The final submission Document / Report for The Dreamers

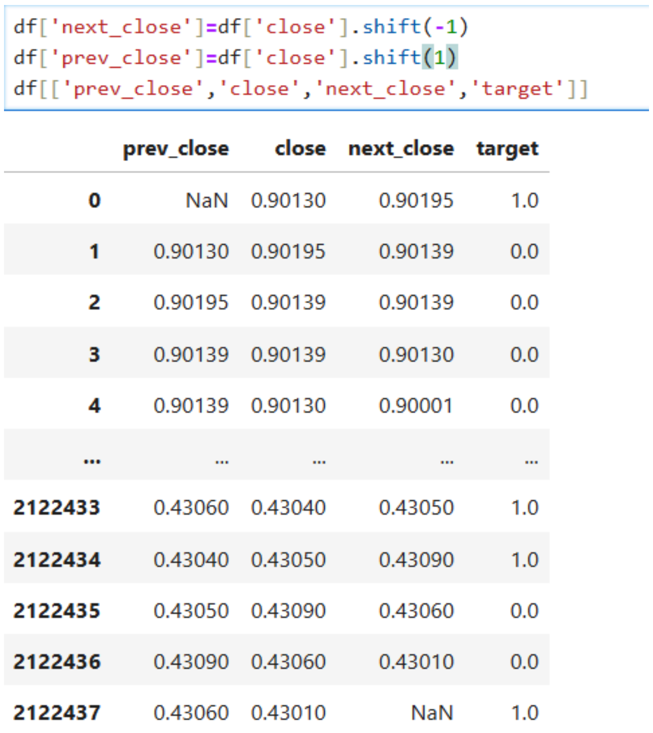
Note: Due to Computation constraints, I can't run all the models, that I have tested in the final submission notebook which is why I will use this file as an explanation document with screenshots of performance, etc (Updated doc link)

In my initial thought process document, I made a mistake which is assuming the target was supposed to be related to the next opening price and current closing price,

But after further exploration of the dataset, I found that the target is

- 0 if the current close is greater than or equal to the next close
- 1 if the next close is greater than the current close

Without the 'next_close' attribute which is obtained by shifting up 'close' we get



```
df['next_close']=df['close'].shift(-1)
df['prev_close']=df['close'].shift(1)
df[['prev_close','close','next_close','target']]
```

	prev_close	close	next_close	target
0	NaN	0.90130	0.90195	1.0
1	0.90130	0.90195	0.90139	0.0
2	0.90195	0.90139	0.90139	0.0
3	0.90139	0.90139	0.90130	0.0
4	0.90139	0.90130	0.90001	0.0
...	...	...	...	...
2122433	0.43060	0.43040	0.43050	1.0
2122434	0.43040	0.43050	0.43090	1.0
2122435	0.43050	0.43090	0.43060	0.0
2122436	0.43090	0.43060	0.43010	0.0
2122437	0.43060	0.43010	NaN	1.0

Such a relation (like the relation between next_close and 'close' and 'target' does not exist between 'close', 'prev_close', and 'target'

```
print('correlation of target and other features')
df.corr()['target']
```

```
correlation of target and other features
timestamp                   0.005468
open                       -0.004030
high                       -0.003995
low                        -0.004087
close                      -0.004100
volume                      0.015103
quote_asset_volume          0.012075
number_of_trades            0.014019
taker_buy_base_volume       0.013395
taker_buy_quote_volume      0.010717
target                      1.000000
next_close                 -0.002394
prev_close                 -0.004015
Name: target, dtype: float64
```

Even with the significant relation between next_close, and target the correlation is low because 'target' is a binary value, and also this is a time series data so it is difficult to capture the relationship across time

(ie price increase, and decrease through time, and there might be instances where 0.4159 might be for a target = 0 or for a target =1 this all depends on the previous close,)

there is no linear relationship between them that can be effectively captured by Pearson's correlation

