0s - loss: 0.6946 - accuracy: 0.5033 - val_loss: 0.7242 - val_accuracy: 0.4468 - 78ms/epoch - 3ms/step
Epoch 39/500
24/24 - 0s - loss: 0.6945 - accuracy: 0.4993 - val_loss: 0.7237 - val_accuracy: 0.4574 - 106ms/epoch - 4ms/step
Epoch 40/500
24/24 - 0s - loss: 0.6943 - accuracy: 0.4993 - val_loss: 0.7231 - val_accuracy: 0.4521 - 80ms/epoch - 3ms/step
Epoch 41/500
24/24 - 0s - loss: 0.6941 - accuracy: 0.4993 - val_loss: 0.7226 - val_accuracy: 0.4521 - 80ms/epoch - 3ms/step
Epoch 42/500
24/24 - 0s - loss: 0.6940 - accuracy: 0.5033 - val_loss: 0.7221 - val_accuracy: 0.4574 - 82ms/epoch - 3ms/step
Epoch 43/500
24/24 - 0s - loss: 0.6938 - accuracy: 0.4980 - val_loss: 0.7216 - val_accuracy: 0.4574 - 81ms/epoch - 3ms/step
Epoch 44/500
24/24 - 0s - loss: 0.6937 - accuracy: 0.4980 - val_loss: 0.7212 - val_accuracy: 0.4574 - 91ms/epoch - 4ms/step
Epoch 45/500
24/24 - 0s - loss: 0.6935 - accuracy: 0.5007 - val_loss: 0.7207 - val_accuracy: 0.4574 - 97ms/epoch - 4ms/step
Epoch 46/500
24/24 - 0s - loss: 0.6933 - accuracy: 0.5020 - val_loss: 0.7200 - val_accuracy: 0.4521 - 84ms/epoch - 3ms/step
Epoch 47/500
24/24 - 0s - loss: 0.6932 - accuracy: 0.5033 - val_loss: 0.7197 - val_accuracy: 0.4468 - 94ms/epoch - 4ms/step
Epoch 48/500
24/24 - 0s - loss: 0.6930 - accuracy: 0.5033 - val_loss: 0.7192 - val_accuracy: 0.4415 - 95ms/epoch - 4ms/step
Epoch 49/500
24/24 - 0s - loss: 0.6929 - accuracy: 0.5033 - val_loss: 0.7187 - val_accuracy: 0.4415 - 99ms/epoch - 4ms/step
```

```
model_diff.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_4 (Dense)             (None, 10)                210

 dropout_3 (Dropout)         (None, 10)                0

 dense_5 (Dense)             (None, 50)                550

 dropout_4 (Dropout)         (None, 50)                0

 dense_6 (Dense)             (None, 50)                2550

 dropout_5 (Dropout)         (None, 50)                0

 dense_7 (Dense)             (None, 10)                510

 dropout_6 (Dropout)         (None, 10)                0

 dense_8 (Dense)             (None, 1)                 11

=================================================================
Total params: 3831 (14.96 KB)
Trainable params: 3831 (14.96 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
acc = model_diff.evaluate(x=X_test_diff, y=y_test_diff)
print("Accuracy over test: {:.2f}%".format(acc[1] * 100))

y_prob = model_diff.predict(X_test_diff)
y_pred = np.round(y_prob)
cm = metrics.confusion_matrix(y_test_diff, y_pred)
cm = metrics.ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=["Down", "Up"])
cm.plot(values_format="", cmap="Blues")
print(classification_report(y_test_diff, y_pred))
```

```
11/11 [==============================] - 0s 2ms/step - loss: 0.6976 - accuracy: 0.5015
Accuracy over test: 50.15%
11/11 [==============================] - 0s 2ms/step
              precision    recall  f1-score   support

         0.0       0.75      0.05      0.10       168
         1.0       0.49      0.98      0.66       157

    accuracy                           0.50       325
   macro avg       0.62      0.52      0.38       325
weighted avg       0.63      0.50      0.37       325
```



Try to train an MLP with default structure.

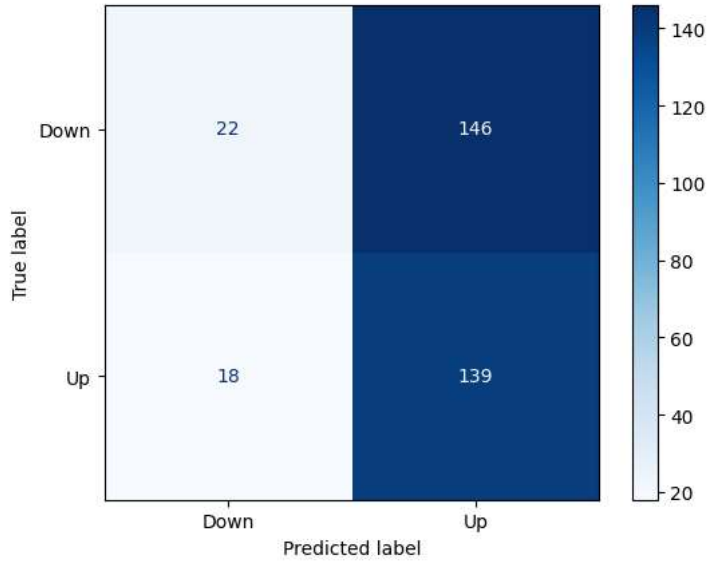
```
model_diff = build_MLP_model()
es = tf.keras.callbacks.EarlyStopping(
        monitor="val_loss", mode="min", verbose=1, patience=30, restore_best_weights=True
)

# fit the model
model_diff.fit(
        X_train_diff,
        y_train_diff,
        validation_split=0.3,
        epochs=500,
        batch_size=32,
        verbose=2,
        callbacks=[es],
)
```

```
21/21 - 0s - loss: 0.6775 - accuracy: 0.5814 - val_loss: 0.6967 - val_accuracy: 0.5071 - 97ms/epoch - 5ms/step
Epoch 6/500
21/21 - 0s - loss: 0.6714 - accuracy: 0.5936 - val_loss: 0.6946 - val_accuracy: 0.5142 - 87ms/epoch - 4ms/step
Epoch 7/500
21/21 - 0s - loss: 0.6648 - accuracy: 0.5997 - val_loss: 0.6999 - val_accuracy: 0.5284 - 77ms/epoch - 4ms/step
Epoch 8/500
21/21 - 0s - loss: 0.6581 - accuracy: 0.6149 - val_loss: 0.7043 - val_accuracy: 0.5284 - 79ms/epoch - 4ms/step
Epoch 9/500
21/21 - 0s - loss: 0.6496 - accuracy: 0.6149 - val_loss: 0.7043 - val_accuracy: 0.5319 - 84ms/epoch - 4ms/step
Epoch 10/500
21/21 - 0s - loss: 0.6430 - accuracy: 0.6240 - val_loss: 0.7270 - val_accuracy: 0.5035 - 88ms/epoch - 4ms/step
Epoch 11/500
21/21 - 0s - loss: 0.6303 - accuracy: 0.6423 - val_loss: 0.7191 - val_accuracy: 0.5461 - 78ms/epoch - 4ms/step
Epoch 12/500
21/21 - 0s - loss: 0.6230 - accuracy: 0.6469 - val_loss: 0.7418 - val_accuracy: 0.5177 - 81ms/epoch - 4ms/step
Epoch 13/500
21/21 - 0s - loss: 0.6127 - accuracy: 0.6591 - val_loss: 0.7351 - val_accuracy: 0.5426 - 80ms/epoch - 4ms/step
Epoch 14/500
21/21 - 0s - loss: 0.6027 - accuracy: 0.6697 - val_loss: 0.7362 - val_accuracy: 0.5461 - 77ms/epoch - 4ms/step
Epoch 15/500
21/21 - 0s - loss: 0.5945 - accuracy: 0.6743 - val_loss: 0.8046 - val_accuracy: 0.5035 - 80ms/epoch - 4ms/step
```

