GROUP WORK PROJECT # 1:
Group Number: 7636

MScFE 642: Deep Learning for Finance

FULL LEGAL NAME	LOCATION (COUNTRY)	EMAIL ADDRESS	MARK X FOR ANY NON-CONTRIBUTING MEMBER
CHIEDOZIE AUGUSTINE IKE-OFFIAH	NORTH CYPRUS (TURKEY)	chiexaustine2005@gmail.com	
ARIFIN LIAUW	INDONESIA	james.arifin@gmail.com	
SHIVANSH KUMAR	INDIA	shivansh.business23@gmail.com	

Statement of integrity: By typing the names of all group members in the text boxes below, you confirm that the assignment submitted is original work produced by the group (excluding any non-contributing members identified with an "X" above).

Team member 1	**CHIEDOZIE AUGUSTINE IKE-OFFIAH**
Team member 2	**ARIFIN LIAUW**
Team member 3	**SHIVANSH KUMAR**

Use the box below to explain any attempts to reach out to a non-contributing member. Type (N/A) if all members contributed.
Note: You may be required to provide proof of your outreach to non-contributing members upon request.

N/A

SCENARIO

For this GWP, the scenario is as follows:

We spearhead the optimization engineering team for a statistical arbitrage division. As our first responsibility, we are tasked with finding optimal parameter configurations in models that trade the firm's capital.

STEP 1.

A.

In this step, we will observe the time series of the PT Bank Rakyat Indonesia (Persero) Tbk, denoted by the ticker symbol of BBRI.JK to first get an overview of its statistical properties, namely its mean, standard deviation, skewness, and its kurtosis. The time series is that of the closing price for the period from 1st January 2017 to 30th October 2024. The series is as presented in Figure 1a.

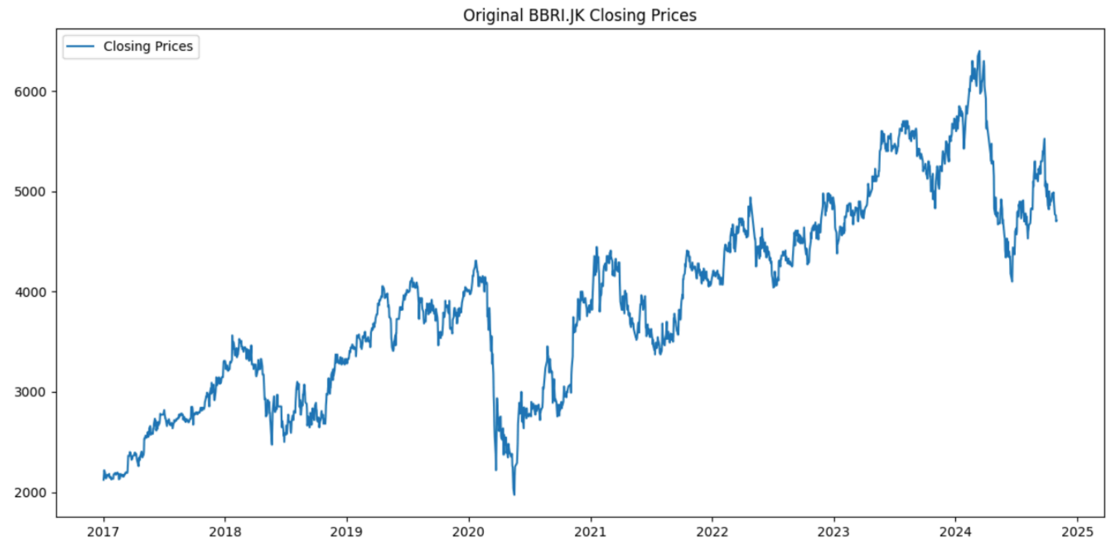


Figure 1a: Time series plot for the closing prices of BBRI.JK

Table 1 briefly summarizes the basic statistical properties of the time series, including its mean, skewness, and kurtosis.

Table 1: Relevant Summary Statistics

Summary Statistics	**Value**
Count	1944
Mean	3875.82
Standard Deviation	968.38

Figure 2a depicts the histogram of the prices to give an idea of how the closing prices vary. As we can see, the shape of the distribution is not exactly normal. However, it still looks like what we can expect from security prices.

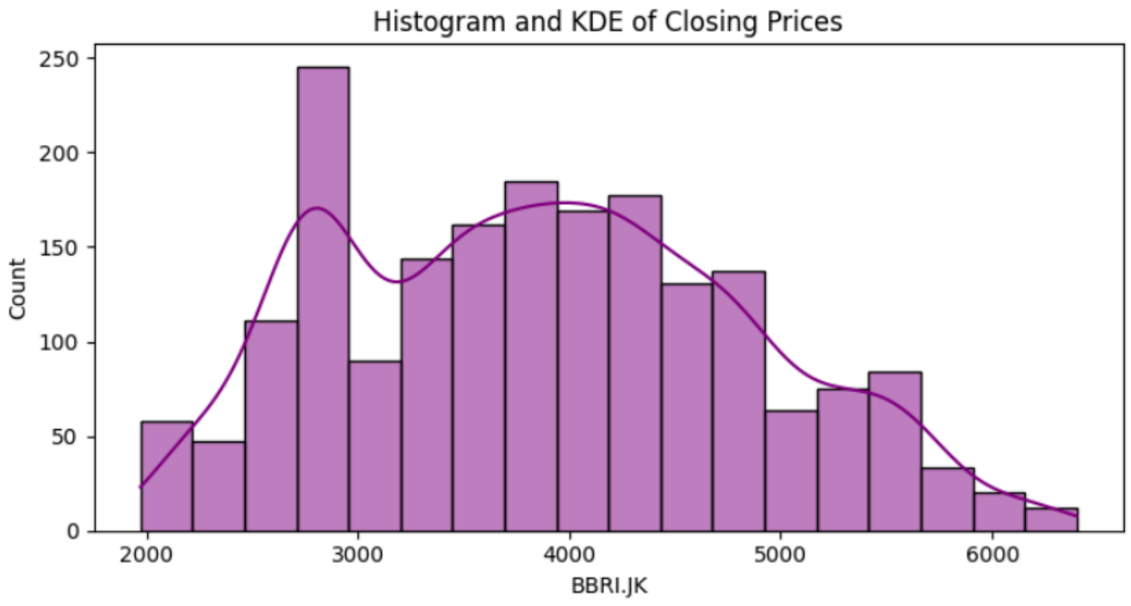


Figure 2a: Histogram of the closing prices of BBRI.JK

Still trying to understand the distribution of the series, we can now observe also the QQ plot and autocorrelation of the prices. This is represented together as in Figure 3a. As we can observe from the QQ plot, we see indeed that the series is close to normality as it almost fits on the diagonal of the plot. The outlier region in the QQ plot may have been due to the COVID period, when prices of securities sort of behaved erratically. Observing from the ACF plot, we can see that the values in the time series plot are highly correlated (using a lag of 30). To confirm this, we can use the ADF test to see the relevant statistics.

The Augmented Dickey-Fuller Test Results are as follows:

{'ADF Statistic': -1.9264705504965844, 'p-value': 0.31971280374902095, 'Critical Values': {'1%': -3.4337252441664483, '5%': -2.8630309758314314, '10%': -2.56756373605059}}

The p-value is greater than 0.05, suggesting that the series is non-stationary. This confirms our initial suspicion.

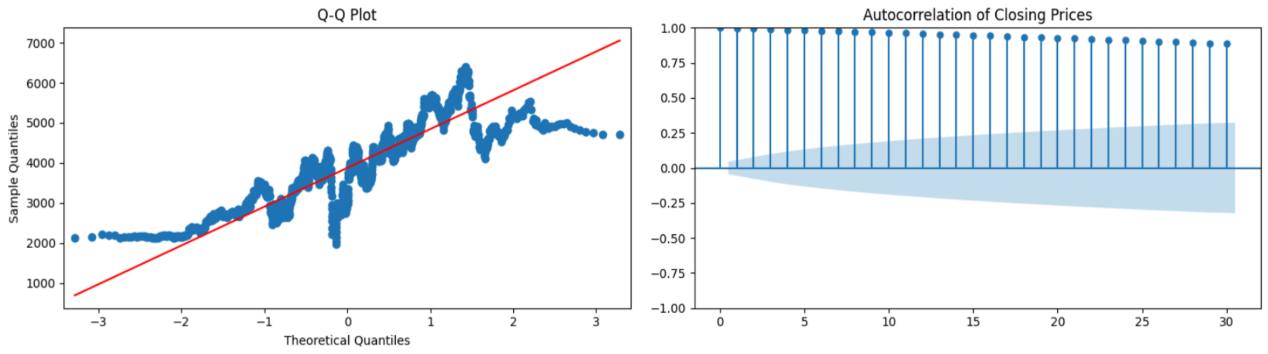


Figure 3a: QQ plot and the autocorrelation of the closing prices for the BBRI.JK stock

B.

In this step, we will perform the same analysis as in Step 1.A. for the transformed version of the time series. We will transform the time series into a log time series of the return. Upon completing the transformation, we perform statistical analysis to capture the mean, standard deviation, skewness, and kurtosis of the transformed time series.

Following is the result of the transform time series for BBRI.JK in figure 1.b. that we used in this work:

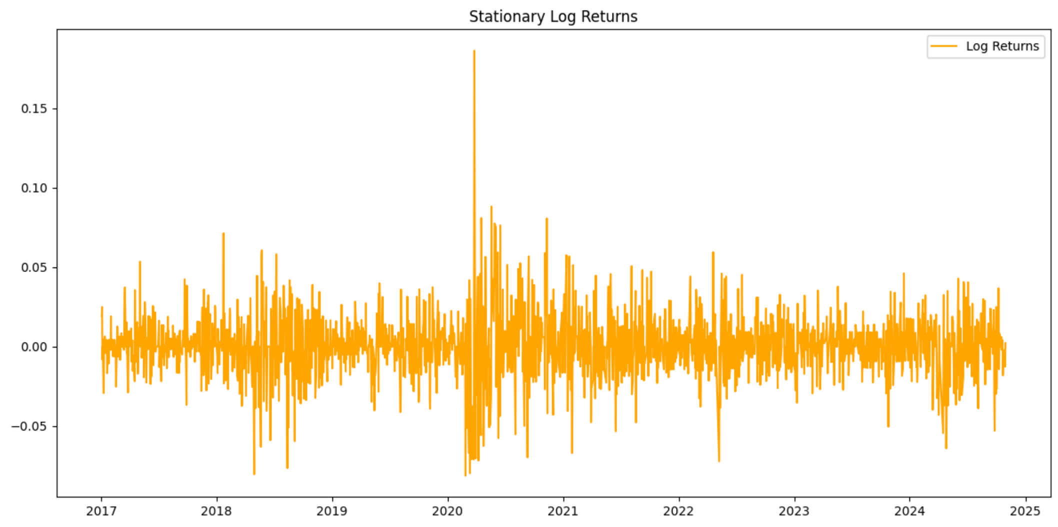


Figure 1.b: Log Returns plot

Based on this transformed time series, we got the following statistics:

Mean	0.00041019
Standard deviation	0.019475
Skewness	0.38336876
Kurtosis	6.66562738

Table 1. b: Summary Statistics

Based on the above statistic summary, the average return for BBRI.JK during the period that we evaluate is around 0.41%, which is very close to zero, indicating that the return is oscillating around that mean with a standard deviation of 1.95%, and the return is slightly positively skewed with high kurtosis (leptokurtic), indicating there is a sharp peak with fat tails.

Then we constructed the histogram and Q-Q plot to visualize the distribution of the stationary time series which resulted in this graph in figure 2.b.