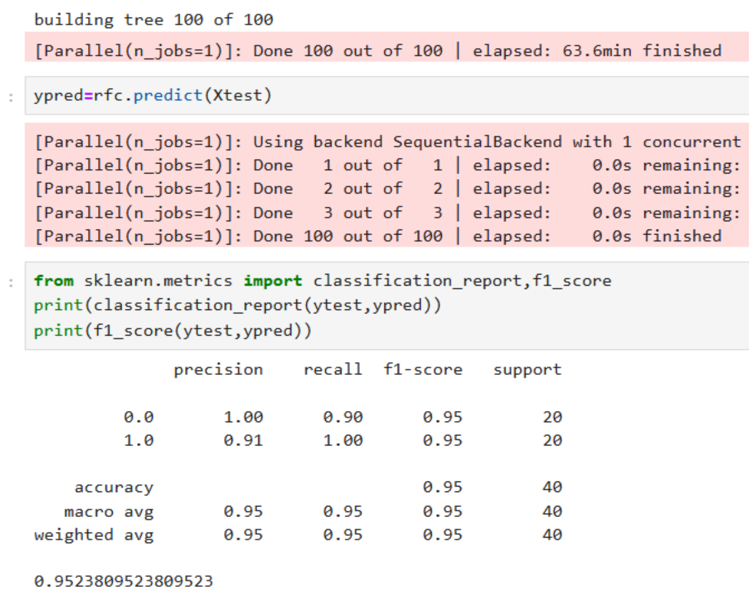
Random Forest

After adding next_close I tried using random first to see if there was any improvement because without 'next_close' or any 'sneak peek feature' (i call them features you get by using dataframe['feature'].shift(-n) where n is any i=number) normally the random forests,xgboost, etc gave an f1 score of around 0.60 (normal f1 but 0.50 in macro f1), 0.55,0.52 around this range

```
from sklearn.model_selection import train_test_split
x1=df.drop(['target'],axis=1)
y1=df['target']
Xtrain,Xtest,ytrain,ytest=train_test_split(x1,y1,test_size=40,shuffle=False)
```

```
from sklearn.ensemble import RandomForestClassifier
rfc=RandomForestClassifier(n_estimators=100,verbose=4).fit(Xtrain,ytrain)
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
building tree 1 of 100
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:   35.2s remaining:    0.0s
building tree 2 of 100
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:  1.1min remaining:    0.0s
building tree 3 of 100
```



```
building tree 100 of 100
[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed: 63.6min finished
```

```
ypred=rfc.predict(Xtest)
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.0s remaining:
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    0.0s remaining:
[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:    0.0s finished
```

```
from sklearn.metrics import classification_report,f1_score
print(classification_report(ytest,ypred))
print(f1_score(ytest,ypred))
```

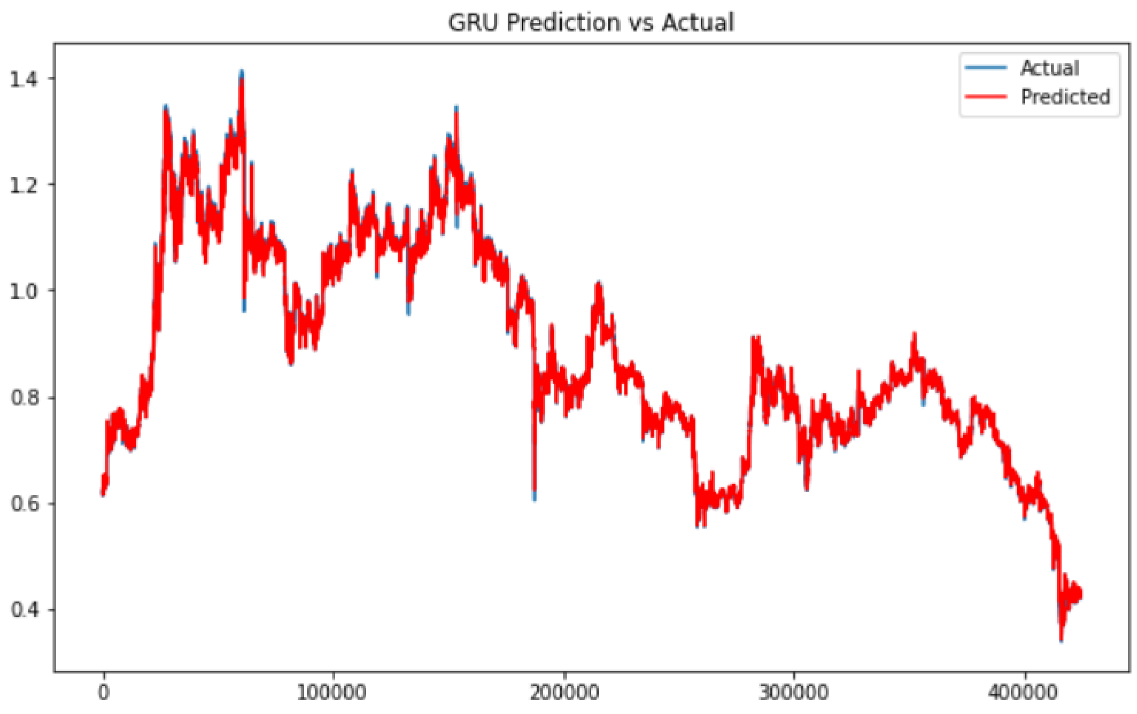
```
              precision    recall  f1-score   support

         0.0       1.00      0.90      0.95        20
         1.0       0.91      1.00      0.95        20

    accuracy                           0.95        40
   macro avg       0.95      0.95      0.95        40
weighted avg       0.95      0.95      0.95        40

0.9523809523809523
```