```
21/21 - 0s - loss: 0.5489 - accuracy: 0.7093 - val_loss: 0.8789 - val_accuracy: 0.5213 - 85ms/epoch - 4ms/step
Epoch 21/500
21/21 - 0s - loss: 0.5421 - accuracy: 0.7184 - val_loss: 0.8940 - val_accuracy: 0.5248 - 83ms/epoch - 4ms/step
Epoch 22/500
21/21 - 0s - loss: 0.5316 - accuracy: 0.7169 - val_loss: 0.8651 - val_accuracy: 0.5319 - 88ms/epoch - 4ms/step
Epoch 23/500
21/21 - 0s - loss: 0.5274 - accuracy: 0.7352 - val_loss: 0.8815 - val_accuracy: 0.5319 - 90ms/epoch - 4ms/step
Epoch 24/500
21/21 - 0s - loss: 0.5175 - accuracy: 0.7352 - val_loss: 0.9019 - val_accuracy: 0.5213 - 86ms/epoch - 4ms/step
Epoch 25/500
21/21 - 0s - loss: 0.5084 - accuracy: 0.7428 - val_loss: 0.9839 - val_accuracy: 0.5106 - 84ms/epoch - 4ms/step
Epoch 26/500
21/21 - 0s - loss: 0.5022 - accuracy: 0.7443 - val_loss: 0.9630 - val_accuracy: 0.5106 - 87ms/epoch - 4ms/step
Epoch 27/500
21/21 - 0s - loss: 0.4924 - accuracy: 0.7626 - val_loss: 0.9874 - val_accuracy: 0.5035 - 80ms/epoch - 4ms/step
Epoch 28/500
21/21 - 0s - loss: 0.4854 - accuracy: 0.7686 - val_loss: 1.0048 - val_accuracy: 0.5142 - 79ms/epoch - 4ms/step
Epoch 29/500
21/21 - 0s - loss: 0.4774 - accuracy: 0.7793 - val_loss: 0.9896 - val_accuracy: 0.5355 - 91ms/epoch - 4ms/step
Epoch 30/500
21/21 - 0s - loss: 0.4670 - accuracy: 0.7823 - val_loss: 1.0183 - val_accuracy: 0.5177 - 83ms/epoch - 4ms/step
Epoch 31/500
21/21 - 0s - loss: 0.4604 - accuracy: 0.7854 - val_loss: 1.0396 - val_accuracy: 0.5319 - 86ms/epoch - 4ms/step
Epoch 32/500
Restoring model weights from the end of the best epoch: 2.
21/21 - 0s - loss: 0.4567 - accuracy: 0.7900 - val_loss: 1.0538 - val_accuracy: 0.5177 - 115ms/epoch - 5ms/step
Epoch 32: early stopping
<keras.src.callbacks.History at 0x7ab638e1ab00>
```

```
acc = model_diff.evaluate(x=X_test_diff, y=y_test_diff)
print("Accuracy over test: {:.2f}%".format(acc[1] * 100))

y_prob = model_diff.predict(X_test_diff)
y_pred = np.round(y_prob)
cm = metrics.confusion_matrix(y_test_diff, y_pred)
cm = metrics.ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=["Down", "Up"])
cm.plot(values_format="", cmap="Blues")
print(classification_report(y_test_diff, y_pred))
```

```
11/11 [==============================] - 0s 3ms/step - loss: 0.6938 - accuracy: 0.4954
Accuracy over test: 49.54%
11/11 [==============================] - 0s 3ms/step
              precision    recall  f1-score   support

         0.0       0.55      0.13      0.21       168
         1.0       0.49      0.89      0.63       157

    accuracy                           0.50       325
   macro avg       0.52      0.51      0.42       325
weighted avg       0.52      0.50      0.41       325
```



Fractional Differencing

```
df1_label_fdiff = df1_fdiff.copy()
df1_label_fdiff['Label'] = df1_label[df1_label.index.isin(df1_fdiff.index)]['Label']
df1_label_fdiff.dropna(inplace=True)

df2_label_fdiff = df2_fdiff.copy()
df2_label_fdiff['Label'] = df2_label[df2_label.index.isin(df2_fdiff.index)]['Label']
df2_label_fdiff.dropna(inplace=True)


df1_feature_fdiff = add_features(df1_label_fdiff)
X_train_fdiff = df1_feature_fdiff[feature_lst].astype("float32")
y_train_fdiff = df1_feature_fdiff['Label'].astype("float32")

df2_feature_fdiff = add_features(df2_label_fdiff)
X_test_fdiff = df2_feature_fdiff[feature_lst].astype("float32")
y_test_fdiff = df2_feature_fdiff['Label'].astype("float32")
print(X_train_fdiff.shape, X_test_fdiff.shape, y_train_fdiff.shape, y_test_fdiff.shape)
```

```
(904, 20) (290, 20) (904,) (290,)
```

```
# First, we clear the session just to make sure our seeds are correctly working and replicability is achieved
K.clear_session()

# Then, we call the model and perform the tuning:
hypermodel = MLP_model()
tuner = kt.Hyperband(
        hypermodel,
        objective=kt.Objective("val_loss", direction="min"),
        overwrite=True,
        max_epochs=30,
        seed=1234,
        directory=os.path.normpath("/content/"),
)

# Let's run the tuner! (Warning: this could take time)
tuner.search(X_train_fdiff, y_train_fdiff, validation_split=0.3)
best_hps_fdiff = tuner.get_best_hyperparameters(num_trials=1)[0]
```

```
Trial 90 Complete [00h 00m 07s]
val_loss: 0.6835712790489197

Best val_loss So Far: 0.649255096912384
Total elapsed time: 00h 05m 57s
```

```
best_hps_fdiff.values
```

```
{'n_dropout': 0.2,
 'num_layers': 5,
 'units_dense_1': 40,
 'units_dense_2': 10,
 'units_dense_3': 50,
 'units_dense_4': 10,
 'tuner/epochs': 30,
 'tuner/initial_epoch': 10,
 'tuner/bracket': 3,
 'tuner/round': 3,
 'tuner/trial_id': '0047'}
```

```
model_fdiff = tuner.hypermodel.build(best_hps_fdiff)
es = tf.keras.callbacks.EarlyStopping(
        monitor="val_loss", mode="min", verbose=1, patience=20, restore_best_weights=True
)