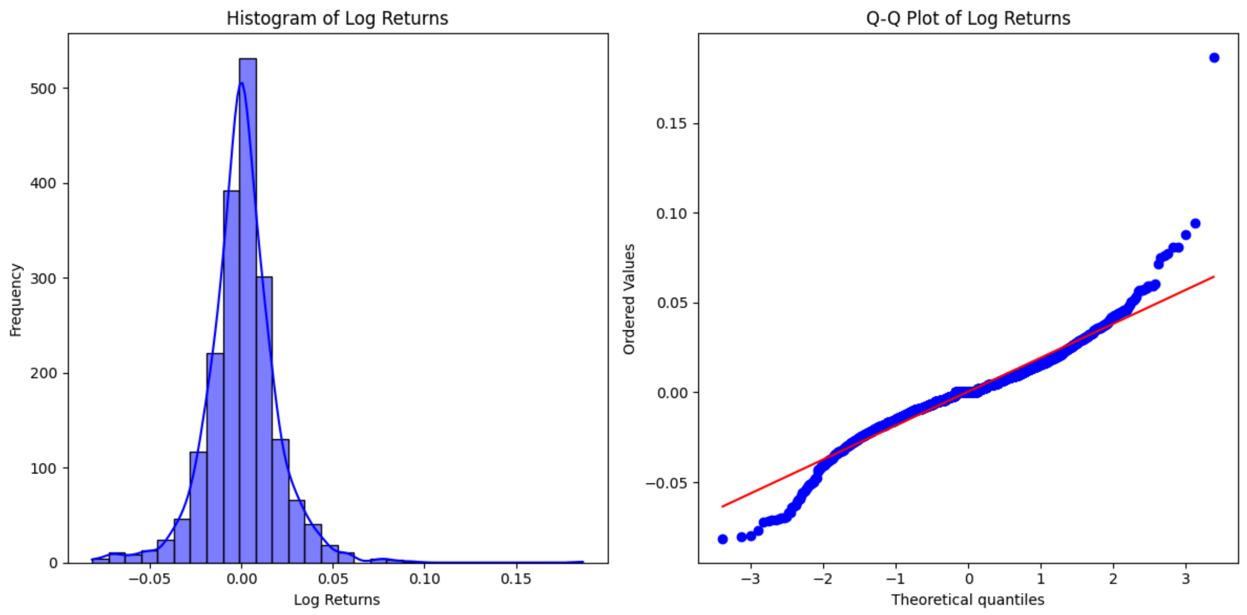
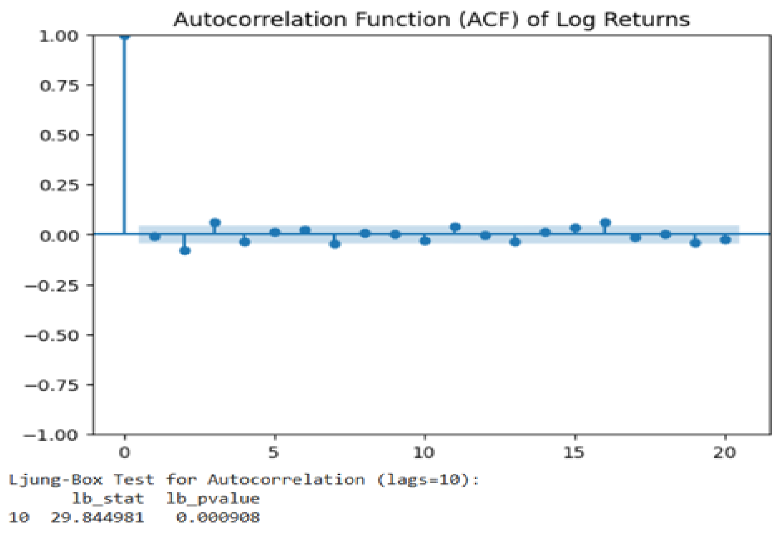


Figure 2.b: Histogram and Q-Q plot of log returns

The above histogram plotting and Q-Q plot visualize the statistical data above, confirming the positive skewness distribution and there is an outlier. The outlier based on the log returns plots happened in 2020 in the COVID era.

To check the stationarity of the transformed time series, we are using an ACF plot in figure 3.b and a Ljung Box test.



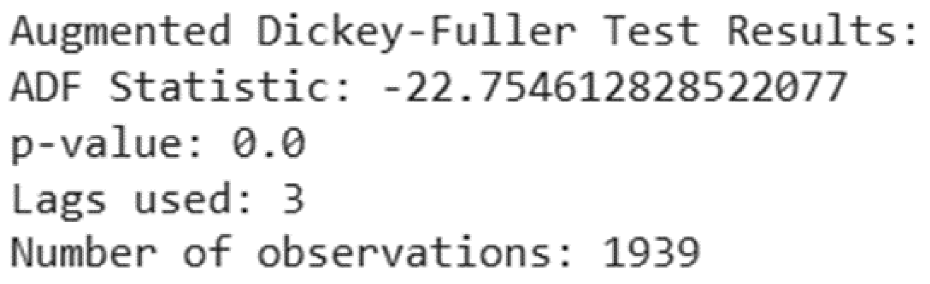
```
Ljung-Box Test for Autocorrelation (lags=10):
      lb_stat  lb_pvalue
10  29.844981   0.000908
```

Figure 3. b: ACF plot of log returns

The result of the Ljung-Box Test Autocorrelation indicates that the time series are strongly autocorrelated in the time series up to 10 lags. The very low p-value enables us to reject the null hypothesis.

In practice, we could use past returns to predict future returns in a short period.

The results of the Augmented Dickey-Fuller (ADF test) are in Figure 4. b:



```
Augmented Dickey-Fuller Test Results:
ADF Statistic: -22.75461282852207
p-value: 0.0
Lags used: 3
Number of observations: 1939
```

Figure 4. b: ADF test output of log retuns

Based on the ADF test result, the log return time series shows that the dataset is stationary.

C.

In this step, we will try something different, i.e., fractional differencing. We are going to obtain a new version of the time series that is stationary but retains relevant information about the data, but first, let's understand what fractional difference is. So it is a technique that is a flexible form of differencing where we transform a time series to make it stationary while retaining more of the long-term memory or dependency structure compared to regular differencing. It is mostly used in financial time series where some degree of memory and persistence in the data is often present.

Traditional differencing:

In time series analysis, differencing is a technique used to make a series stationary. Stationarity is a property where the mean, variance, and autocovariance of the series do not change over time. For example, if a stock closing price series has a trend, differencing can help remove this trend and stabilize the series (Maitra and Sarit).

The traditional differencing is as follows:

$$Y_t = X_t = X_{t-1} \qquad (1)$$

Where:

X_t is the original time series

Y_t is the differenced series

This is an example of first-order differencing.

Fractional differencing:

Fractional differencing involves taking a non-integer order of differencing to retain more of the memory in the data while still achieving stationarity. It is widely used for financial and economic time series because of its long-range dependencies. Instead of fully differencing the series and removing a large amount of this memory, fractional differencing helps reduce the dependency while preserving some of the long-term correlation structure.

The fractional difference relies on the binomial expansion to compute non-integer differences. If we let d represent the fractional differencing order, then the fractional difference of a time series X_t can be written as (Maitra and Sarit):

$$Y_t = (1 - B)^d X_t \qquad (2)$$

Where:

B is the backshift operator, i.e. $BX_t = X_{t-1}$

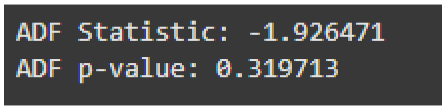
d is the fractional value.

$(1 - B)^d$ from eq. (2) can also be written in binomial series : $Y_t = \sum_{k=0}^{\infty} \binom{d}{K} (-1)^k B^k X_t$

This expansion lets us apply a fractional power d to the differencing operator (1−B), creating fractional differences.

Now let's return to our analysis. For that, we have used the closing price data of BBRI.JK from 2017-2024 using Yahoo Finance API.

We will first start with the ADF test to check the stationarity of the closing price time series



```
ADF Statistic: -1.926471
ADF p-value: 0.319713
```

Figure 1. c: Adf Test for closing prices

We get the ADF stats of -1.926471 and ADF p-value: 0.319713 so we cannot reject the null hypothesis of non-stationarity. Now what we will do is see traditional differences and how it is working. We will do two things. We will first calculate the first-order difference of the losing time series, and we will also calculate the log first-order difference to see how traditional differencing behaves.

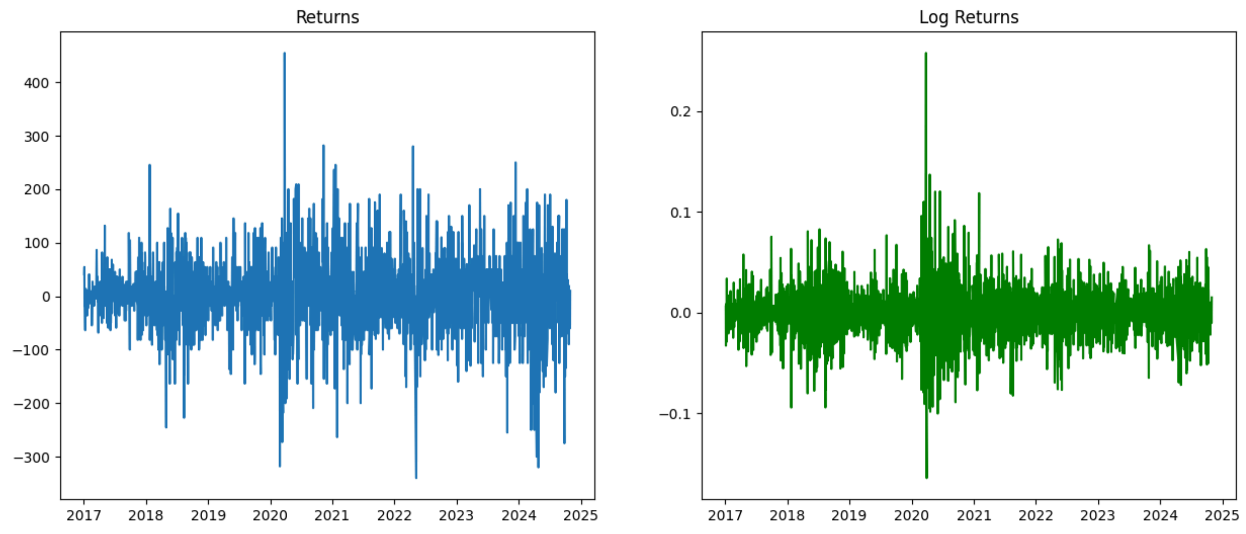
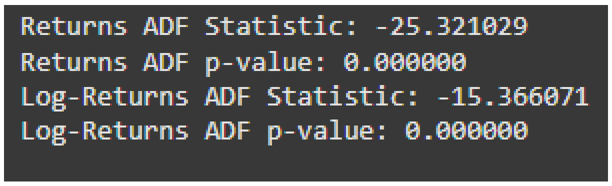


Figure 2. c: plot for return and log returns

Here in the above image of Figure 2. c , we can plot the first-order difference as well as log the first-order difference Now we will see the ADF test result of them.



```
Returns ADF Statistic: -25.321029
Returns ADF p-value: 0.000000
Log-Returns ADF Statistic: -15.366071
Log-Returns ADF p-value: 0.000000
```

Figure 3.c: Adf test for normal and log returns

Here in the above image we can interpret that can reject the null hypothesis of non-stationarity, i.e., the time series are stationary, but we have already seen how traditional differencing works and what the results are. For this section, we are more interested in fractional difference (Elvis-Espinal).

As we have already discussed fractional differencing and how it works, so using a fractional value of d = 0.3 we have obtained our time series. We have used d as a small value, i.e. 0 < d < 0.5 as it removes the non-stationarity and also keeps working memory (ElvisEspinal).

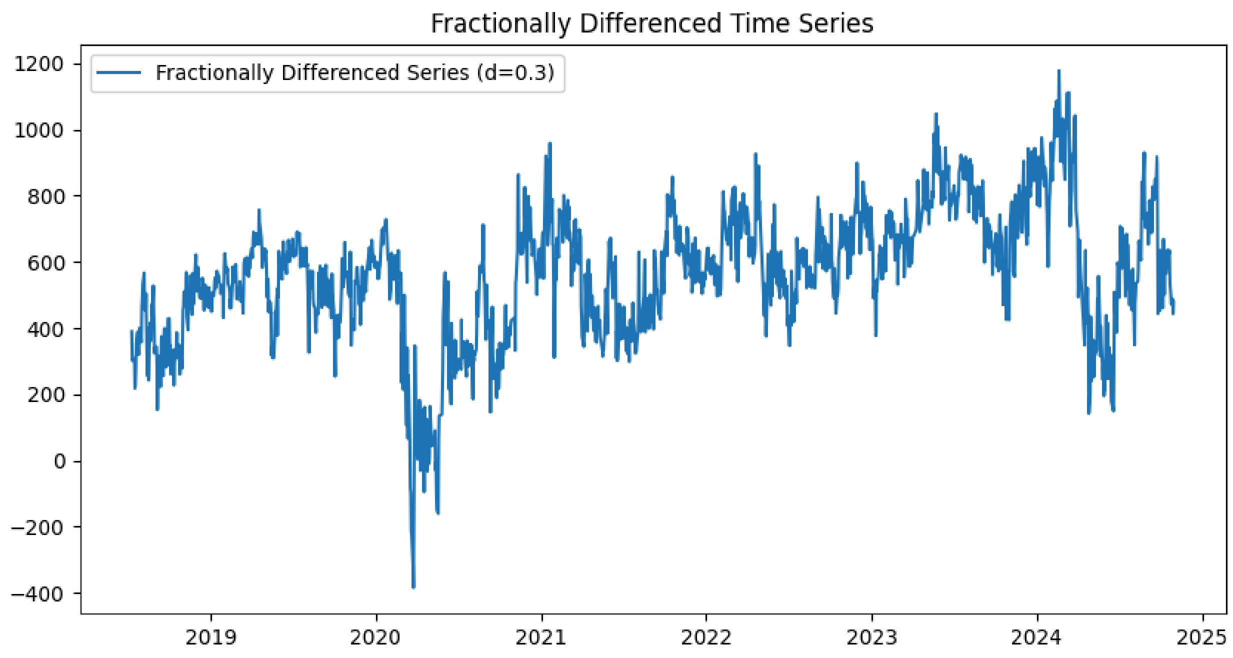


Figure 4.c: Fractionally Differenced Time series plot

In Figure 4.c we can see our fractionally differenced time series using d = 0.3 which is stationary and also preserving long term memory.