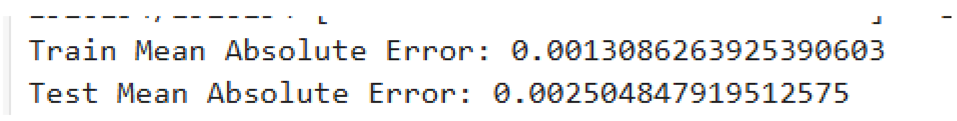
But after the addition of ‘next_close’ the model’s performance goes to 0.95

This shows the need to use ‘next_close’ as it is the close of next minute we need to forecast it and use it along with the current close to forecast the direction (+ve or -ve) for the cryptocurrency

So we focus on forecasting ‘next_close’ with a regression-based model

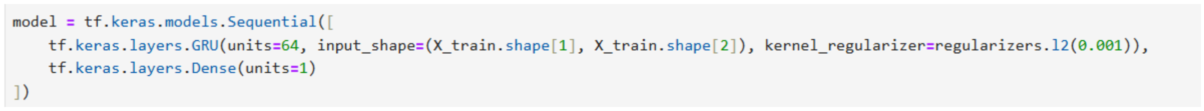
GRU base Model





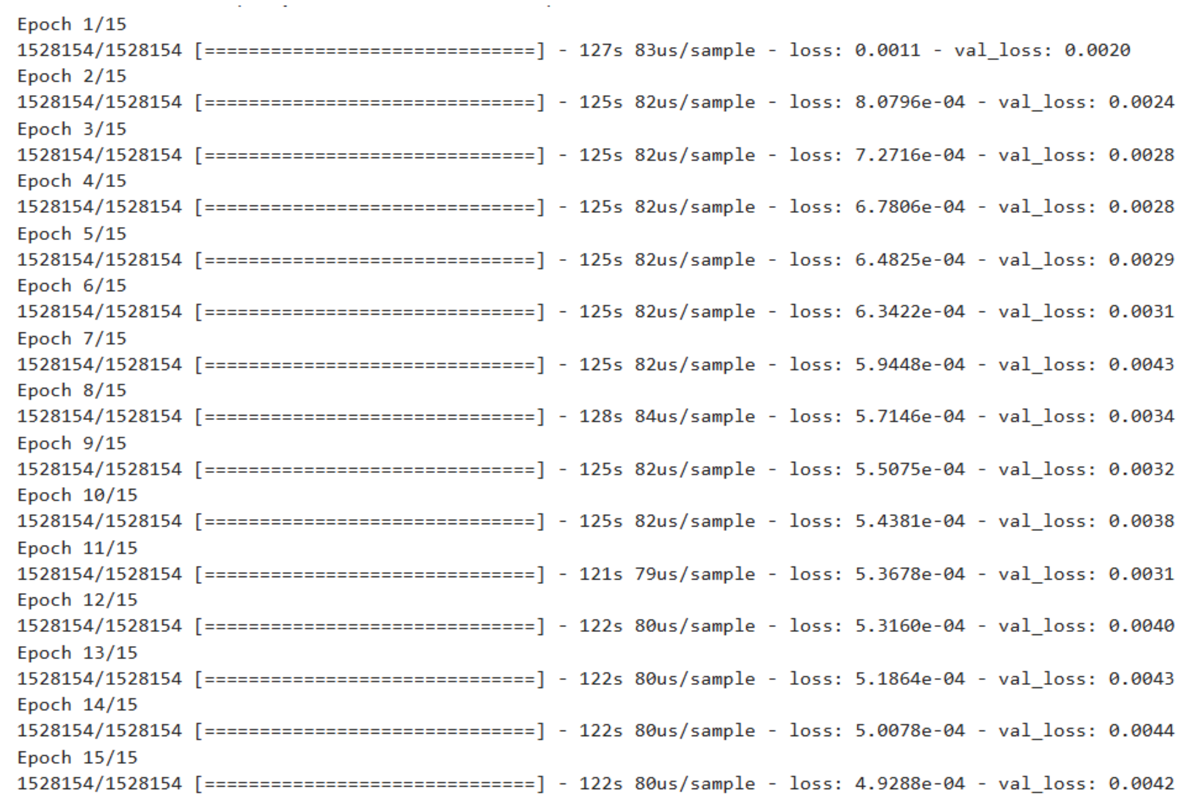
```
Train Mean Absolute Error: 0.0013086263925390603
Test Mean Absolute Error: 0.002504847919512575
```