# fit the model
model_fdiff.fit(
        X_train_fdiff,
        y_train_fdiff,
        validation_split=0.3,
        epochs=500,
        batch_size=32,
        verbose=2,
        callbacks=[es],
)
```

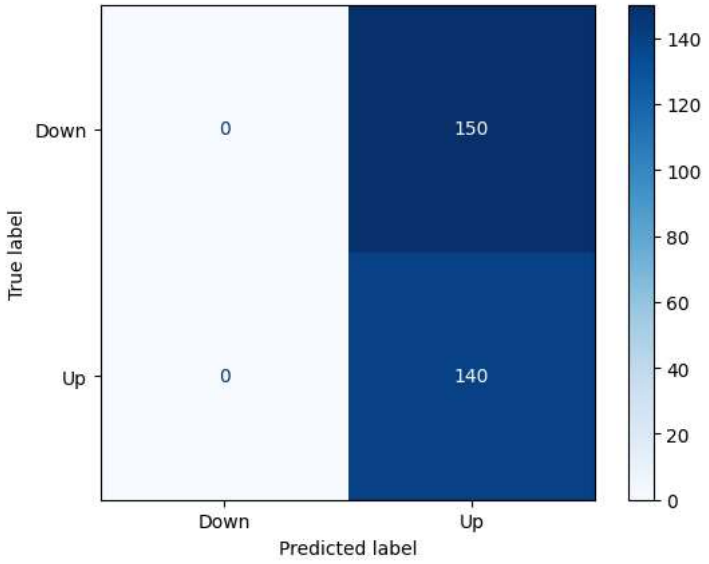
```
Epoch 28/500
20/20 - 0s - loss: 22.6317 - accuracy: 0.4858 - val_loss: 1.8714 - val_accuracy: 0.3566 - 92ms/epoch - 5ms/step
Epoch 29/500
20/20 - 0s - loss: 22.2866 - accuracy: 0.4794 - val_loss: 0.9569 - val_accuracy: 0.3566 - 94ms/epoch - 5ms/step
Epoch 30/500
20/20 - 0s - loss: 22.3913 - accuracy: 0.4715 - val_loss: 0.6701 - val_accuracy: 0.6397 - 131ms/epoch - 7ms/step
Epoch 31/500
20/20 - 0s - loss: 19.7266 - accuracy: 0.4430 - val_loss: 0.6925 - val_accuracy: 0.6434 - 83ms/epoch - 4ms/step
Epoch 32/500
20/20 - 0s - loss: 19.5566 - accuracy: 0.5000 - val_loss: 0.7137 - val_accuracy: 0.6434 - 82ms/epoch - 4ms/step
Epoch 33/500
20/20 - 0s - loss: 19.8053 - accuracy: 0.4573 - val_loss: 0.7386 - val_accuracy: 0.6434 - 89ms/epoch - 4ms/step
Epoch 34/500
20/20 - 0s - loss: 17.4580 - accuracy: 0.4873 - val_loss: 0.7664 - val_accuracy: 0.6434 - 82ms/epoch - 4ms/step
Epoch 35/500
20/20 - 0s - loss: 17.4593 - accuracy: 0.5032 - val_loss: 0.7981 - val_accuracy: 0.6434 - 87ms/epoch - 4ms/step
Epoch 36/500
20/20 - 0s - loss: 18.9552 - accuracy: 0.4763 - val_loss: 0.8218 - val_accuracy: 0.6434 - 78ms/epoch - 4ms/step
Epoch 37/500
20/20 - 0s - loss: 18.6487 - accuracy: 0.4873 - val_loss: 0.8504 - val_accuracy: 0.6434 - 95ms/epoch - 5ms/step
Epoch 38/500
20/20 - 0s - loss: 17.4093 - accuracy: 0.4810 - val_loss: 0.8864 - val_accuracy: 0.6434 - 93ms/epoch - 5ms/step
Epoch 39/500
20/20 - 0s - loss: 16.8172 - accuracy: 0.5000 - val_loss: 0.9140 - val_accuracy: 0.6434 - 84ms/epoch - 4ms/step
Epoch 40/500
20/20 - 0s - loss: 15.9836 - accuracy: 0.4873 - val_loss: 0.9450 - val_accuracy: 0.6434 - 79ms/epoch - 4ms/step
Epoch 41/500
20/20 - 0s - loss: 17.0433 - accuracy: 0.4747 - val_loss: 0.9730 - val_accuracy: 0.6434 - 79ms/epoch - 4ms/step
Epoch 42/500
20/20 - 0s - loss: 15.2942 - accuracy: 0.5142 - val_loss: 0.9954 - val_accuracy: 0.6434 - 87ms/epoch - 4ms/step
Epoch 43/500
20/20 - 0s - loss: 16.2075 - accuracy: 0.4541 - val_loss: 1.0120 - val_accuracy: 0.6434 - 87ms/epoch - 4ms/step
Epoch 44/500
20/20 - 0s - loss: 14.6470 - accuracy: 0.4842 - val_loss: 1.0371 - val_accuracy: 0.6434 - 89ms/epoch - 4ms/step
Epoch 45/500
20/20 - 0s - loss: 14.1775 - accuracy: 0.4889 - val_loss: 1.0724 - val_accuracy: 0.6434 - 104ms/epoch - 5ms/step
Epoch 46/500
20/20 - 0s - loss: 14.7419 - accuracy: 0.4889 - val_loss: 1.1153 - val_accuracy: 0.6434 - 89ms/epoch - 4ms/step
Epoch 47/500
20/20 - 0s - loss: 13.9817 - accuracy: 0.5063 - val_loss: 1.1560 - val_accuracy: 0.6434 - 87ms/epoch - 4ms/step
Epoch 48/500
20/20 - 0s - loss: 13.4638 - accuracy: 0.4541 - val_loss: 1.1904 - val_accuracy: 0.6434 - 79ms/epoch - 4ms/step
Epoch 49/500
20/20 - 0s - loss: 13.4083 - accuracy: 0.4810 - val_loss: 1.2291 - val_accuracy: 0.6434 - 91ms/epoch - 5ms/step
Epoch 50/500
Restoring model weights from the end of the best epoch: 30.
20/20 - 0s - loss: 13.7986 - accuracy: 0.4794 - val_loss: 1.2780 - val_accuracy: 0.6434 - 91ms/epoch - 5ms/step
Epoch 50: early stopping
<keras.src.callbacks.History at 0x7ab631617af0>
```

```
model_fdiff.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_9 (Dense)             (None, 40)                840

 dropout_7 (Dropout)         (None, 40)                0

 dense_10 (Dense)            (None, 10)                410

 dropout_8 (Dropout)         (None, 10)                0

 dense_11 (Dense)            (None, 50)                550

 dropout_9 (Dropout)         (None, 50)                0

 dense_12 (Dense)            (None, 10)                510

 dropout_10 (Dropout)        (None, 10)                0

 dense_13 (Dense)            (None, 1)                 11

=================================================================
Total params: 2321 (9.07 KB)
Trainable params: 2321 (9.07 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
acc = model_fdiff.evaluate(x=X_test_fdiff, y=y_test_fdiff)
print("Accuracy over test: {:.2f}%".format(acc[1] * 100))

y_prob = model_fdiff.predict(X_test_fdiff)
y_pred = np.round(y_prob)
cm = metrics.confusion_matrix(y_test_fdiff, y_pred)
cm = metrics.ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=["Down", "Up"])
cm.plot(values_format="", cmap="Blues")
print(classification_report(y_test_fdiff, y_pred))
```

```
10/10 [==============================] - 0s 2ms/step - loss: 0.8214 - accuracy: 0.4828
Accuracy over test: 48.28%
10/10 [==============================] - 0s 2ms/step
              precision    recall  f1-score   support

         0.0       0.00      0.00      0.00       150
         1.0       0.48      1.00      0.65       140

    accuracy                           0.48       290
   macro avg       0.24      0.50      0.33       290
weighted avg       0.23      0.48      0.31       290

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined a
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined a
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined a
  _warn_prf(average, modifier, msg_start, len(result))
```



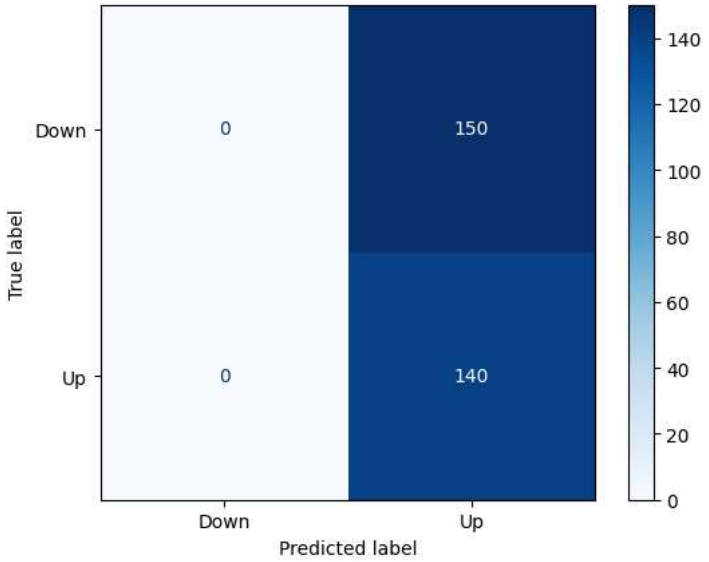
Again, try to train an MLP with default structure.

```
model_fdiff = build_MLP_model()
es = tf.keras.callbacks.EarlyStopping(
    monitor="val_loss", mode="min", verbose=1, patience=30, restore_best_weights=True
)

# fit the model
model_fdiff.fit(
    X_train_fdiff,
    y_train_fdiff,
    validation_split=0.3,
    epochs=500,
    batch_size=32,
    verbose=2,
    callbacks=[es],
)
```