But we will now do some statistics tests to see how our transformed data is behaving; we will do some basic descriptive statistics, some distribution analysis, the ACF and Ljung test and at last the ADF test.

Based on this transformed time series, we got the following statistics:

Table 1.c: Summary statistics

Mean	563.3403688759303
Standard deviation	195.95526890439922
Skewness	-0.4937077953830805
Kurtosis	1.263174272580759

Mean value 563.34 suggests the stock have upward trend and the high value of the mean suggests a strong average growth rate, the std dev 195.96 indicates a significant volatility, meaning that the stock price exhibits large fluctuations over time. Skewness of -0.49 indicates that the left tail of the distribution is longer or fatter than the right tail. This implies more extreme negative returns than positive returns. At last the kurtosis of 1.26 indicates the distribution is platykurtic, meaning it has flatter tails and fewer extreme events compared to a normal distribution.

Next, we will see the Histogram and QQ plot of fractional difference:

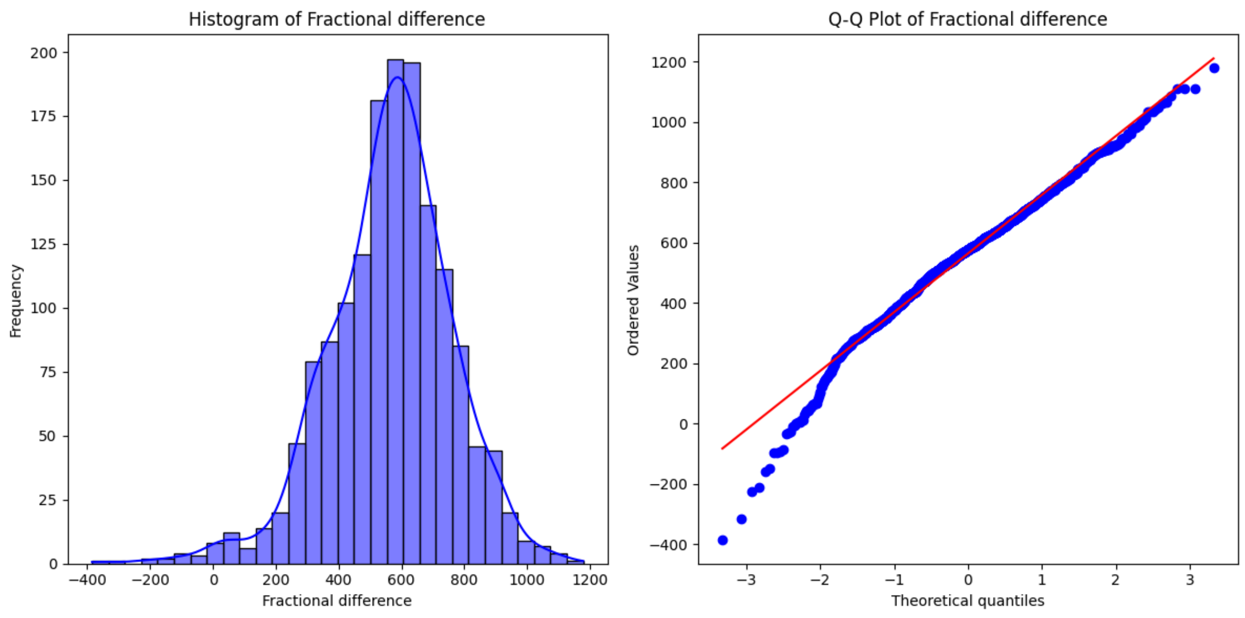
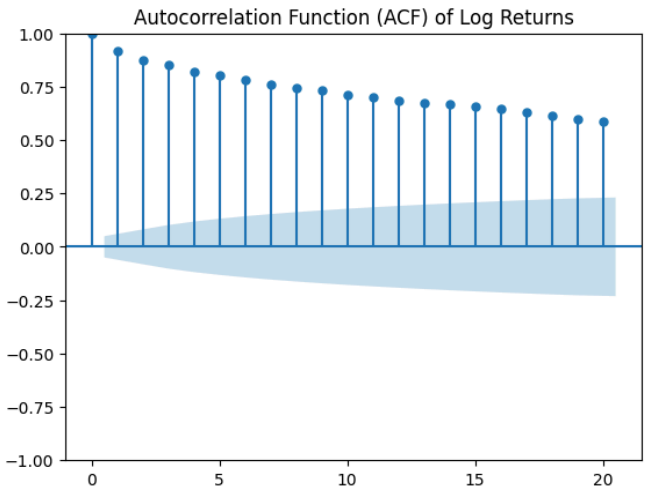
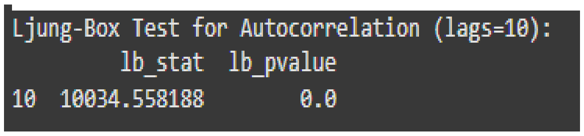


Figure 5.c: Histogram and Q-Q plot of fractionally difference time series

We can see the skewness in the histogram and Q-Q plot in Figure 5.c also how we have more extreme negative returns than positive returns.

We will now do the ACF and Ljung box test and see how our transformed data is working.



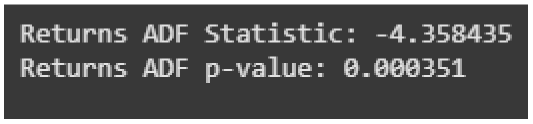


```
Ljung-Box Test for Autocorrelation (lags=10):
         lb_stat  lb_pvalue
10  10034.558188        0.0
```

Figure 6.c : ACF ad Ljung test output

In Figure 6.c we can see how our data is performing we can see in the first images how there is slow decay in dependent which signifies that the time series has long memory. And our ljung-boc test also suggests that there is strong autocorrelation in our time series. Our p-value is also zero which rejects the null hypothesis.

Finally we Check ADF test of our fractional time series and the output is:



```
Returns ADF Statistic: -4.358435
Returns ADF p-value: 0.000351
```

Figure 7.c : ADF test

Interpreting the above output we can conclude that our fractional time series is stationary.

So as the goal of fractional differencing is to provide a stationary time series with Long-Term Dependence, and through our analysis we can see that we have achieved that, now we will use this fractionally differenced data in some DL techniques and see how our prediction task goes.

STEP 2.

A.

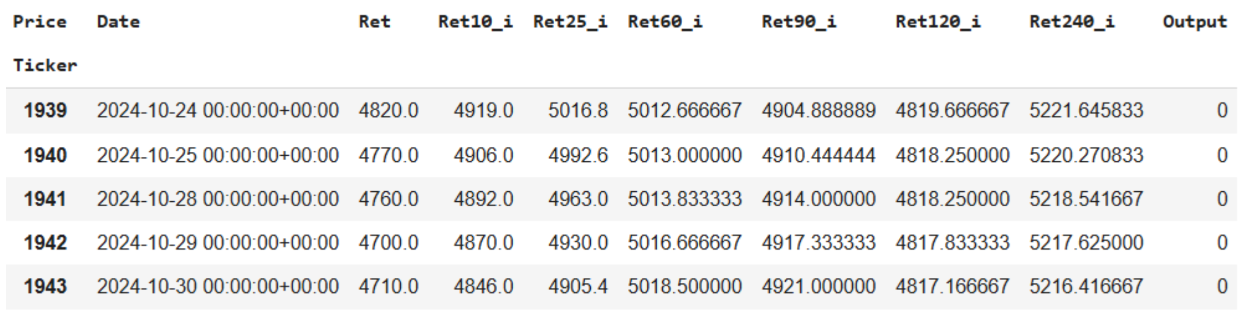
In this section of the task, we aim to build a multilayer perceptron model (MLP) that will take as inputs the past information of the time series. We aim to understand the performance of the model when our data is yet untransformed. Table 2a summarizes the relevant parameters we used to build the aforementioned model. In our model, we consider as inputs into the network the mean of the past prices for the 10, 20, 25, 60, 120, and 240 days, respectively. Hence, our model has 6 inputs. We aim to implement a classification task in this first instance. Hence, we need to create an output that outputs 1 or 0 (binary classification). We will focus on the ***t***+10 trading days for the BBRI.JK stock. In this case, we will perform a classification task without the MLP model so that we simply aim to predict whether, on a given time, from ***t*** to ***t***+10 days, the prices are greater or less than the average price.

Table 2a. Summary of the Model Parameters

Parameter Models	**Values, parameters**
Input layer activation function	RELU
First hidden layer	25
Second hidden layer	15
Third hidden layer	10
Dropout	0.2
Learning rate	0.00001
Optimizer	Adam Optimizer
Output layer activation function	Sigmoid

Table 3a gives a snippet of what the data frame of the last rows in our task looks like. Here the 'Ret' column actually refers to the closing prices and not the returns.

Table 3a: Summary of the data frame for the initial task.



Price	Date	Ret	Ret10_i	Ret25_i	Ret60_i	Ret90_i	Ret120_i	Ret240_i	Output
Ticker									
1939	2024-10-24 00:00:00+00:00	4820.0	4919.0	5016.8	5012.666667	4904.888889	4819.666667	5221.645833	0
1940	2024-10-25 00:00:00+00:00	4770.0	4906.0	4992.6	5013.000000	4910.444444	4818.250000	5220.270833	0
1941	2024-10-28 00:00:00+00:00	4760.0	4892.0	4963.0	5013.833333	4914.000000	4818.250000	5218.541667	0
1942	2024-10-29 00:00:00+00:00	4700.0	4870.0	4930.0	5016.666667	4917.333333	4817.833333	5217.625000	0
1943	2024-10-30 00:00:00+00:00	4710.0	4846.0	4905.4	5018.500000	4921.000000	4817.166667	5216.416667	0

We use the default test-train split of 25% and 75%, respectively. The results of our training are presented in Table 4a. The results in the table present quite an interesting insight. Initially, we have a training accuracy of 97% and a loss of 0.37, which is good, but we then have a test loss of 2.23, which contrasts with the test accuracy of 97.07%. The test loss alone might suggest overfitting, but that is not implied by the test accuracy. This result may hence be due to the behavior of the binary cross entropy, which heavily penalizes incorrect prediction (Sudre et al).

Table 4a. Summary of model results.

Parameter Name	**Value**
Training Accuracy	99.51%
Testing Accuracy	97.07%
Training Loss	0.37
Test Loss	2.22

We can observe the confusion matrix from figure 4a, which summarizes our results.