The model is able to capture the general trend but it cannot properly give a result that we can compare against a reliable number, Gru model architecture



```
model = tf.keras.models.Sequential([
    tf.keras.layers.GRU(units=64, input_shape=(X_train.shape[1], X_train.shape[2]), kernel_regularizer=regularizers.l2(0.001)),
    tf.keras.layers.Dense(units=1)
])
```

The model was

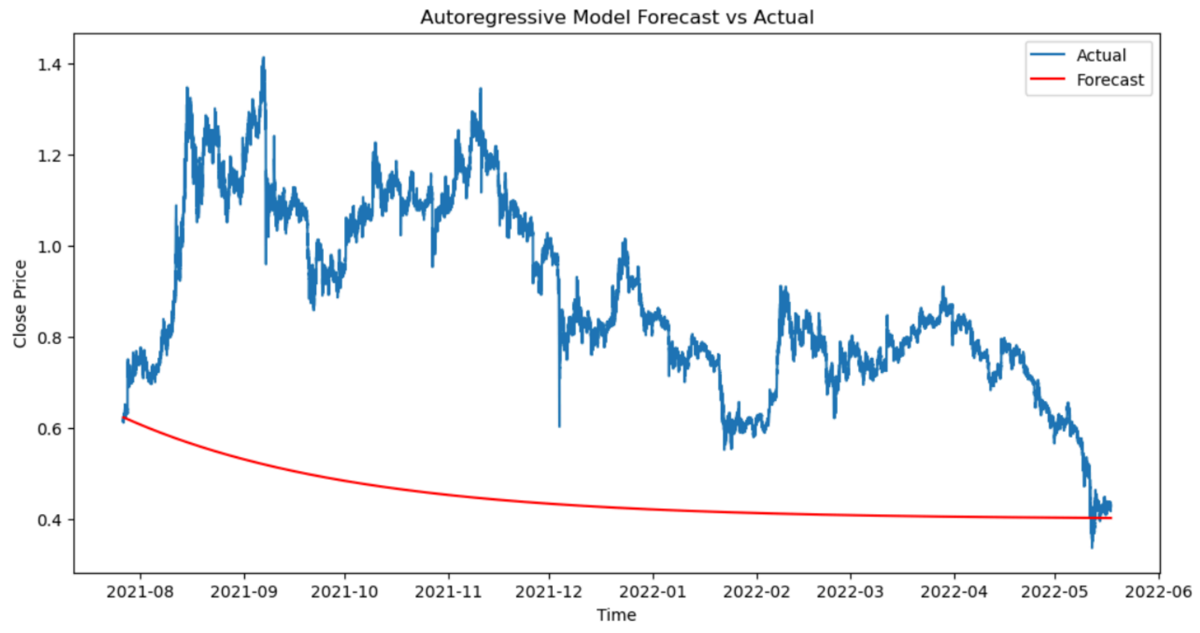


```
Epoch 1/15
1528154/1528154 [==============================] - 127s 83us/sample - loss: 0.0011 - val_loss: 0.0020
Epoch 2/15
1528154/1528154 [==============================] - 125s 82us/sample - loss: 8.0796e-04 - val_loss: 0.0024
Epoch 3/15
1528154/1528154 [==============================] - 125s 82us/sample - loss: 7.2716e-04 - val_loss: 0.0028
Epoch 4/15
1528154/1528154 [==============================] - 125s 82us/sample - loss: 6.7806e-04 - val_loss: 0.0028
Epoch 5/15
1528154/1528154 [==============================] - 125s 82us/sample - loss: 6.4825e-04 - val_loss: 0.0029
Epoch 6/15
1528154/1528154 [==============================] - 125s 82us/sample - loss: 6.3422e-04 - val_loss: 0.0031
Epoch 7/15
1528154/1528154 [==============================] - 125s 82us/sample - loss: 5.9448e-04 - val_loss: 0.0043
Epoch 8/15
1528154/1528154 [==============================] - 128s 84us/sample - loss: 5.7146e-04 - val_loss: 0.0034