```
Epoch 22/500
20/20 - 0s - loss: 0.7908 - accuracy: 0.5237 - val_loss: 0.7841 - val_accuracy: 0.3529 - 75ms/epoch - 4ms/step
Epoch 23/500
20/20 - 0s - loss: 0.7506 - accuracy: 0.4889 - val_loss: 0.6566 - val_accuracy: 0.6434 - 83ms/epoch - 4ms/step
Epoch 24/500
20/20 - 0s - loss: 0.7951 - accuracy: 0.5380 - val_loss: 0.9404 - val_accuracy: 0.3566 - 92ms/epoch - 5ms/step
Epoch 25/500
20/20 - 0s - loss: 0.7552 - accuracy: 0.5237 - val_loss: 0.8033 - val_accuracy: 0.3566 - 79ms/epoch - 4ms/step
Epoch 26/500
20/20 - 0s - loss: 0.7781 - accuracy: 0.5016 - val_loss: 0.6903 - val_accuracy: 0.6434 - 94ms/epoch - 5ms/step
Epoch 27/500
20/20 - 0s - loss: 0.7645 - accuracy: 0.5016 - val_loss: 0.8637 - val_accuracy: 0.3566 - 138ms/epoch - 7ms/step
Epoch 28/500
20/20 - 0s - loss: 0.7869 - accuracy: 0.4747 - val_loss: 0.6840 - val_accuracy: 0.6434 - 176ms/epoch - 9ms/step
Epoch 29/500
20/20 - 0s - loss: 0.7215 - accuracy: 0.5222 - val_loss: 0.6678 - val_accuracy: 0.6250 - 174ms/epoch - 9ms/step
Epoch 30/500
20/20 - 0s - loss: 0.7350 - accuracy: 0.5269 - val_loss: 0.8514 - val_accuracy: 0.3566 - 138ms/epoch - 7ms/step
Epoch 31/500
20/20 - 0s - loss: 0.7316 - accuracy: 0.5000 - val_loss: 0.6763 - val_accuracy: 0.6434 - 86ms/epoch - 4ms/step
Epoch 32/500
20/20 - 0s - loss: 0.7418 - accuracy: 0.4794 - val_loss: 0.6954 - val_accuracy: 0.6434 - 84ms/epoch - 4ms/step
Epoch 33/500
20/20 - 0s - loss: 0.7386 - accuracy: 0.5032 - val_loss: 0.7277 - val_accuracy: 0.3860 - 196ms/epoch - 10ms/step
Epoch 34/500
20/20 - 0s - loss: 0.6995 - accuracy: 0.5491 - val_loss: 0.7133 - val_accuracy: 0.4449 - 150ms/epoch - 7ms/step
Epoch 35/500
20/20 - 0s - loss: 0.6988 - accuracy: 0.5332 - val_loss: 0.6608 - val_accuracy: 0.6360 - 185ms/epoch - 9ms/step
Epoch 36/500
20/20 - 0s - loss: 0.7010 - accuracy: 0.5253 - val_loss: 0.7335 - val_accuracy: 0.3824 - 228ms/epoch - 11ms/step
Epoch 37/500
20/20 - 0s - loss: 0.7268 - accuracy: 0.4937 - val_loss: 0.6856 - val_accuracy: 0.6434 - 161ms/epoch - 8ms/step
Epoch 38/500
20/20 - 0s - loss: 0.7302 - accuracy: 0.4953 - val_loss: 0.7026 - val_accuracy: 0.4449 - 112ms/epoch - 6ms/step
Epoch 39/500
20/20 - 0s - loss: 0.7095 - accuracy: 0.5206 - val_loss: 0.6638 - val_accuracy: 0.6397 - 145ms/epoch - 7ms/step
Epoch 40/500
20/20 - 0s - loss: 0.7578 - accuracy: 0.5142 - val_loss: 1.0544 - val_accuracy: 0.3566 - 233ms/epoch - 12ms/step
Epoch 41/500
20/20 - 0s - loss: 0.8425 - accuracy: 0.5142 - val_loss: 0.7097 - val_accuracy: 0.4375 - 169ms/epoch - 8ms/step
Epoch 42/500
Restoring model weights from the end of the best epoch: 12.
20/20 - 0s - loss: 0.7858 - accuracy: 0.5063 - val_loss: 0.7532 - val_accuracy: 0.6434 - 154ms/epoch - 8ms/step
Epoch 42: early stopping
<keras.src.callbacks.History at 0x7ab63162a6e0>
```

```
acc = model_fdiff.evaluate(x=X_test_fdiff, y=y_test_fdiff)
print("Accuracy over test: {:.2f}%".format(acc[1] * 100))

y_prob = model_fdiff.predict(X_test_fdiff)
y_pred = np.round(y_prob)
cm = metrics.confusion_matrix(y_test_fdiff, y_pred)
cm = metrics.ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=["Down", "Up"])
cm.plot(values_format="", cmap="Blues")
print(classification_report(y_test_fdiff, y_pred))
```

```
10/10 [==============================] - 0s 9ms/step - loss: 0.7283 - accuracy: 0.4828
Accuracy over test: 48.28%
10/10 [==============================] - 0s 3ms/step
              precision    recall  f1-score   support

         0.0       0.00      0.00      0.00       150
         1.0       0.48      1.00      0.65       140

    accuracy                           0.48       290
   macro avg       0.24      0.50      0.33       290
weighted avg       0.23      0.48      0.31       290

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined a
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined a
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined a
  _warn_prf(average, modifier, msg_start, len(result))
```



It hardly has any predictive power.

We will try again but this time, we substrct "Close" from the past information.

```
df1_feature_fdiff_zero = add_features(df1_label_fdiff, set_zero=True)
X_train_fdiff_zero = df1_feature_fdiff_zero[feature_lst[1:]].astype("float32")

df2_feature_fdiff_zero = add_features(df2_label_fdiff, set_zero=True)
X_test_fdiff_zero = df2_feature_fdiff_zero[feature_lst[1:]].astype("float32")

print(X_train_fdiff_zero.shape, X_test_fdiff_zero.shape)
```

```
(904, 19) (290, 19)
```

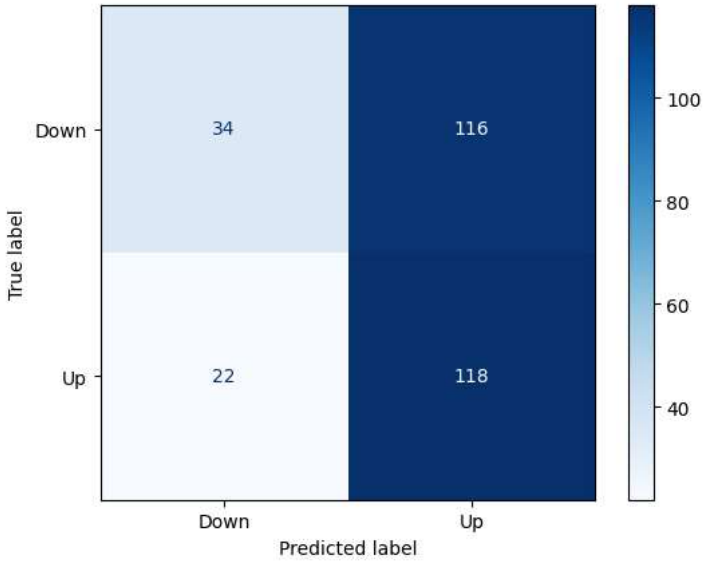
```
model_fdiff_zero = build_MLP_model()
es = tf.keras.callbacks.EarlyStopping(
    monitor="val_loss", mode="min", verbose=1, patience=30, restore_best_weights=True
)

# fit the model
model_fdiff_zero.fit(
    X_train_fdiff_zero,
    y_train_fdiff,
    validation_split=0.3,
    epochs=500,
    batch_size=32,
    verbose=2,
    callbacks=[es],
)
```

```
20/20 - 0s - loss: 0.6750 - accuracy: 0.5870 - val_loss: 0.6907 - val_accuracy: 0.5074 - 192ms/epoch - 10ms/step
Epoch 5/500
20/20 - 0s - loss: 0.6714 - accuracy: 0.5981 - val_loss: 0.6911 - val_accuracy: 0.5074 - 246ms/epoch - 12ms/step
```