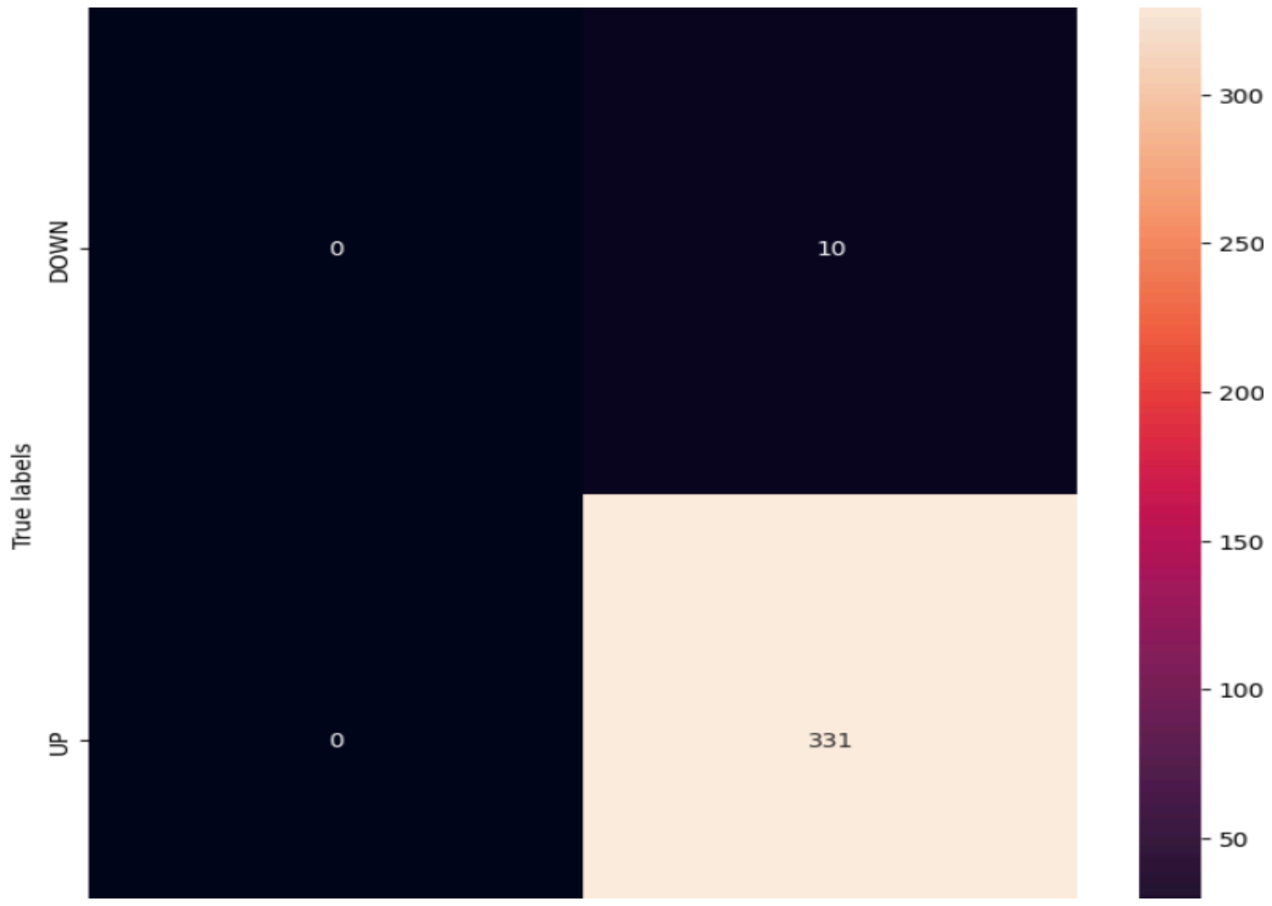


Figure 4a. Confusion Matrix for Prediction of Price Levels of BBRI.JK

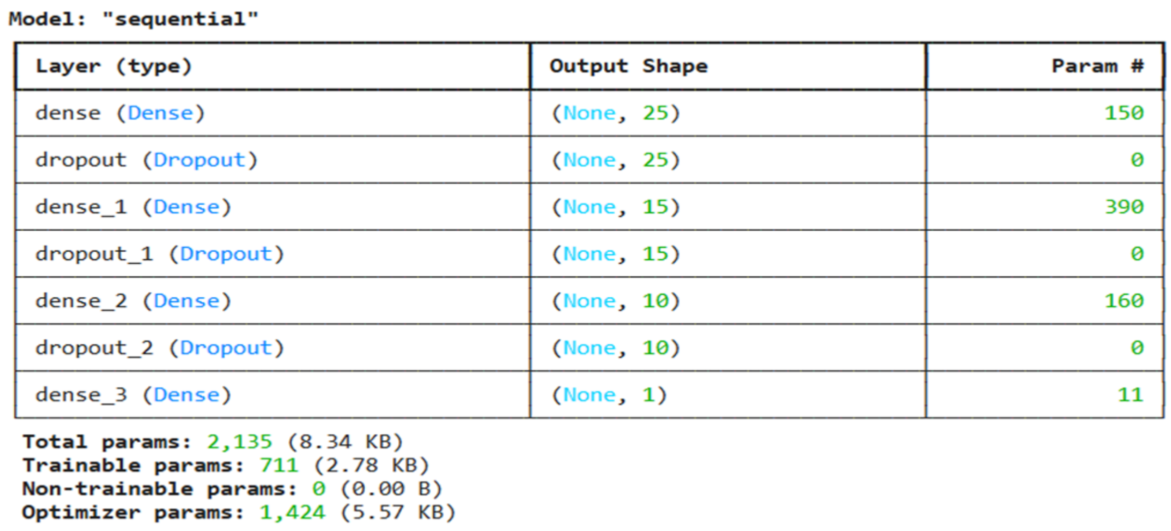
B.

In this task, we will rework all the tasks from Step 2.A. by using past information of the stationary time series as inputs. We then build and train the MLP model.

Using Module 1 from Deep Learning, we convert the Python code from Lesson 1 to Lesson 4 and use it for this project by converting the necessary data into log return. In Step 1.B., we have demonstrated that such log returns time series comprehended with stationary. Therefore, in this MLP model, we will directly use the log returns as stationary time series.

For modeling, we are using the Activation Function "RELU" for all the 3 hidden layers where hidden layers 1 will have 25 neurons, hidden layer 2 will have 15 neurons and hidden layer 3 will have 10 neurons. For this model, we are using dropout 20%. For the output layer, another activation function is being used instead of RELU, that is Sigmoid activation. For learning rate, we use 0.00001 with Adam as optimizer and binary cross entropy being used as loss function in this model.

Below is the model summary in Figure 5.bfor all the above criteria:



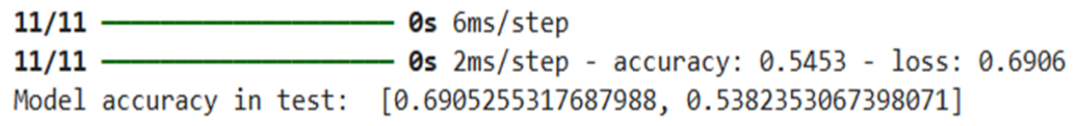
Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 25)	150
dropout (Dropout)	(None, 25)	0
dense_1 (Dense)	(None, 15)	390
dropout_1 (Dropout)	(None, 15)	0
dense_2 (Dense)	(None, 10)	160
dropout_2 (Dropout)	(None, 10)	0
dense_3 (Dense)	(None, 1)	11

```
Total params: 2,135 (8.34 KB)
Trainable params: 711 (2.78 KB)
Non-trainable params: 0 (0.00 B)
Optimizer params: 1,424 (5.57 KB)
```

Figure 5.b : MLP Model Summary

There are 2,135 parameters being used on this modeling. Once the model run, we got resultin figure 6.b:



```
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.5453 - loss: 0.6906
Model accuracy in test:  [0.6905255317687988, 0.5382353067398071]
```

Figure 6.b: MLP model metrics results

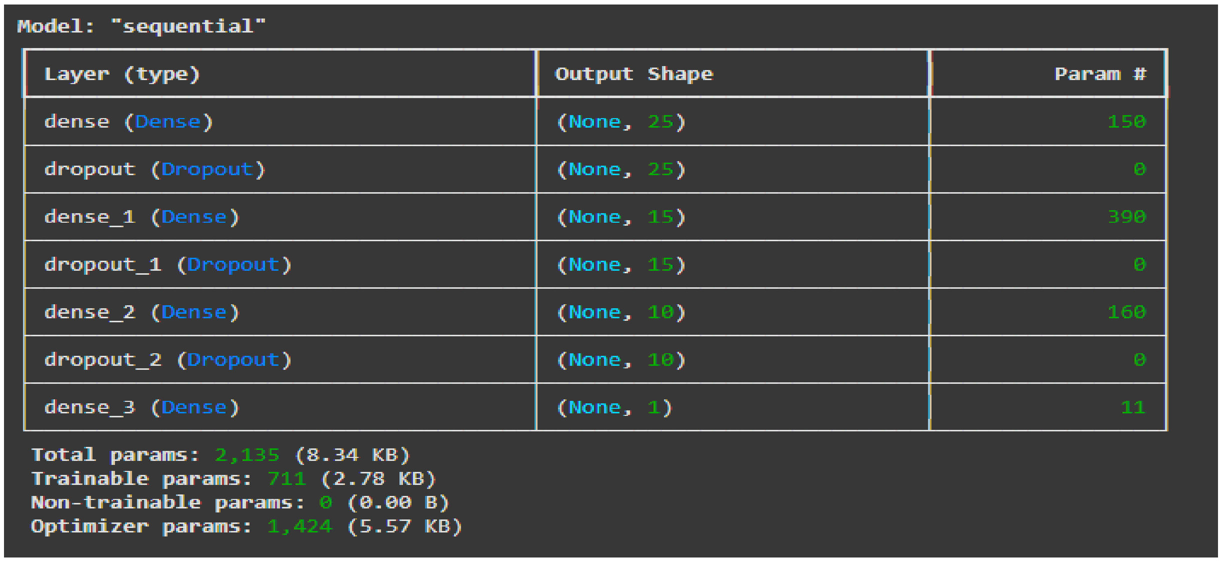
In terms of binary cross-entropy loss, the model resulted in 69%, which basically could be further improved. Meanwhile, in terms of accuracy, the model can only predict 54%, which is close to the probability of randomly flipping a coin. With such low accuracy, the model is not very helpful in making a prediction.

C.

For this task we will use our Fractionally differenced time series data and use this as an input to our MLP model. Let's see how this way of transformed data works when we build and train an MLP model on top of it.

We are going to use the same model as we have used in step 2 a. And b. Because our goal here is not to build the next fancy model but to understand how different types of transformed data perform on a given model and what type of edge it can provide us, so we will use the same model and compare the results.

We will quickly look into our model summary, in our model we have used 3 hidden layers 25, 15, 10 units respectively, each hidden layer uses the ReLU activation function and we have a dropout layer with a rate of 0.2. Our output layer is a single unit with a sigmoid activation function for our binary classification tasks. We are also using Adam optimizer with a learning rate of 1 X 10^-5 and Binary cross-entropy loss function.

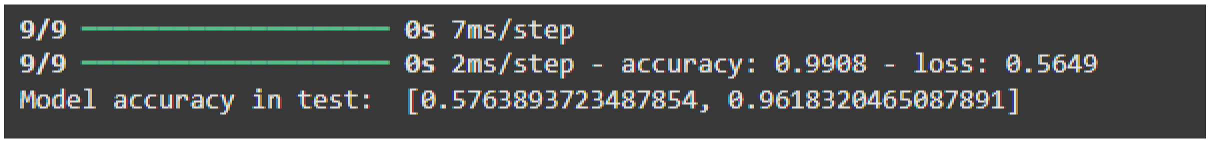


```
Model: "sequential"
┌─────────────────────────────┬─────────────────────────┬───────────┐
│ Layer (type)                │ Output Shape            │   Param # │
├─────────────────────────────┼─────────────────────────┼───────────┤
│ dense (Dense)               │ (None, 25)              │       150 │
├─────────────────────────────┼─────────────────────────┼───────────┤
│ dropout (Dropout)           │ (None, 25)              │         0 │
├─────────────────────────────┼─────────────────────────┼───────────┤
│ dense_1 (Dense)             │ (None, 15)              │       390 │
├─────────────────────────────┼─────────────────────────┼───────────┤
│ dropout_1 (Dropout)         │ (None, 15)              │         0 │
├─────────────────────────────┼─────────────────────────┼───────────┤
│ dense_2 (Dense)             │ (None, 10)              │       160 │
├─────────────────────────────┼─────────────────────────┼───────────┤
│ dropout_2 (Dropout)         │ (None, 10)              │         0 │
├─────────────────────────────┼─────────────────────────┼───────────┤
│ dense_3 (Dense)             │ (None, 1)               │        11 │
└─────────────────────────────┴─────────────────────────┴───────────┘
 Total params: 2,135 (8.34 KB)
 Trainable params: 711 (2.78 KB)
 Non-trainable params: 0 (0.00 B)
 Optimizer params: 1,424 (5.57 KB)
```

Figure 8.c: MLP model summary for Fractionally differenced time series

The model Summary will be the same and the task that we have also the same i.e. classification of stock prices, in which we are trying to predict up and down movement of a stock. We have done some feature engineering and built some features of returns of different time frames and the build and output of that if output value is "0" Stock is down if "1" Stock is up.

So, now let's see the result of our model and how it has performed in our fractional differenced data.



```
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9908 - loss: 0.5649
Model accuracy in test:  [0.5763893723487854, 0.9618320465087891]
```

Figure 9.c: Model metrics result

Above you can see the model metrics in Figure 9.c. The test loss (0.576) is close to the training loss (0.5649), indicating the model's predictions are relatively consistent across both datasets. And the main part Training accuracy (99.08%) is higher than the test accuracy (96.18%), which is typical and indicates some generalization gap though this small difference indicates that the model generalizes reasonably well without severe overfitting.