Epoch 9/15
1528154/1528154 [==============================] - 125s 82us/sample - loss: 5.5075e-04 - val_loss: 0.0032
Epoch 10/15
1528154/1528154 [==============================] - 125s 82us/sample - loss: 5.4381e-04 - val_loss: 0.0038
Epoch 11/15
1528154/1528154 [==============================] - 121s 79us/sample - loss: 5.3678e-04 - val_loss: 0.0031
Epoch 12/15
1528154/1528154 [==============================] - 122s 80us/sample - loss: 5.3160e-04 - val_loss: 0.0040
Epoch 13/15
1528154/1528154 [==============================] - 122s 80us/sample - loss: 5.1864e-04 - val_loss: 0.0043
Epoch 14/15
1528154/1528154 [==============================] - 122s 80us/sample - loss: 5.0078e-04 - val_loss: 0.0044
Epoch 15/15
1528154/1528154 [==============================] - 122s 80us/sample - loss: 4.9288e-04 - val_loss: 0.0042
```

I added regularization to help reduce the overfitting and generalize the model but it did not help
(basically, it's getting the big picture, but not the fine details most models I tried are like this)
GRU's performance(mse,mae) was slightly better than an LSTM, Although overfitting was an issue in LSTM as well

Auto-Regressive Models

```
Mean Squared Error: 0.22098901562008394
```



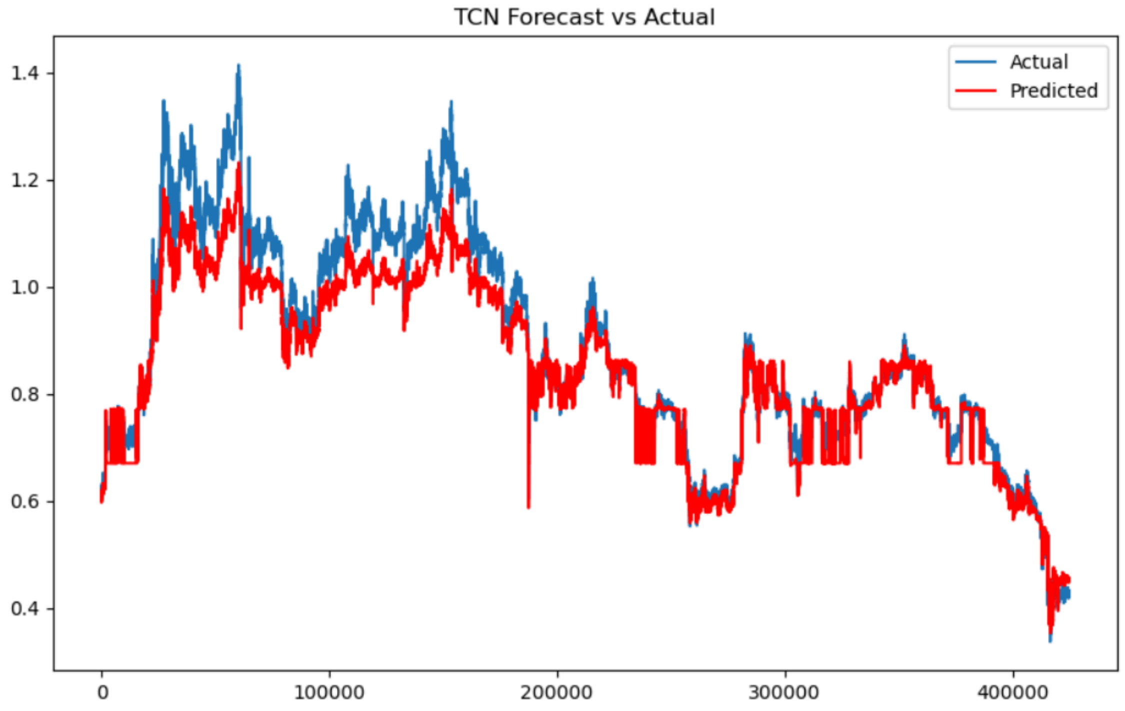
The autoregressive model did not perform well,

```
Mean Squared Error: 0.22098901562008394
```