```
20/20 - 0s - loss: 0.6546 - accuracy: 0.6108 - val_loss: 0.7076 - val_accuracy: 0.5037 - 216ms/epoch - 11ms/step
Epoch 10/500
20/20 - 0s - loss: 0.6488 - accuracy: 0.6155 - val_loss: 0.7108 - val_accuracy: 0.5221 - 182ms/epoch - 9ms/step
Epoch 11/500
20/20 - 0s - loss: 0.6447 - accuracy: 0.6234 - val_loss: 0.7436 - val_accuracy: 0.4779 - 225ms/epoch - 11ms/step
Epoch 12/500
20/20 - 0s - loss: 0.6400 - accuracy: 0.6282 - val_loss: 0.7225 - val_accuracy: 0.5147 - 247ms/epoch - 12ms/step
Epoch 13/500
20/20 - 0s - loss: 0.6358 - accuracy: 0.6234 - val_loss: 0.7508 - val_accuracy: 0.4669 - 257ms/epoch - 13ms/step
Epoch 14/500
20/20 - 0s - loss: 0.6292 - accuracy: 0.6313 - val_loss: 0.7270 - val_accuracy: 0.5294 - 131ms/epoch - 7ms/step
Epoch 15/500
20/20 - 0s - loss: 0.6218 - accuracy: 0.6361 - val_loss: 0.7818 - val_accuracy: 0.4779 - 182ms/epoch - 9ms/step
Epoch 16/500
20/20 - 0s - loss: 0.6236 - accuracy: 0.6487 - val_loss: 0.7431 - val_accuracy: 0.5331 - 131ms/epoch - 7ms/step
Epoch 17/500
20/20 - 0s - loss: 0.6165 - accuracy: 0.6614 - val_loss: 0.7679 - val_accuracy: 0.5074 - 136ms/epoch - 7ms/step
Epoch 18/500
20/20 - 0s - loss: 0.6084 - accuracy: 0.6551 - val_loss: 0.7462 - val_accuracy: 0.5184 - 157ms/epoch - 8ms/step
Epoch 19/500
20/20 - 0s - loss: 0.6031 - accuracy: 0.6582 - val_loss: 0.7692 - val_accuracy: 0.5331 - 127ms/epoch - 6ms/step
Epoch 20/500
20/20 - 0s - loss: 0.5975 - accuracy: 0.6709 - val_loss: 0.7685 - val_accuracy: 0.5257 - 133ms/epoch - 7ms/step
Epoch 21/500
20/20 - 0s - loss: 0.5906 - accuracy: 0.6709 - val_loss: 0.8784 - val_accuracy: 0.4522 - 129ms/epoch - 6ms/step
Epoch 22/500
20/20 - 0s - loss: 0.5938 - accuracy: 0.6646 - val_loss: 0.7967 - val_accuracy: 0.5221 - 126ms/epoch - 6ms/step
Epoch 23/500
20/20 - 0s - loss: 0.5814 - accuracy: 0.6851 - val_loss: 0.8157 - val_accuracy: 0.5257 - 120ms/epoch - 6ms/step
Epoch 24/500
20/20 - 0s - loss: 0.5742 - accuracy: 0.6978 - val_loss: 0.8125 - val_accuracy: 0.5368 - 122ms/epoch - 6ms/step
Epoch 25/500
20/20 - 0s - loss: 0.5648 - accuracy: 0.7120 - val_loss: 0.8117 - val_accuracy: 0.5331 - 167ms/epoch - 8ms/step
Epoch 26/500
20/20 - 0s - loss: 0.5638 - accuracy: 0.6962 - val_loss: 0.8115 - val_accuracy: 0.5147 - 128ms/epoch - 6ms/step
Epoch 27/500
20/20 - 0s - loss: 0.5536 - accuracy: 0.7152 - val_loss: 0.8329 - val_accuracy: 0.5294 - 176ms/epoch - 9ms/step
Epoch 28/500
20/20 - 0s - loss: 0.5446 - accuracy: 0.7263 - val_loss: 0.8743 - val_accuracy: 0.5110 - 177ms/epoch - 9ms/step
Epoch 29/500
20/20 - 0s - loss: 0.5431 - accuracy: 0.7247 - val_loss: 0.8710 - val_accuracy: 0.5294 - 169ms/epoch - 8ms/step
Epoch 30/500
20/20 - 0s - loss: 0.5432 - accuracy: 0.7104 - val_loss: 0.9647 - val_accuracy: 0.4926 - 126ms/epoch - 6ms/step
Epoch 31/500
Restoring model weights from the end of the best epoch: 1.
20/20 - 0s - loss: 0.5269 - accuracy: 0.7310 - val_loss: 0.9315 - val_accuracy: 0.4853 - 186ms/epoch - 9ms/step
Epoch 31: early stopping
<keras.src.callbacks.History at 0x7ab631a7f6d0>
```

```
acc = model_fdiff_zero.evaluate(x=X_test_fdiff_zero, y=y_test_fdiff)
print("Accuracy over test: {:.2f}%".format(acc[1] * 100))

y_prob = model_fdiff_zero.predict(X_test_fdiff_zero)
y_pred = np.round(y_prob)
cm = metrics.confusion_matrix(y_test_fdiff, y_pred)
cm = metrics.ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=["Down", "Up"])
cm.plot(values_format="", cmap="Blues")
print(classification_report(y_test_fdiff, y_pred))
```

```
10/10 [==============================] - 0s 2ms/step - loss: 0.6927 - accuracy: 0.5241
Accuracy over test: 52.41%
10/10 [==============================] - 0s 2ms/step
              precision    recall  f1-score   support

         0.0       0.61      0.23      0.33       150
         1.0       0.50      0.84      0.63       140

    accuracy                           0.52       290
   macro avg       0.56      0.53      0.48       290
weighted avg       0.56      0.52      0.48       290
```

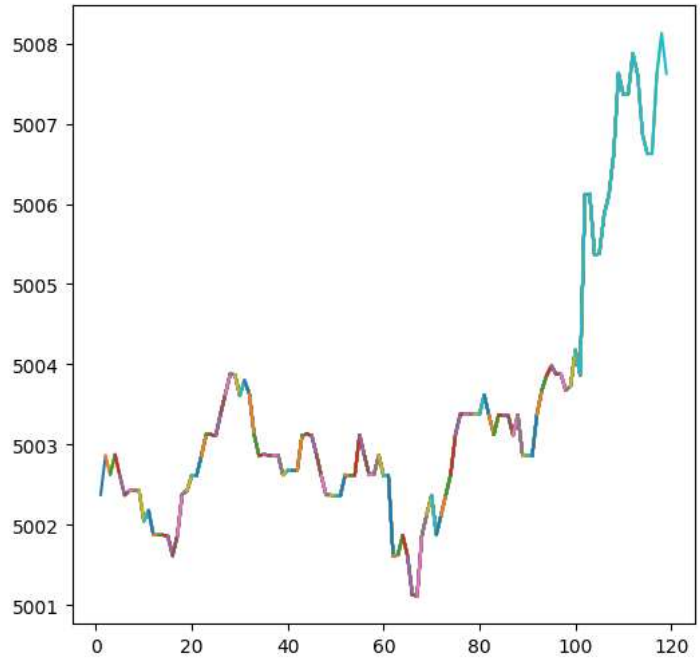


GAF and CNN

Original Time Series

```
window_size = 20

def create_gaf_dataset(df_label, plot=False):
    X = []
    for i in range(window_size, df_label.shape[0]):
            X.append(df_label['Close'][i - window_size : i])
    X = np.array(X)
    print(X.shape)
    y = df_label['Label'][window_size:]
    print(y.shape)

    if plot:
        # Define a "window_size"-days interval
        days = np.linspace(1, X.shape[0], num=X.shape[0])
        # Plot the overlapping time series
        plt.figure(figsize=(6, 6))
        for i in range(100):
                plt.plot(days[i : window_size + i], X[i, :])

    return X, y

X_train_gaf, y_train_gaf = create_gaf_dataset(df1_label, plot=True)
```

```
(939, 20)
(939,)
```

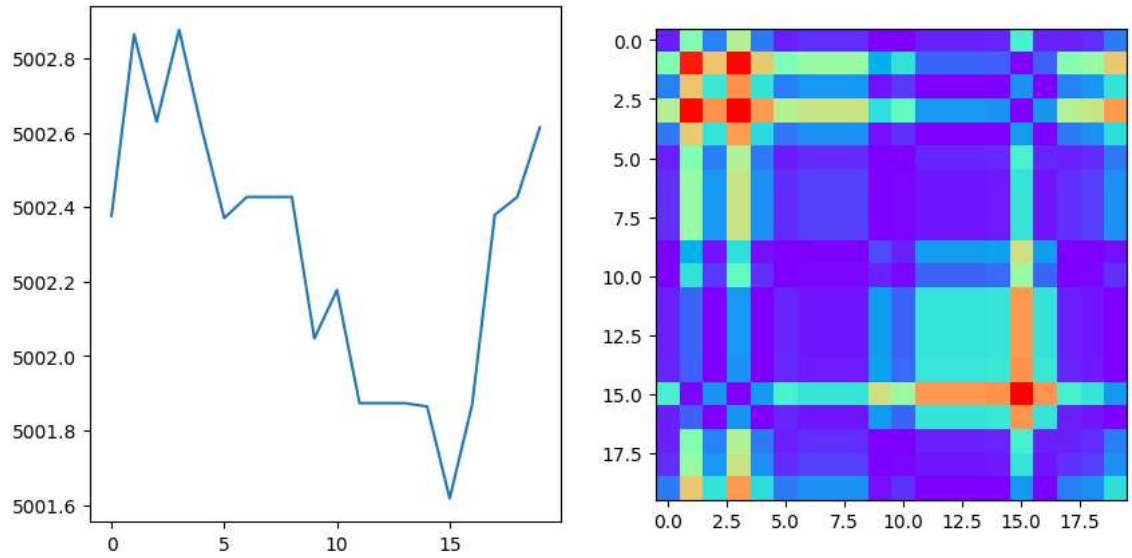



```
transformer = GramianAngularField()
X_train_gaf_new = transformer.transform(X_train_gaf)
print(X_train_gaf_new.shape)
```