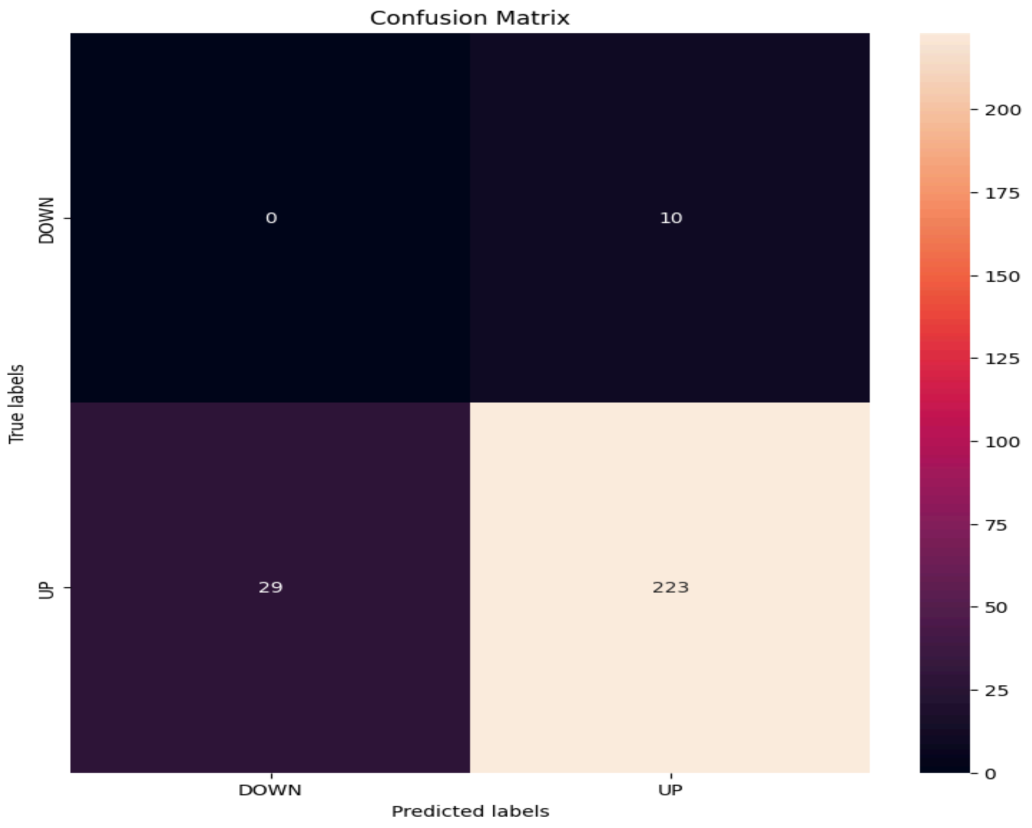


Figure 10.c: Confusion matrix

Overall, we can say the model has performed well on our transformed data, achieving high accuracy with a relatively small gap between training and test metrics.

STEP 3.

A.

In this section of the task, we aim to transform the given time series using the Gramian Angular Field tool in python, to images and perform a classification task on them. The first step is to use the window sliding method to create overlapping images using a window size of 30. After applying the window sizing method, we create an image as depicted in figure 5a. The various colors present in the image is due to the sliding window size that we applied. The next step is to use the GAF function to transform this representation into images like we earlier discussed. This representation is shown for the first batch and the 100th batch in figure 6a, and figure 7a respectively.

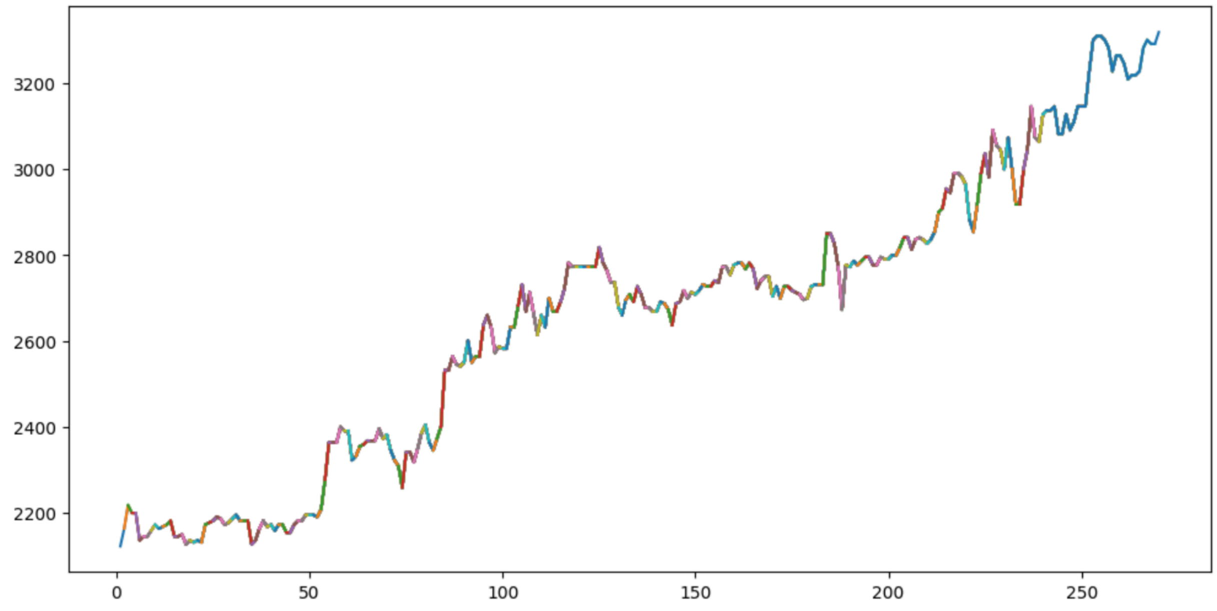


Figure 5a. Overlapping representation of the BBRI.JK using sliding window (window size=30)

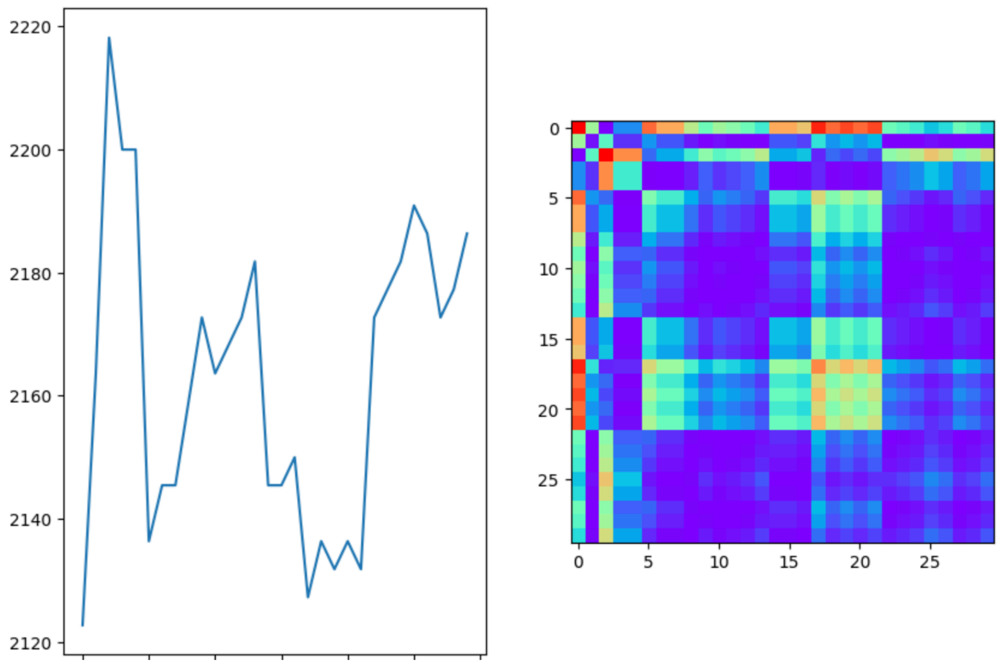


Figure 6a. First batch image representation of the original time series.

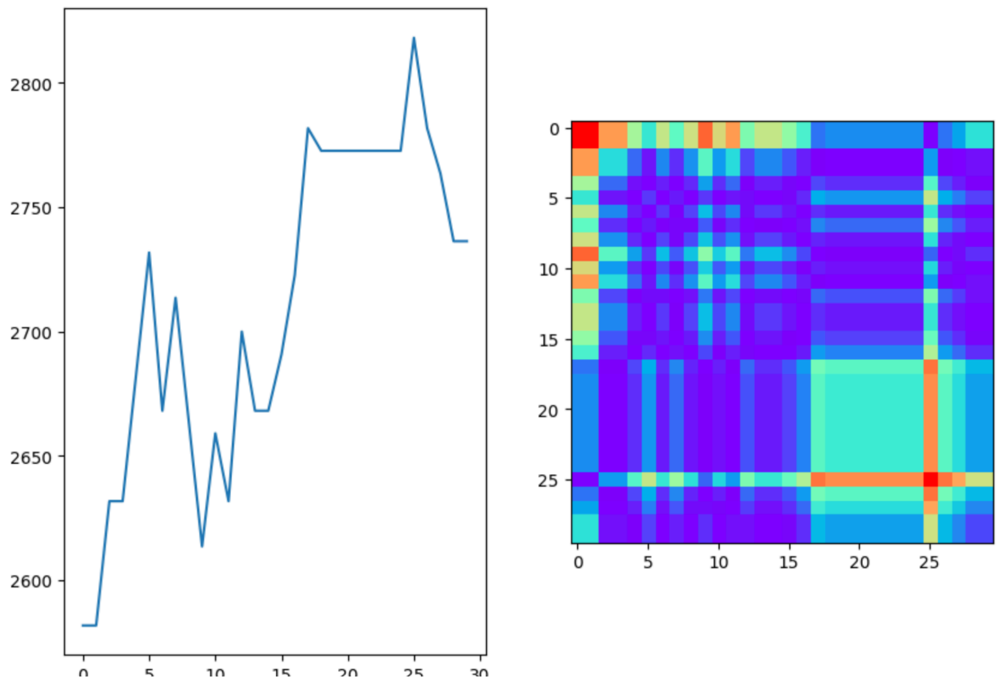
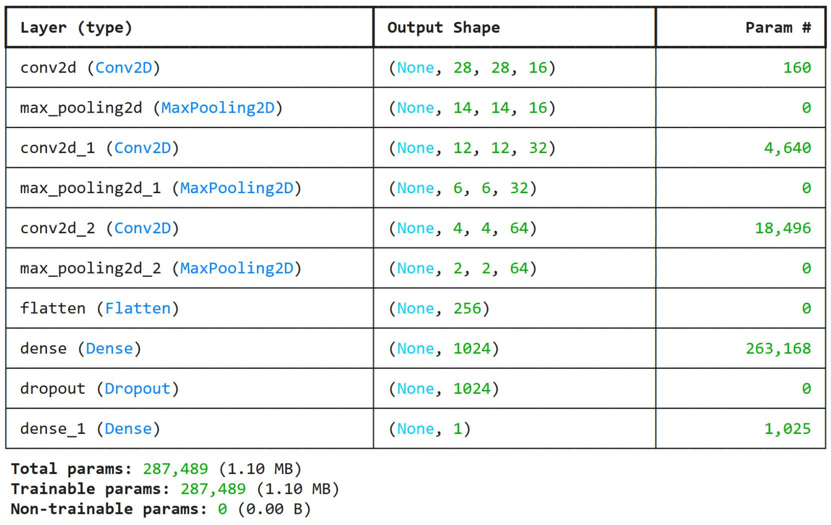


Figure 7a. Hundredth batch image representation of the original time series.

The data is split into 80 percent for the training set and then 20 percent for the test set. The summary of the model parameter used in training these images is presented in figure 8a.



```
┃ Layer (type)                    ┃ Output Shape           ┃       Param # ┃
│ conv2d (Conv2D)                 │ (None, 28, 28, 16)     │           160 │
│ max_pooling2d (MaxPooling2D)    │ (None, 14, 14, 16)     │             0 │
│ conv2d_1 (Conv2D)               │ (None, 12, 12, 32)     │         4,640 │
│ max_pooling2d_1 (MaxPooling2D)  │ (None, 6, 6, 32)       │             0 │
│ conv2d_2 (Conv2D)               │ (None, 4, 4, 64)       │        18,496 │
│ max_pooling2d_2 (MaxPooling2D)  │ (None, 2, 2, 64)       │             0 │
│ flatten (Flatten)               │ (None, 256)            │             0 │
│ dense (Dense)                   │ (None, 1024)           │       263,168 │
│ dropout (Dropout)               │ (None, 1024)           │             0 │
│ dense_1 (Dense)                 │ (None, 1)              │         1,025 │
 Total params: 287,489 (1.10 MB)
 Trainable params: 287,489 (1.10 MB)
 Non-trainable params: 0 (0.00 B)
```

Figure 8a. Model parameters for training image data for the original time series.

After training we obtain the results of our training process as follows as shown in table 5a.

Table 5a. Summary results for image training task.