Temporal CNN

I also tried using a Temporal CNN (using 1d conv layers across)

```
Mean Squared Error: 0.0036529665706238955
```



Temporal cnn was also overfitting, it have a training loss (mse) of 0.0004

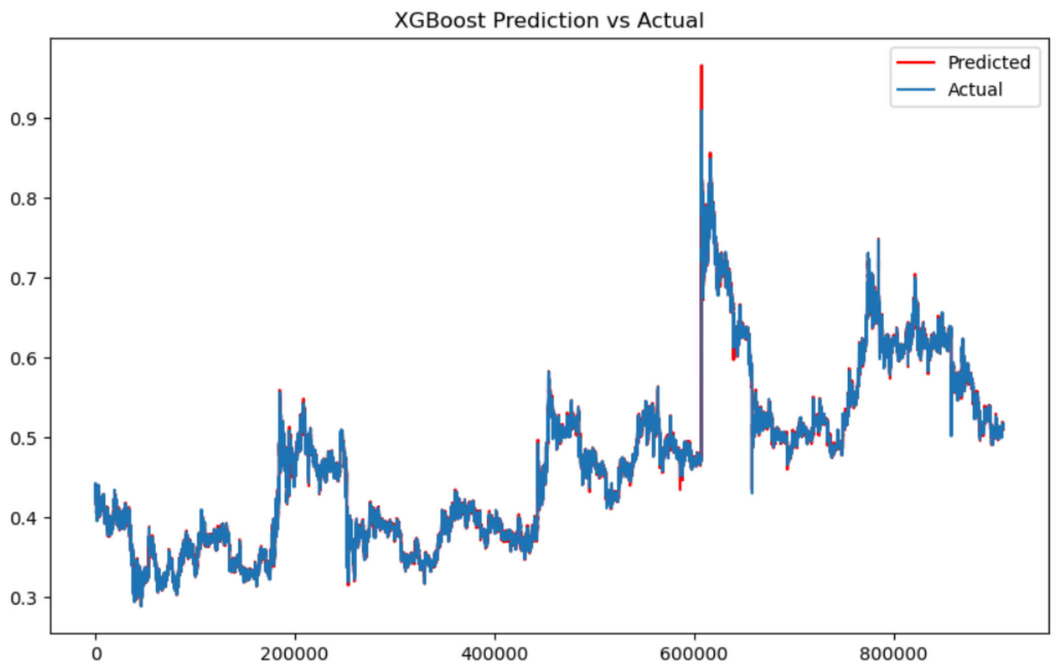
XGboost

XGboost was among the better models which gave a mse of about

```
Train Mean Squared Error: 9.350105181994123e-06
Test Mean Squared Error: 2.2137793165518896e-06
```

(train mse i passed training data to get train mse,)

```
Train Mean Absolute Error: 0.0007876253645049976
Test Mean Absolute Error: 0.0010759334568460016
```