```
(939, 20, 20)
```

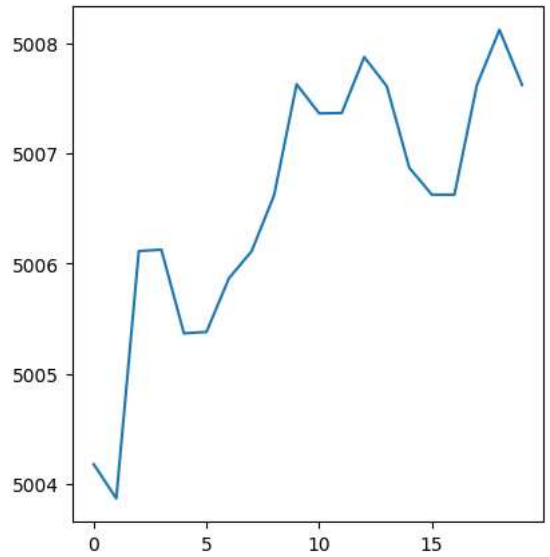
```
# Show the results for the first time series
fig, ax = plt.subplots(1, 2, figsize=(10, 5), gridspec_kw={"width_ratios": [1, 1]})
ax[0].plot(X_train_gaf[0, :])
ax[1].imshow(X_train_gaf_new[0], cmap="rainbow")
```

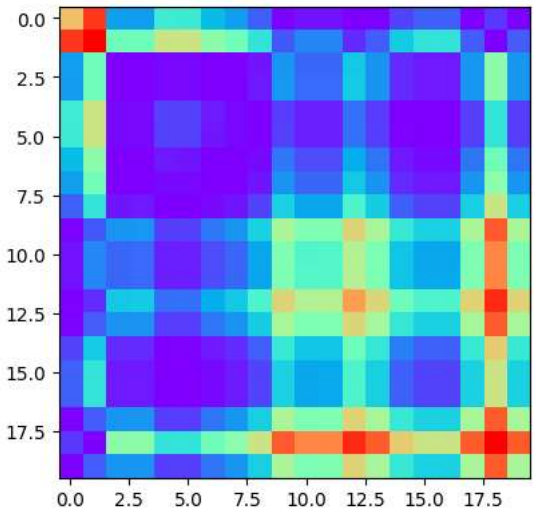
```
<matplotlib.image.AxesImage at 0x7ab630bdeb90>
```




```
# Show the results for the 100th time series
fig, ax = plt.subplots(1, 2, figsize=(10, 5), gridspec_kw={"width_ratios": [1, 1]})
ax[0].plot(X_train_gaf[99, :])
ax[1].imshow(X_train_gaf_new[99], cmap="rainbow")
```

```
<matplotlib.image.AxesImage at 0x7ab62e0b2050>
```






```
X_test_gaf, y_test_gaf = create_gaf_dataset(df2_label)
transformer = GramianAngularField()
X_test_gaf_new = transformer.transform(X_test_gaf)
print(X_test_gaf_new.shape)
```

```
(325, 20)
(325,)
(325, 20, 20)
```

```
def build_CNN_model():
    tf.keras.backend.clear_session()
    tf.random.set_seed(1234)
    model = tf.keras.Sequential()
    # Input layer
    model.add(tf.keras.layers.InputLayer(input_shape=(20, 20, 1)))
    model.add(tf.keras.layers.Conv2D(16, 3, activation="relu"))
    model.add(tf.keras.layers.MaxPooling2D(2))
    model.add(tf.keras.layers.Conv2D(32, 3, activation="relu"))
    model.add(tf.keras.layers.MaxPooling2D(2))
    # model.add(tf.keras.layers.Conv2D(64, 3, activation="relu"))
    # model.add(tf.keras.layers.MaxPooling2D(2))
    model.add(tf.keras.layers.Flatten())
    # model.add(tf.keras.layers.Dense(1024, activation="relu"))
    model.add(tf.keras.layers.Dense(512, activation="relu"))
    model.add(tf.keras.layers.Dropout(0.5, seed=1234))
    model.add(tf.keras.layers.Dense(1, activation="sigmoid"))
    model.compile(loss="binary_crossentropy", optimizer="RMSprop", metrics=["accuracy"])

    return model


model_gaf = build_CNN_model()
model_gaf.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 18, 18, 16)        160

 max_pooling2d (MaxPooling2  (None, 9, 9, 16)          0
 D)

 conv2d_1 (Conv2D)           (None, 7, 7, 32)          4640

 max_pooling2d_1 (MaxPoolin  (None, 3, 3, 32)          0
 g2D)

 flatten (Flatten)           (None, 288)               0

 dense (Dense)               (None, 512)               147968

 dropout (Dropout)           (None, 512)               0

 dense_1 (Dense)             (None, 1)                 513

=================================================================
Total params: 153281 (598.75 KB)
Trainable params: 153281 (598.75 KB)
```

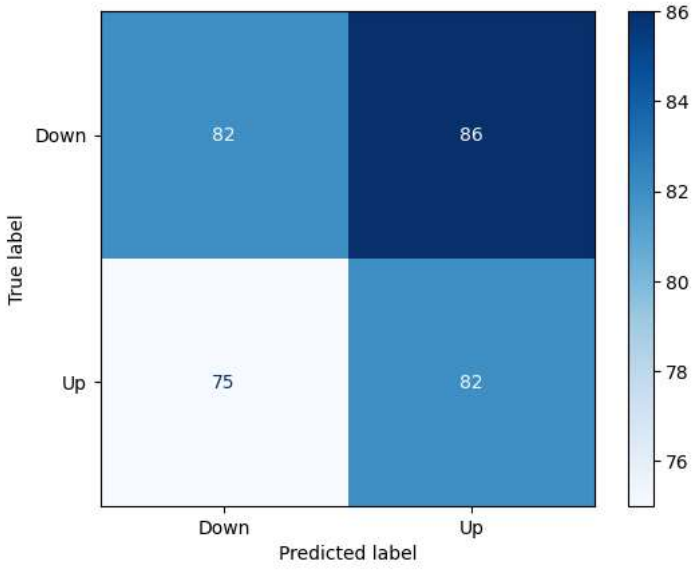
```
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
es = tf.keras.callbacks.EarlyStopping(
    monitor="val_loss", mode="min", verbose=1, patience=30, restore_best_weights=True
)

model_gaf.fit(
    X_train_gaf_new * 255.0,
    y_train_gaf,
    validation_split=0.2,
    epochs=500,
    batch_size=32,
    verbose=2,
    callbacks=[es],
)
```