Parameter Name	**Value**
Training Accuracy	66.91%
Validation Accuracy	60.27%

The low accuracy scores of both training and validation set, might point to the fact that we may need a more suited deep learning network for this task , as these scores are relatively low.The discrepancy in the results of the training and validation accuracy might suggest that the model is overfitting in the training data. As we can also observe, the validation accuracy is lower than the training accuracy. This might indicate that our model indeed struggles to generalize. As we already know, solely looking at the accuracy of our model might sometimes be misleading and not give us any relevant information on how to tweak certain parameters of our model to improve performance. Hence the need to look at and observe parameters like the precision, f1 score and recall.

Figure 9a provides the information on aforementioned parameters.

```
              precision    recall  f1-score   support

       False       0.44      0.19      0.26       142
        True       0.63      0.85      0.73       235

    accuracy                           0.60       377
   macro avg       0.54      0.52      0.50       377
weighted avg       0.56      0.60      0.55       377
```

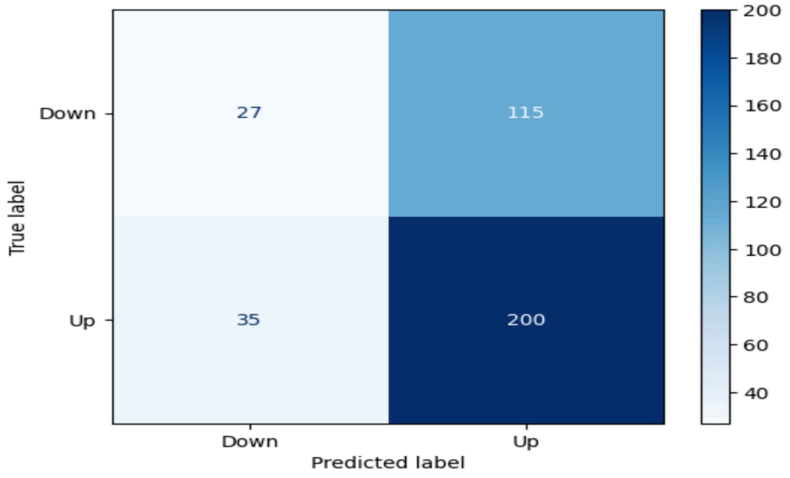


Figure 9a: Precision, recall, f1 score and confusion matrix for the model.

As we can observe from figure 9a, the true class has a higher precision, recall and f1 score than the false class. This indicates that the model does a good job at identifying the true class than it does at identifying the false class. This could be due to class imbalance in the data set used.Overall, this shows the model needs adjustments to be able to recognize false class, or possibly by tweaking the loss function to take the class imbalance to account for this false class.

B.

For this task, we are working based on the reference that we have learned from Module 3 of World Quant University on Deep Learning.

Transforming the stationary time series towards GAF is one of the first things that we need to do.

The resulting graphic for this transformation is shown below.

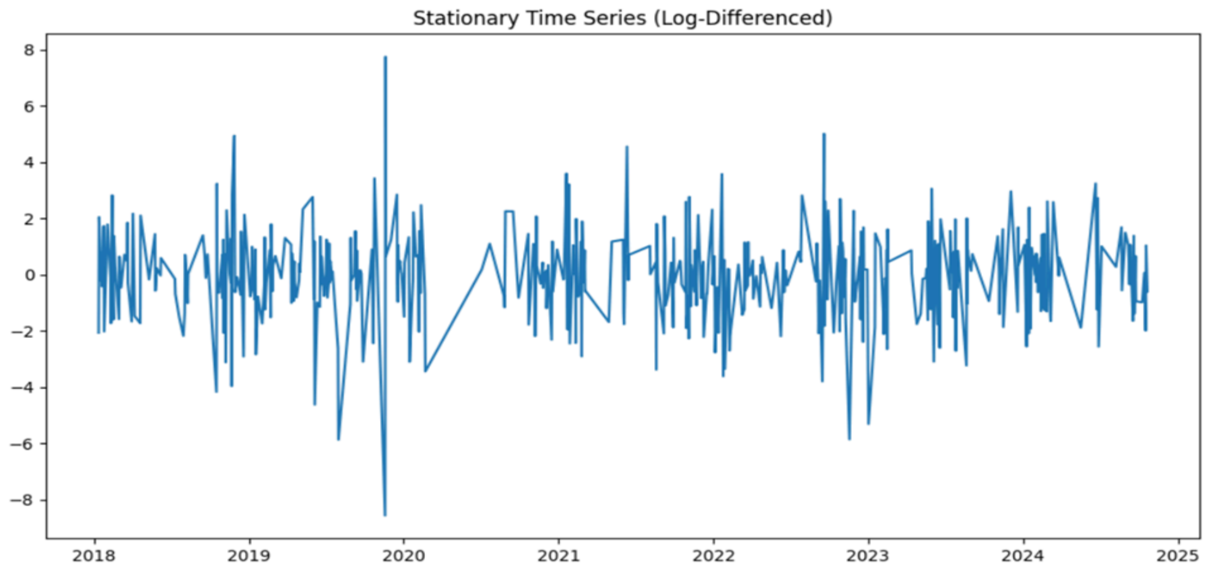


Figure 7.b: Stationary time series log-difference

To train the model, we will use the window sliding method to create an overlapping window of time series. For this model, we use window sliding of 30 with the result presented in the figure 8.b after this statement.

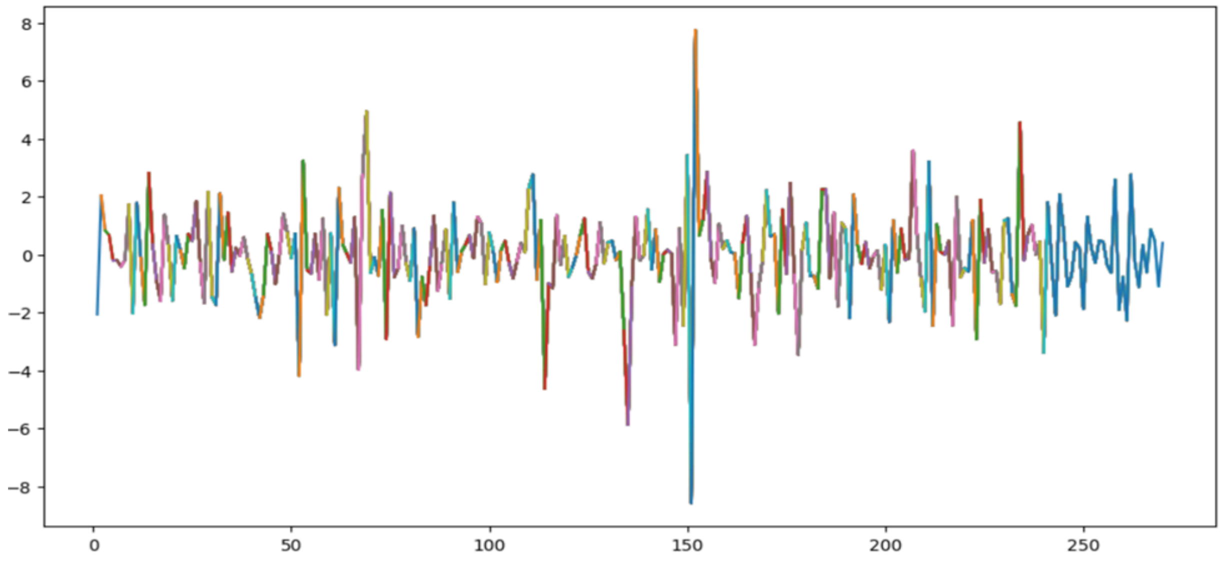
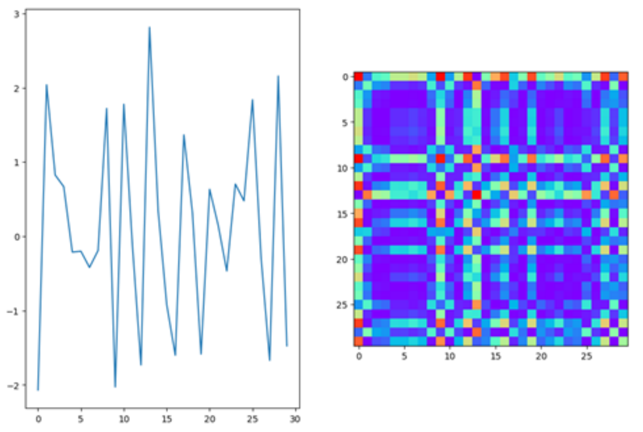


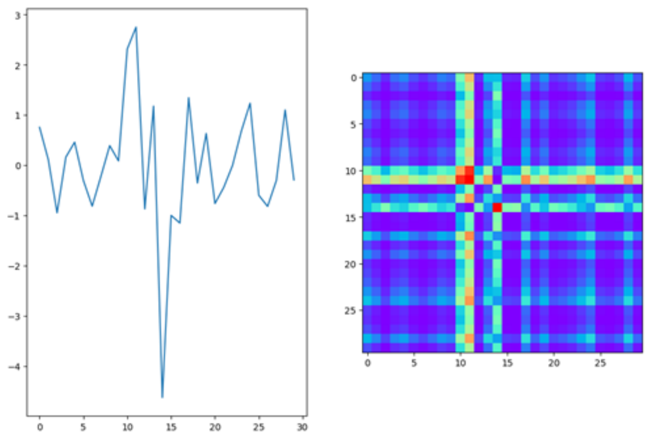
Figure 8.b: Overlapping representation of prices with 30 days sliding windows

The graphic indicates that the time series has been stationary as it is oscillating within the range -8 to +8. We can conclude that there is neither seasonality nor trend on the transformed time series.

We then convert each time series into images using the GAF (Gramian Angular Field method). We will show the first and the 100th batch of the time series and its image in figure 9.b..



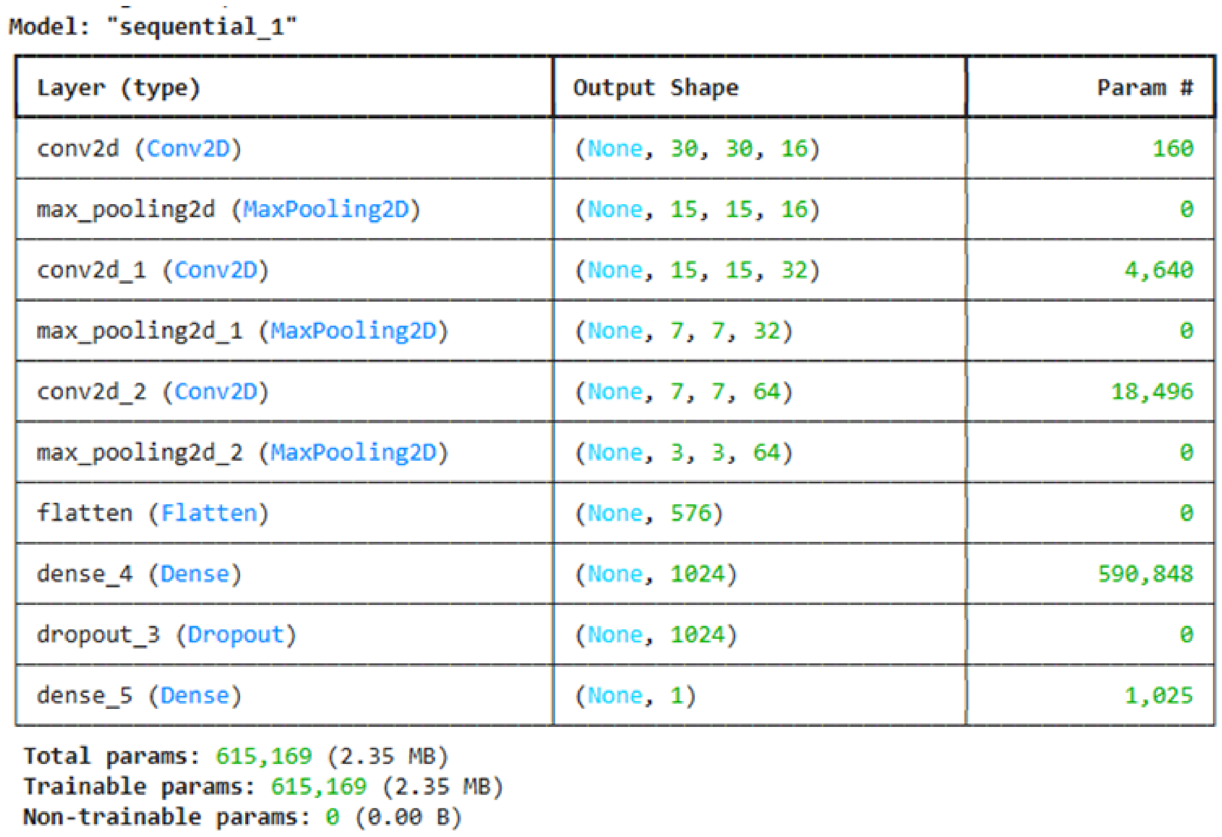
1st time series



100th time series

Figure 9.b: 1st and 100th batch image representation of the original time series.