I only tried getting F1 results for models with good mse/mae

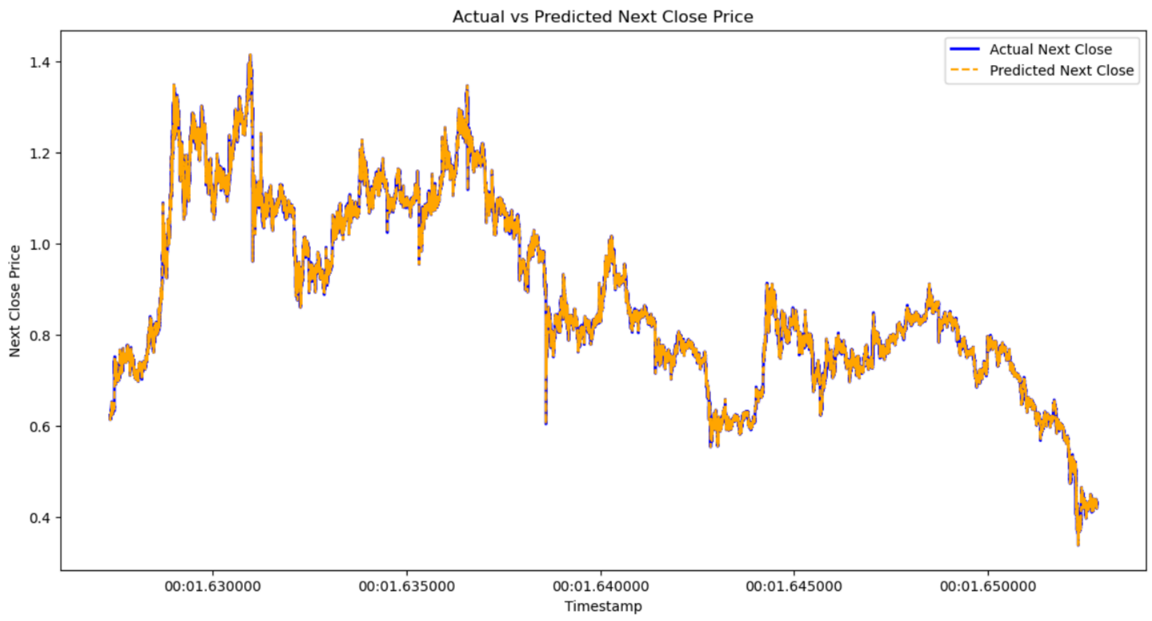
So for the xgboost model

```
from sklearn.metrics import f1_score,classification_report
print(classification_report(dft['target'],test['target']))
```

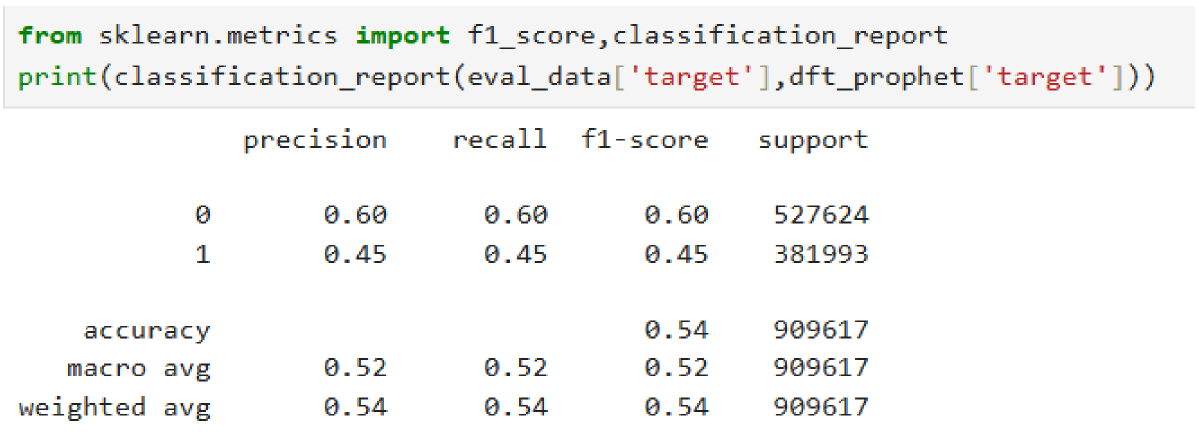
```
              precision    recall  f1-score   support

           0       0.58      0.49      0.53    527624
           1       0.42      0.52      0.46    381993

    accuracy                           0.50    909617
   macro avg       0.50      0.50      0.50    909617
weighted avg       0.51      0.50      0.50    909617
```