```
Epoch 1/500
24/24 - 0s - loss: 0.7259 - accuracy: 0.5925 - val_loss: 0.8676 - val_accuracy: 0.5479 - 444ms/epoch - 18ms/step
Epoch 2/500
24/24 - 0s - loss: 0.6919 - accuracy: 0.5779 - val_loss: 0.7999 - val_accuracy: 0.5160 - 315ms/epoch - 13ms/step
Epoch 3/500
24/24 - 0s - loss: 0.6454 - accuracy: 0.6218 - val_loss: 0.8300 - val_accuracy: 0.4468 - 290ms/epoch - 12ms/step
Epoch 4/500
24/24 - 0s - loss: 0.6086 - accuracy: 0.6591 - val_loss: 0.8116 - val_accuracy: 0.4362 - 316ms/epoch - 13ms/step
Epoch 5/500
24/24 - 0s - loss: 0.5997 - accuracy: 0.6551 - val_loss: 0.8367 - val_accuracy: 0.5638 - 271ms/epoch - 11ms/step
Epoch 6/500
24/24 - 0s - loss: 0.5674 - accuracy: 0.6738 - val_loss: 0.9210 - val_accuracy: 0.4894 - 285ms/epoch - 12ms/step
Epoch 7/500
24/24 - 0s - loss: 0.5572 - accuracy: 0.6844 - val_loss: 0.8970 - val_accuracy: 0.4415 - 277ms/epoch - 12ms/step
Epoch 8/500
24/24 - 0s - loss: 0.5390 - accuracy: 0.6858 - val_loss: 1.0146 - val_accuracy: 0.5213 - 274ms/epoch - 11ms/step
Epoch 9/500
24/24 - 0s - loss: 0.5093 - accuracy: 0.7124 - val_loss: 0.9036 - val_accuracy: 0.5904 - 324ms/epoch - 14ms/step
Epoch 10/500
24/24 - 0s - loss: 0.5147 - accuracy: 0.7337 - val_loss: 1.2830 - val_accuracy: 0.5585 - 337ms/epoch - 14ms/step
Epoch 11/500
24/24 - 0s - loss: 0.5363 - accuracy: 0.7244 - val_loss: 1.0595 - val_accuracy: 0.4734 - 315ms/epoch - 13ms/step
Epoch 12/500
24/24 - 0s - loss: 0.4663 - accuracy: 0.7204 - val_loss: 1.2667 - val_accuracy: 0.4894 - 318ms/epoch - 13ms/step
Epoch 13/500
24/24 - 0s - loss: 0.4719 - accuracy: 0.7457 - val_loss: 1.1982 - val_accuracy: 0.4574 - 328ms/epoch - 14ms/step
Epoch 14/500
24/24 - 0s - loss: 0.4483 - accuracy: 0.7630 - val_loss: 1.2448 - val_accuracy: 0.4947 - 452ms/epoch - 19ms/step
Epoch 15/500
24/24 - 0s - loss: 0.4460 - accuracy: 0.7656 - val_loss: 1.2814 - val_accuracy: 0.5053 - 460ms/epoch - 19ms/step
Epoch 16/500
24/24 - 0s - loss: 0.4185 - accuracy: 0.7776 - val_loss: 1.2071 - val_accuracy: 0.5266 - 444ms/epoch - 18ms/step
Epoch 17/500
24/24 - 0s - loss: 0.4192 - accuracy: 0.7696 - val_loss: 1.5675 - val_accuracy: 0.4521 - 466ms/epoch - 19ms/step
Epoch 18/500
24/24 - 0s - loss: 0.4031 - accuracy: 0.7816 - val_loss: 1.4636 - val_accuracy: 0.5266 - 445ms/epoch - 19ms/step
Epoch 19/500
24/24 - 0s - loss: 0.4184 - accuracy: 0.7870 - val_loss: 1.4051 - val_accuracy: 0.4947 - 373ms/epoch - 16ms/step
Epoch 20/500
24/24 - 0s - loss: 0.3797 - accuracy: 0.7883 - val_loss: 1.6697 - val_accuracy: 0.5266 - 321ms/epoch - 13ms/step
Epoch 21/500
24/24 - 0s - loss: 0.3471 - accuracy: 0.8242 - val_loss: 1.5932 - val_accuracy: 0.5000 - 284ms/epoch - 12ms/step
Epoch 22/500
24/24 - 0s - loss: 0.4006 - accuracy: 0.8029 - val_loss: 1.6662 - val_accuracy: 0.4255 - 286ms/epoch - 12ms/step
Epoch 23/500
24/24 - 0s - loss: 0.3241 - accuracy: 0.8336 - val_loss: 1.8787 - val_accuracy: 0.5053 - 276ms/epoch - 11ms/step
Epoch 24/500
24/24 - 0s - loss: 0.3832 - accuracy: 0.8189 - val_loss: 1.8326 - val_accuracy: 0.5053 - 330ms/epoch - 14ms/step
Epoch 25/500
24/24 - 0s - loss: 0.3383 - accuracy: 0.8535 - val_loss: 1.6251 - val_accuracy: 0.5372 - 317ms/epoch - 13ms/step
Epoch 26/500
24/24 - 0s - loss: 0.3455 - accuracy: 0.8429 - val_loss: 1.6396 - val_accuracy: 0.5426 - 276ms/epoch - 11ms/step
Epoch 27/500
24/24 - 0s - loss: 0.3170 - accuracy: 0.8282 - val_loss: 1.7875 - val_accuracy: 0.4840 - 324ms/epoch - 13ms/step
Epoch 28/500
24/24 - 0s - loss: 0.3098 - accuracy: 0.8575 - val_loss: 2.1311 - val_accuracy: 0.5745 - 316ms/epoch - 13ms/step
Epoch 29/500
24/24 - 0s - loss: 0.2603 - accuracy: 0.8788 - val_loss: 1.6920 - val_accuracy: 0.5585 - 319ms/epoch - 13ms/step
```

```
acc = model_gaf.evaluate(x=X_test_gaf_new * 255.0, y=y_test_gaf)
print("Accuracy over test: {:.2f}%".format(acc[1] * 100))

y_prob = model_gaf.predict(X_test_gaf_new * 255.0)
y_pred = np.round(y_prob)
cm = metrics.confusion_matrix(y_test_gaf, y_pred)
cm = metrics.ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=["Down", "Up"])
cm.plot(values_format="", cmap="Blues")
print(classification_report(y_test_diff, y_pred))
```

```
11/11 [==============================] - 0s 4ms/step - loss: 1.0633 - accuracy: 0.5046
Accuracy over test: 50.46%
11/11 [==============================] - 0s 4ms/step
              precision    recall  f1-score   support

         0.0       0.52      0.49      0.50       168
         1.0       0.49      0.52      0.50       157

    accuracy                           0.50       325
   macro avg       0.51      0.51      0.50       325
weighted avg       0.51      0.50      0.50       325
```



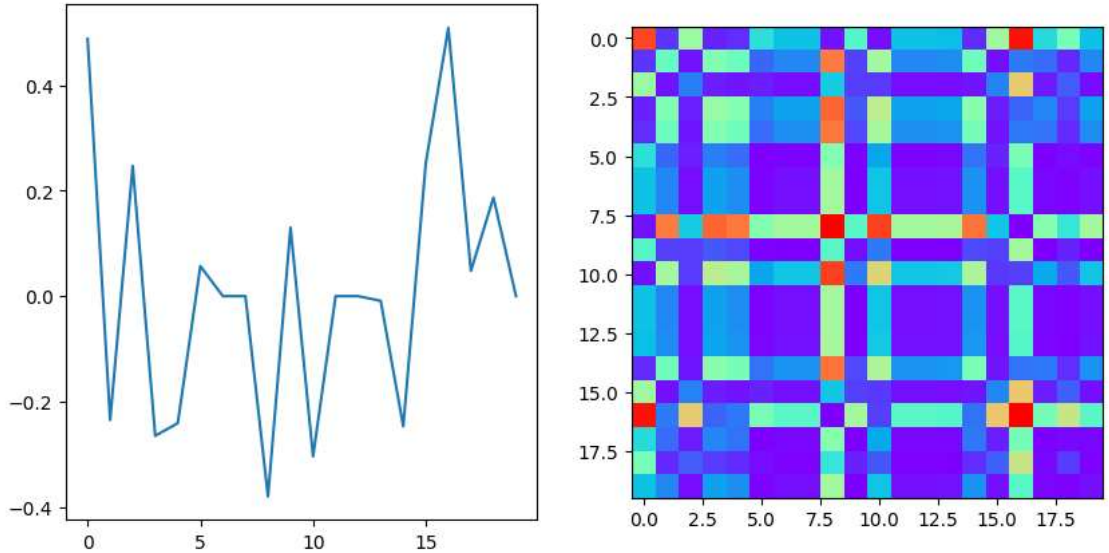
Stationary Time Series

```
X_train_gaf_diff, y_train_gaf_diff = create_gaf_dataset(df1_label_diff)
transformer = GramianAngularField()
X_train_gaf_diff_new = transformer.transform(X_train_gaf_diff)
print(X_train_gaf_diff_new.shape)
```