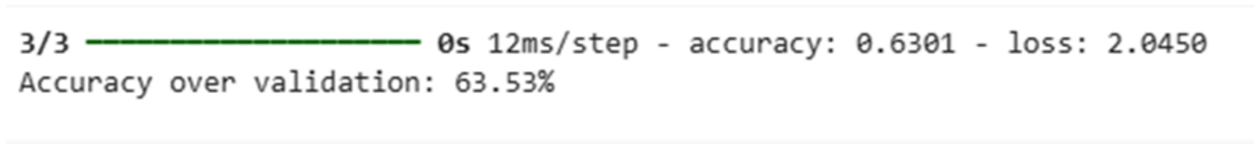
Here is the model summary that being used for further evaluation in figure 10.b:



```
Model: "sequential_1"
┌──────────────────────────────────┬──────────────────────────┬──────────┐
│ Layer (type)                     │ Output Shape             │  Param # │
├──────────────────────────────────┼──────────────────────────┼──────────┤
│ conv2d (Conv2D)                  │ (None, 30, 30, 16)       │      160 │
├──────────────────────────────────┼──────────────────────────┼──────────┤
│ max_pooling2d (MaxPooling2D)     │ (None, 15, 15, 16)       │        0 │
├──────────────────────────────────┼──────────────────────────┼──────────┤
│ conv2d_1 (Conv2D)                │ (None, 15, 15, 32)       │    4,640 │
├──────────────────────────────────┼──────────────────────────┼──────────┤
│ max_pooling2d_1 (MaxPooling2D)   │ (None, 7, 7, 32)         │        0 │
├──────────────────────────────────┼──────────────────────────┼──────────┤
│ conv2d_2 (Conv2D)                │ (None, 7, 7, 64)         │   18,496 │
├──────────────────────────────────┼──────────────────────────┼──────────┤
│ max_pooling2d_2 (MaxPooling2D)   │ (None, 3, 3, 64)         │        0 │
├──────────────────────────────────┼──────────────────────────┼──────────┤
│ flatten (Flatten)                │ (None, 576)              │        0 │
├──────────────────────────────────┼──────────────────────────┼──────────┤
│ dense_4 (Dense)                  │ (None, 1024)             │  590,848 │
├──────────────────────────────────┼──────────────────────────┼──────────┤
│ dropout_3 (Dropout)              │ (None, 1024)             │        0 │
├──────────────────────────────────┼──────────────────────────┼──────────┤
│ dense_5 (Dense)                  │ (None, 1)                │    1,025 │
└──────────────────────────────────┴──────────────────────────┴──────────┘
 Total params: 615,169 (2.35 MB)
 Trainable params: 615,169 (2.35 MB)
 Non-trainable params: 0 (0.00 B)
```

Figure 10.b: Model summary for CNN model

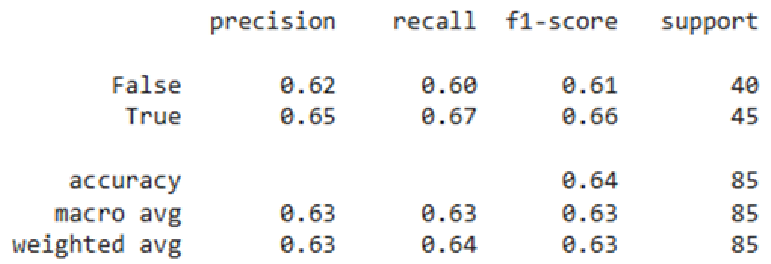
Upon running the model, we check the accuracy over validation on the model in figure 11.b with result 63% accuracy.



```
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.6301 - loss: 2.0450
Accuracy over validation: 63.53%
```

Figure 11.b: Model metrics for log transformed time series

We further analyze the model for Precision, Recall, F1-score as shown on the below screenshot in figure 12.b.



```
              precision    recall  f1-score   support

       False       0.62      0.60      0.61        40
        True       0.65      0.67      0.66        45

    accuracy                           0.64        85
   macro avg       0.63      0.63      0.63        85
weighted avg       0.63      0.64      0.63        85
```

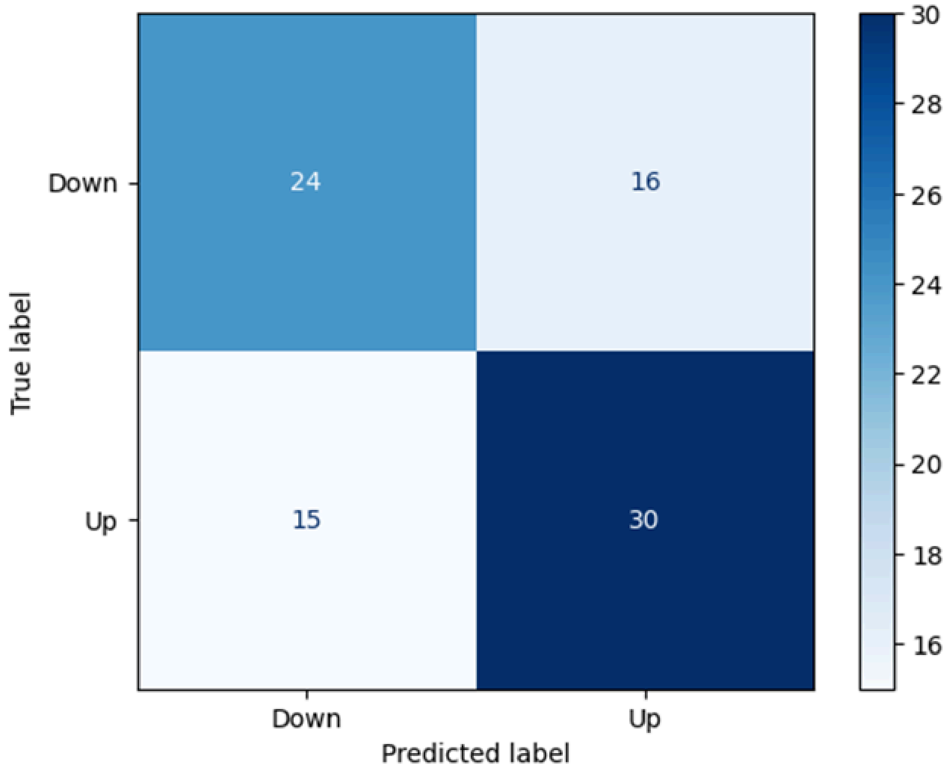


Figure 12. b: Precision, recall, f1 score and confusion matrix for the model.

C.

For this part we are going to work on a very different approach we will again use the same fractionally time series data that we have obtained in Step 1.c. but this time we will use GAF representation of the time series data and build and train a CNN model on top of it and try to predict the time series.

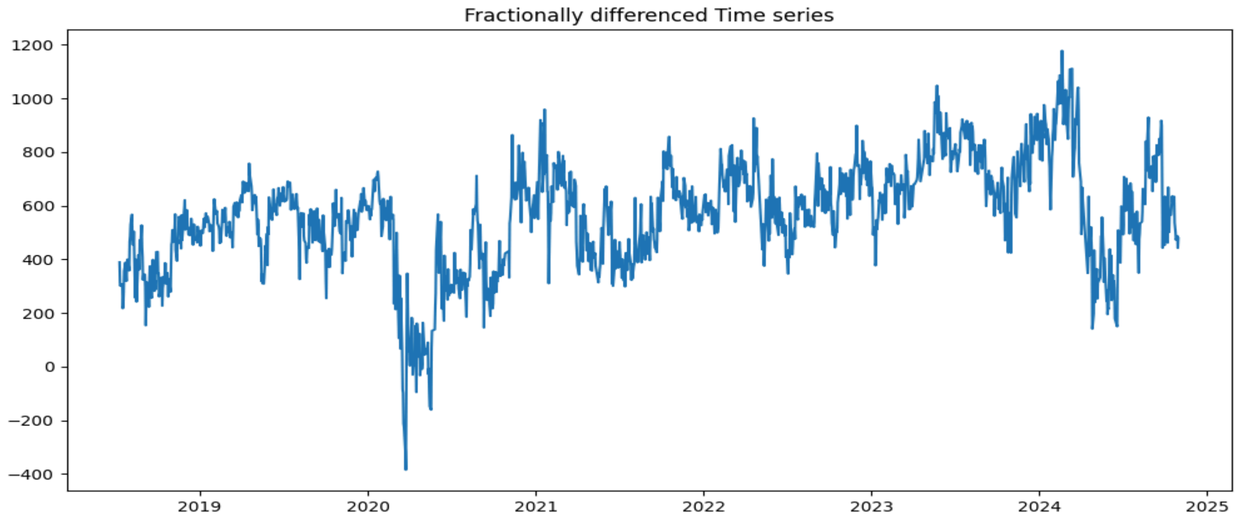


Figure 11.c : Fractionally differenced time series with d = 0.3

This above image in Figure 11.c. is the plot of our fractionally differenced data. We will now build a matrix by stacking a time series of 30 observations, in the image below how the matrix represents information about the 1524 time series of 30 observations on overlapping intervals.

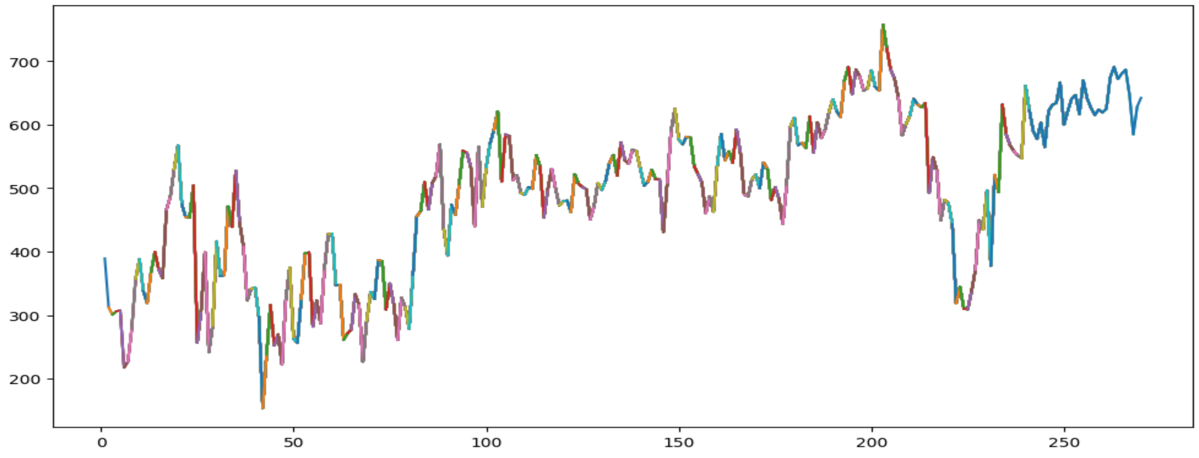
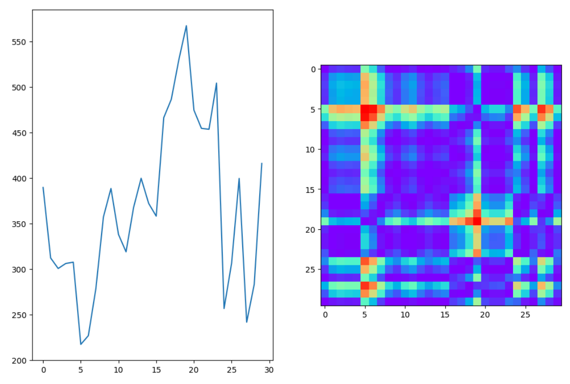


Figure 12.c : Overlapping representation of prices with 30 days sliding windows

You can see the the plot of stacked time series of 30 observations in Figure 12.c, Now we will apply the GAF transformation and we will see some illustration of some observations.we have a sample of 1524 images, to which we can attach a label whether the price went up or down over the following days, and train a Convolutional Neural Network. We can see the first and the 100th batch of the time series and its image in Figure 13.c.