Prophet model
(note for this I have multiplied all features by 1000 to see if it gives better performance)
[only predicting ‘next_close’ price]

(note i did not plot the graph for prophet with scaling, but i ploted graph for unscaled prophet results)



F1 score(features scaled by 1000) [only ‘next_close’ was predicted]
(here eval_data[‘target’] is the actual ‘target’ for test data that i found using the “sneak peak” method pls refer jupyter-notebook at [link] for mode context



```
from sklearn.metrics import f1_score,classification_report
print(classification_report(eval_data['target'],dft_prophet['target']))
```

```
              precision    recall  f1-score   support

           0       0.60      0.60      0.60    527624
           1       0.45      0.45      0.45    381993

    accuracy                           0.54    909617
   macro avg       0.52      0.52      0.52    909617
weighted avg       0.54      0.54      0.54    909617
```

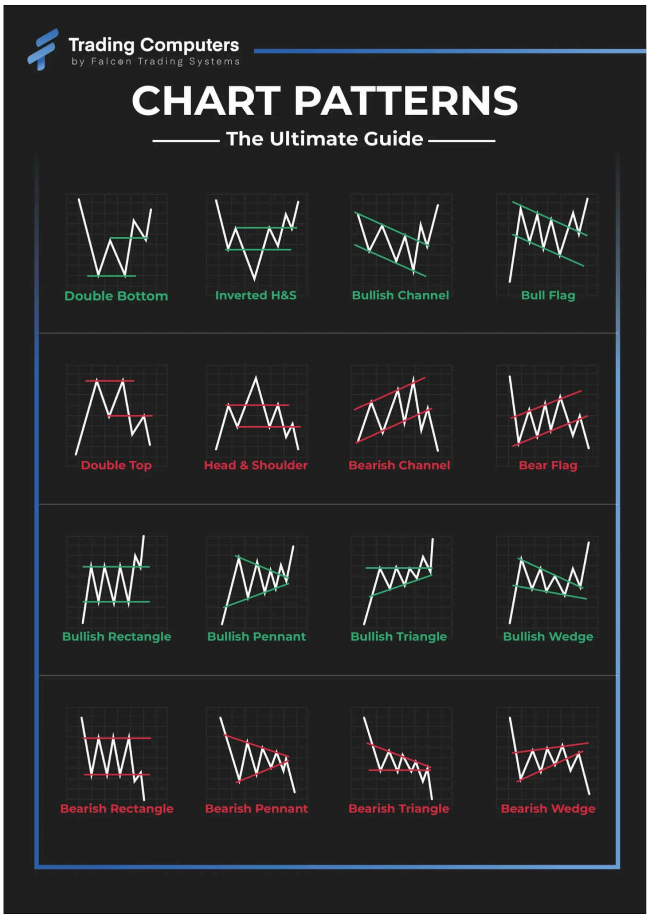
F1 score(features scaled by 1000)(after predicting both current close, and next_close)

```
              precision    recall  f1-score   support

           0       0.59      0.75      0.66    527624
           1       0.44      0.27      0.33    381993

    accuracy                           0.55    909617
   macro avg       0.51      0.51      0.50    909617
weighted avg       0.52      0.55      0.52    909617
```

The Main Idea here was because the model has an idea of the trend but cannot like give a proper number, we have the model predict both current close, and future close (2 separate models of same type) this way because both models understand the trend, we might be able to derive the target from them , and this way the model will also be able to give future predictions without needing 'next_close' attribute as an input

Other Method I would like to try

- Trying Custom indicators
- Plotting candle stick plots and using a CNN to figure out if there are any patterns like



And have the CNN model recognize patterns from the candle stick plot and have the system react accordingly (assuming these patterns work and have been tested and are reliable)