```
(938, 20)
(938,)
(938, 20, 20)
```

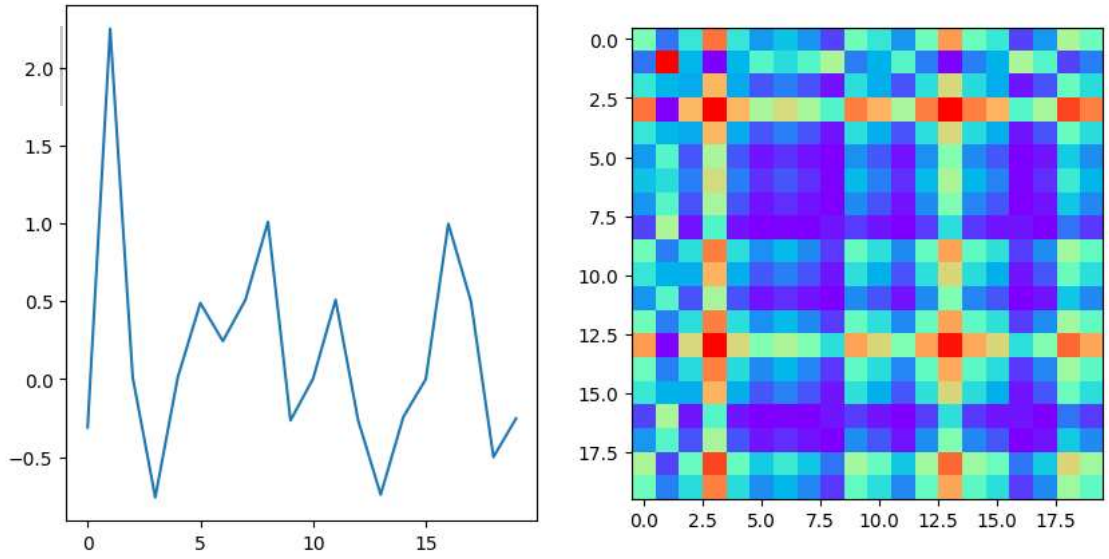
```
# Show the results for the first time series
fig, ax = plt.subplots(1, 2, figsize=(10, 5), gridspec_kw={"width_ratios": [1, 1]})
ax[0].plot(X_train_gaf_diff[0, :])
ax[1].imshow(X_train_gaf_diff_new[0], cmap="rainbow")
```

```
<matplotlib.image.AxesImage at 0x7ab62f3086a0>
```



```
# Show the results for the first time series
fig, ax = plt.subplots(1, 2, figsize=(10, 5), gridspec_kw={"width_ratios": [1, 1]})
ax[0].plot(X_train_gaf_diff[99, :])
ax[1].imshow(X_train_gaf_diff_new[99], cmap="rainbow")
```

```
<matplotlib.image.AxesImage at 0x7ab62f1d4850>
```




```
X_test_gaf_diff, y_test_gaf_diff = create_gaf_dataset(df2_label_diff)
transformer = GramianAngularField()
X_test_gaf_diff_new = transformer.transform(X_test_gaf_diff)
print(X_test_gaf_diff_new.shape)
```

```
(324, 20)
(324,)
(324, 20, 20)
```

```
model_gaf_diff = build_CNN_model()
es = tf.keras.callbacks.EarlyStopping(
    monitor="val_loss", mode="min", verbose=1, patience=30, restore_best_weights=True
)
model_gaf_diff.fit(
    X_train_gaf_diff_new * 255.0,
    y_train_gaf_diff,
    validation_split=0.2,
    epochs=500,
    batch_size=32,
    verbose=2,
    callbacks=[es],
)
```

```
24/24 - 0s - loss: 0.6834 - accuracy: 0.5853 - val_loss: 0.7160 - val_accuracy: 0.6170 - 321ms/epoch - 13ms/step
Epoch 7/500
24/24 - 0s - loss: 0.6347 - accuracy: 0.6160 - val_loss: 0.7084 - val_accuracy: 0.5798 - 332ms/epoch - 14ms/step
Epoch 8/500
24/24 - 0s - loss: 0.6034 - accuracy: 0.6227 - val_loss: 0.6998 - val_accuracy: 0.4947 - 315ms/epoch - 13ms/step
Epoch 9/500
24/24 - 0s - loss: 0.6508 - accuracy: 0.6427 - val_loss: 0.9638 - val_accuracy: 0.4309 - 312ms/epoch - 13ms/step
Epoch 10/500
24/24 - 0s - loss: 0.6428 - accuracy: 0.6653 - val_loss: 0.7049 - val_accuracy: 0.4734 - 329ms/epoch - 14ms/step
Epoch 11/500
24/24 - 0s - loss: 0.5715 - accuracy: 0.6653 - val_loss: 0.8679 - val_accuracy: 0.5479 - 321ms/epoch - 13ms/step
Epoch 12/500
24/24 - 0s - loss: 0.5856 - accuracy: 0.6813 - val_loss: 0.9044 - val_accuracy: 0.5372 - 290ms/epoch - 12ms/step
Epoch 13/500
24/24 - 0s - loss: 0.5720 - accuracy: 0.6933 - val_loss: 0.7143 - val_accuracy: 0.5266 - 287ms/epoch - 12ms/step
Epoch 14/500
24/24 - 0s - loss: 0.5581 - accuracy: 0.7013 - val_loss: 0.7562 - val_accuracy: 0.5213 - 308ms/epoch - 13ms/step
Epoch 15/500
24/24 - 0s - loss: 0.4904 - accuracy: 0.7293 - val_loss: 0.8202 - val_accuracy: 0.4947 - 311ms/epoch - 13ms/step
Epoch 16/500
24/24 - 0s - loss: 0.5488 - accuracy: 0.7013 - val_loss: 0.8131 - val_accuracy: 0.5053 - 308ms/epoch - 13ms/step
```

```
24/24 - 0s - loss: 0.4421 - accuracy: 0.7840 - val_loss: 1.0264 - val_accuracy: 0.5372 - 311ms/epoch - 13ms/step
Epoch 23/500
24/24 - 0s - loss: 0.4386 - accuracy: 0.7693 - val_loss: 1.0146 - val_accuracy: 0.5426 - 396ms/epoch - 17ms/step
Epoch 24/500
24/24 - 0s - loss: 0.4126 - accuracy: 0.7693 - val_loss: 1.2746 - val_accuracy: 0.5160 - 453ms/epoch - 19ms/step
Epoch 25/500
24/24 - 0s - loss: 0.4904 - accuracy: 0.7720 - val_loss: 1.2114 - val_accuracy: 0.5426 - 474ms/epoch - 20ms/step
Epoch 26/500
24/24 - 0s - loss: 0.5833 - accuracy: 0.7680 - val_loss: 1.3422 - val_accuracy: 0.5106 - 483ms/epoch - 20ms/step
Epoch 27/500
24/24 - 0s - loss: 0.4053 - accuracy: 0.7867 - val_loss: 1.2672 - val_accuracy: 0.5000 - 464ms/epoch - 19ms/step
Epoch 28/500
24/24 - 0s - loss: 0.3838 - accuracy: 0.8067 - val_loss: 1.2720 - val_accuracy: 0.4894 - 426ms/epoch - 18ms/step
Epoch 29/500
24/24 - 0s - loss: 0.4018 - accuracy: 0.7853 - val_loss: 1.1290 - val_accuracy: 0.4894 - 273ms/epoch - 11ms/step
Epoch 30/500
24/24 - 0s - loss: 0.3164 - accuracy: 0.8293 - val_loss: 1.7429 - val_accuracy: 0.4415 - 279ms/epoch - 12ms/step
Epoch 31/500
24/24 - 0s - loss: 0.4029 - accuracy: 0.7933 - val_loss: 1.0968 - val_accuracy: 0.5213 - 297ms/epoch - 12ms/step
Epoch 32/500
24/24 - 0s - loss: 0.3401 - accuracy: 0.8120 - val_loss: 1.3342 - val_accuracy: 0.5798 - 275ms/epoch - 11ms/step
Epoch 33/500
Restoring model weights from the end of the best epoch: 3.
24/24 - 0s - loss: 0.3923 - accuracy: 0.8107 - val_loss: 1.2002 - val_accuracy: 0.4894 - 288ms/epoch - 12ms/step
Epoch 33: early stopping
<keras.src.callbacks.History at 0x7ab62dc0f100>
```

```
acc = model_gaf_diff.evaluate(x=X_test_gaf_diff_new * 255.0, y=y_test_gaf_diff)
print("Accuracy over test: {:.2f}%".format(acc[1] * 100))

y_prob = model_gaf_diff.predict(X_test_gaf_diff_new * 255.0)
y_pred = np.round(y_prob)
cm = metrics.confusion_matrix(y_test_gaf_diff, y_pred)
cm = metrics.ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=["Down", "Up"])
cm.plot(values_format="", cmap="Blues")
print(classification_report(y_test_gaf_diff, y_pred))
```

```
11/11 [==============================] - 0s 8ms/step - loss: 0.7333 - accuracy: 0.4599
Accuracy over test: 45.99%
11/11 [==============================] - 0s 6ms/step
              precision    recall  f1-score   support

         0.0       0.45      0.22      0.29       167
         1.0       0.46      0.72      0.56       157

    accuracy                           0.46       324
   macro avg       0.46      0.47      0.43       324
weighted avg       0.46      0.46      0.42       324
```