1st Time Series GAF

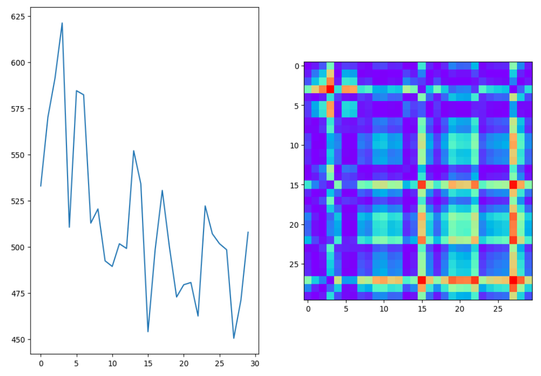
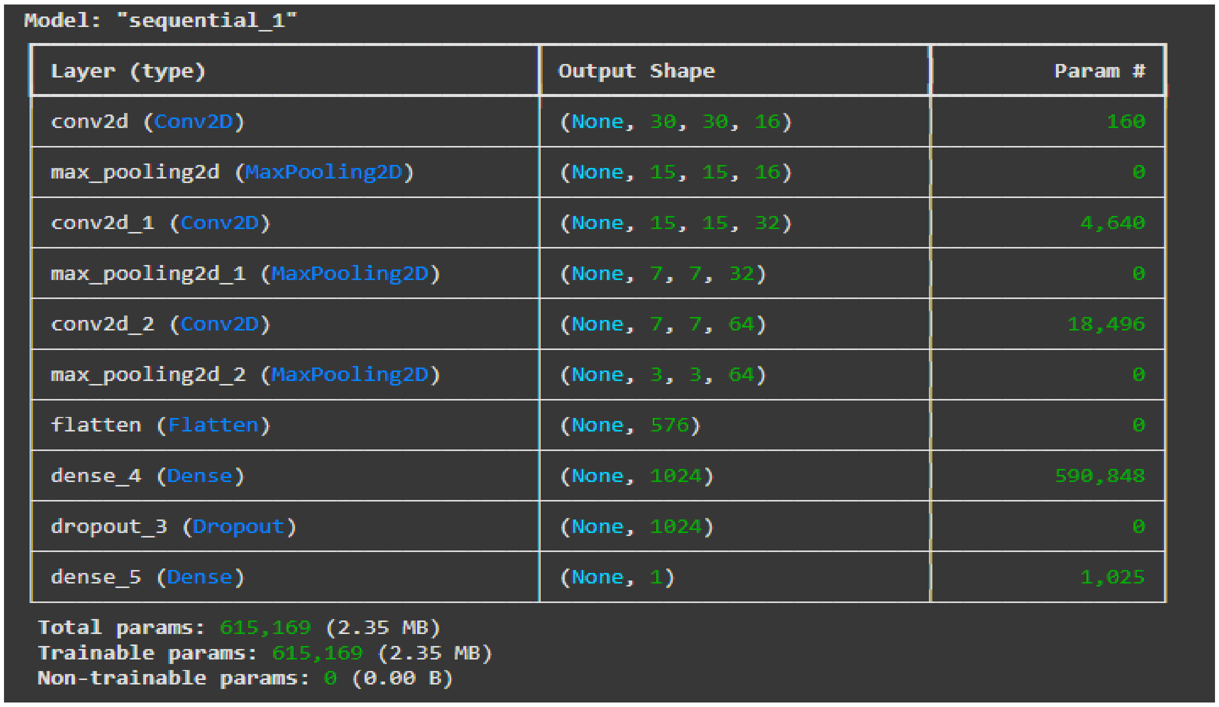

100th Time Series GAF

Figure 13.c: 1st and 100th batch image representation of the original time series.

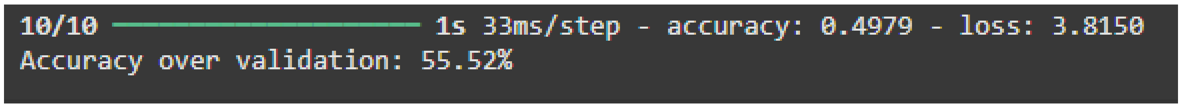
Now we will use the same CNN model as Step 3.b. For the input layer we have an input shape of (30,30,1), i.e. our grayscale images of 30 X 30 pixels. Now let's discuss convolution layers. Our First Convolution layer has 16 filters each of 3x3 with RELU activation, with a max pooling layer of size 2x2. The second convolution layer has 32 filters of size 3x3 with RELU activation function and a second pooling layer of pooling size 2x2. We have a third convolution layer with 64 filters size 3x3 with RELU activation function and a second pooling layer of pooling size 2x2. After that, we have a flattened layer which flattens the 2D output from the convolutional layers into a 1D vector. Then we have dense layers in which we have the first dense layer of 1024 units with a RELU activation function and a dropout layer with a dropout rate of 0.5 to prevent overfitting and at last we have an Output layer of 1 unit with a sigmoid activation function, suitable for binary classification tasks.



```
Model: "sequential_1"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━┓
┃ Layer (type)                         ┃ Output Shape                ┃         Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━┩
│ conv2d (Conv2D)                      │ (None, 30, 30, 16)          │             160 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ max_pooling2d (MaxPooling2D)         │ (None, 15, 15, 16)          │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ conv2d_1 (Conv2D)                    │ (None, 15, 15, 32)          │           4,640 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ max_pooling2d_1 (MaxPooling2D)       │ (None, 7, 7, 32)            │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ conv2d_2 (Conv2D)                    │ (None, 7, 7, 64)            │          18,496 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ max_pooling2d_2 (MaxPooling2D)       │ (None, 3, 3, 64)            │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ flatten (Flatten)                    │ (None, 576)                 │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dense_4 (Dense)                      │ (None, 1024)                │         590,848 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dropout_3 (Dropout)                  │ (None, 1024)                │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dense_5 (Dense)                      │ (None, 1)                   │           1,025 │
└──────────────────────────────────────┴─────────────────────────────┴─────────────────┘
 Total params: 615,169 (2.35 MB)
 Trainable params: 615,169 (2.35 MB)
 Non-trainable params: 0 (0.00 B)
```

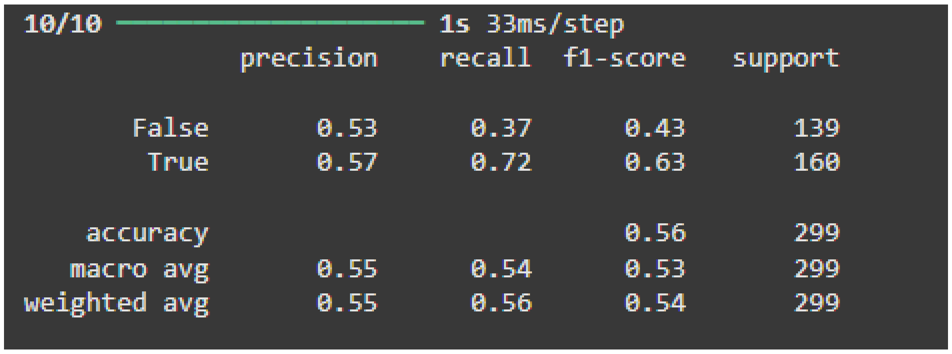
Figure 14.c: Model summary for CNN model for fractionally differenced time series

As you can see the model summary in figure 14.c in the above image now we will see how our model have performed.



```
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - accuracy: 0.4979 - loss: 3.8150
Accuracy over validation: 55.52%
```

Figure 15.c: Model metrics for fractionally differenced time series



```
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step
              precision    recall  f1-score   support

       False       0.53      0.37      0.43       139
        True       0.57      0.72      0.63       160

    accuracy                           0.56       299
   macro avg       0.55      0.54      0.53       299
weighted avg       0.55      0.56      0.54       299
```

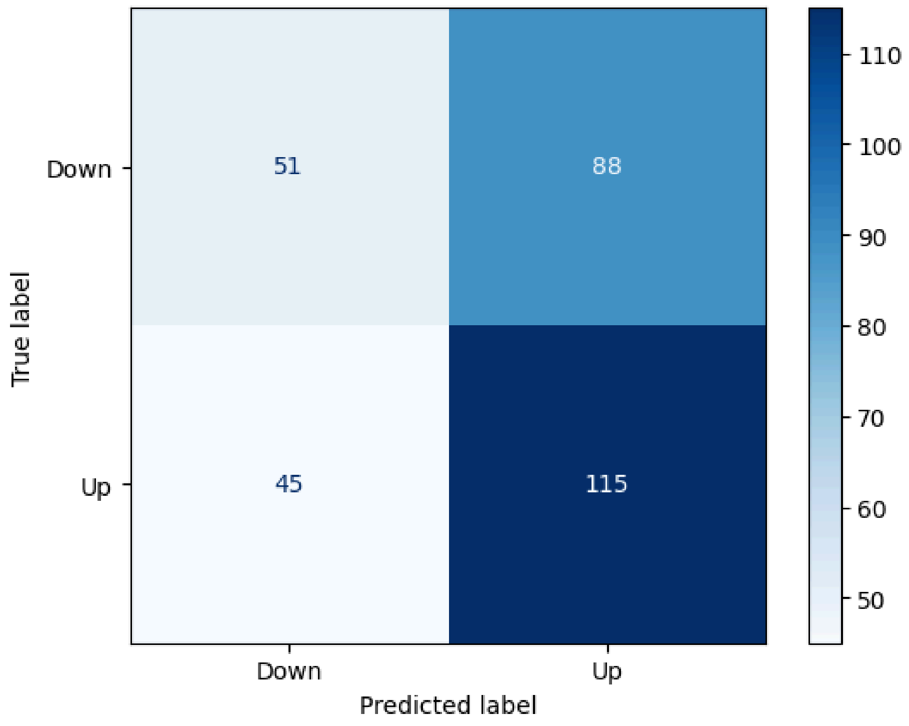


Figure 15. c: Precision, recall, f1 score and confusion matrix for the model.

In the figure 14.c we can see Accuracy is 49.79% suggests the model is barely better than random guessing (for a binary classification task). And loss of 3.8150 is quite high, indicating the model has not learned well during training. The model achieved 55.52% accuracy on the validation dataset, which is slightly better than random guessing but still indicates poor performance for most practical applications. This suggests the model is struggling to generalize to unseen data.

If we talk about overall metrics in Figure 15.c :

- Accuracy: 56% confirms that the model's overall classification performance is mediocre.
- Macro Avg: Averaged metrics across both classes give:
    - Precision: 0.55, Recall: 0.54, F1-Score: 0.53
    - These averages are unweighted, treating both classes equally.
- Weighted Avg: Weighted by class support:
    - Precision: 0.55, Recall: 0.56, F1-Score: 0.54
    - Reflects the imbalance in the dataset.

So we can conclude that the CNN model using GAF transformation based on fractionally differenced data did not perform well and is not very suitable for this kind of task.

Step 4.

For the MLP and CNN results obtained for the original time series as presented in steps.1a and 1b respectively, we clearly see that the MLP far outperforms the CNN architecture. This might be due to a few reasons. Firstly, the MLP directly works on the raw untransformed data. Consequently, it will be able to capture existing temporal relationships existing between the variables in the original time series. This greatly contrasts when we used the CNN algorithm to train image representations of the data. This is because the GAF algorithm introduces an added complexity into the data that might not have been captured by the CNN architecture. Added to this complexity could be the fact that our time series data is not stationary, making it even more difficult for the CNN architecture to learn patterns. As a suggestion to improve future performance, we may incorporate different time series to image techniques and study how they may in fact perform better than we observe with the CNN architecture.

Related to MLP and CNN results for Transformed Time Series as shown in Step 2. B and Step 3. B., considering that we are working on a transformed time series where the transformed time series is stationary based on several tests that have been conducted on it. Considering log return transformed time series is stationary, MLP algorithm outperforms the CNN algorithm since there are not many complex patterns that need to be identified by CNN algorithm. Therefore, the reason that MLP accuracy is better than CNN is due to the transformed log return time series not having too many complex patterns that need to be recognized.

Now let's talk about models trained on fractionally trained time series. As we have already seen the results in step 2.c. and step 3.c., we found that the MLP model worked far superior that the CNN model based on GSF transformation. MLP outperformed because it processes data globally, capturing the long-term dependencies and smooth transitions inherent in fractionally differenced time series, without relying on localized spatial patterns. And we have already discussed that Fractional differencing retains memory and subtle trends on the other hand CNN is designed for localized pattern recognition which struggled to find meaningful patterns from GAF transformation as it lacked strong localized patterns. The transformation may be a possible reason to introduce noise or fail to preserve long term memory.

References

1. ElvisEspinal. *Time series analysis using fractional differencing*. https://www.kaggle.com/code/elvisesp/time-series-analysis-using-fractional-differencing/notebook.
2. Maitra, and Sarit. “Time-Series Forecasting: Unleashing Long-Term Dependencies with Fractionally Differenced Data.” *2023 IEEE 64th International Scientific Conference on Information Technology and Management Science of Riga Technical University (ITMS). IEEE, 2023*, 2023, https://arxiv.org/pdf/2309.13409.
3. Sudre, Carole H., et al. “Generalized Dice Overlap as a Deep Learning Loss Function for Highly Unbalanced Segmentations.” *Lecture Notes in Computer Science*, Jan. 2017, pp. 240–48. https://doi.org/10.1007/978-3-319-67558-9_28.