CSE 587 : Data Intensive Computing

Project Phase 2

Authors:
Yahsika Nihalani 50425015
yashikav@buffalo.edu
Ashutosh Shailesh Bhawsar 50416025
abhawsar@buffalo.edu

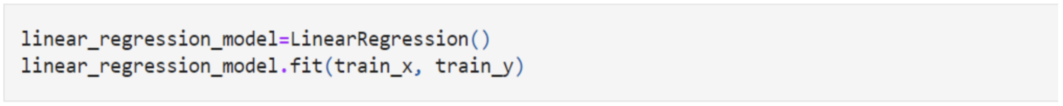
Instructor: **Prof. Eric Mikida**

Date Last Edited: November 14, 2022

1 Linear Regression

1.1 Reason for choosing this algorithm:

Linear regression uses independent variables to model a goal prediction value. We decided to choose this model to predict the stock price based on other factors like Closing price, Opening price, Volume_BTC, and Volume_Currency. This model will help determine the relationship between these variables and the weighted price. 85% of the data was used as train data, and the rest 15% was used as test data.

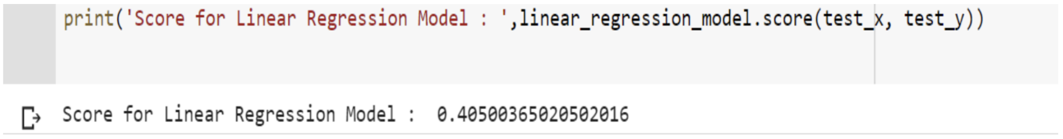


```
linear_regression_model=LinearRegression()
linear_regression_model.fit(train_x, train_y)
```

Figure 1: Fitting Linear Regression model

1.2 Effectiveness of this algorithm:

We can see from the graph and score that the model gives an accuracy of almost 40 to 45%. There is a large variety of independent variables, and we are trying to fit the price that is the dependent variable for these independent variables, so it is not a very good fit. Using the opening price, low price, and high price as features can perform well because it will reduce high variance in data and limit the train data within some boundaries.



```
print('Score for Linear Regression Model : ',linear_regression_model.score(test_x, test_y))
```

```
Score for Linear Regression Model :  0.40500365020502016
```

Figure 2: Score of Linear Regression model

1.3 Intelligence gained from this algorithm:

From the graph and score of the model, it can be understood that there are better ideas than trying to fit the model for such data with so much variance. Tuning specific parameters also resulted in approximated same accuracy. The model is under-fitted for this data set. This model can be further explored by limiting the features and then testing out the accuracy of those.

1.4 Results and Visualizations:

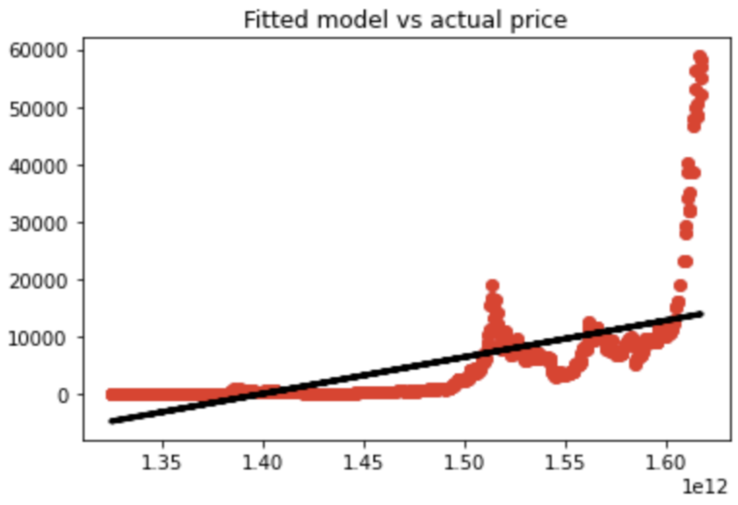


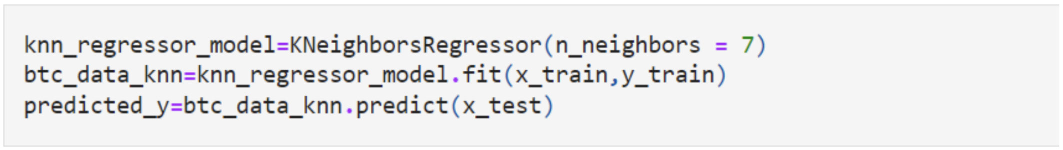
Figure 3: Model and BTC price

The graph shows that the model is **UNDER FITTED**. There is variance in data along with some outliers, while our predicted model is linearly fitted and does not consider this variance. Linear regression is better fitted for the data set, which does not have much variance and has a single trend that either the trend goes up or down.

2 KNN Regressor

2.1 Reason for choosing this algorithm:

The KNN regressor models predict the class of the output variable by calculating local probabilities. As this algorithm performs regression based on the neighborhood points, we decided to explore this model to predict the price of stocks. As linear regression is a linear model that was under-fitted, we decided to determine the performance of the KNN model.

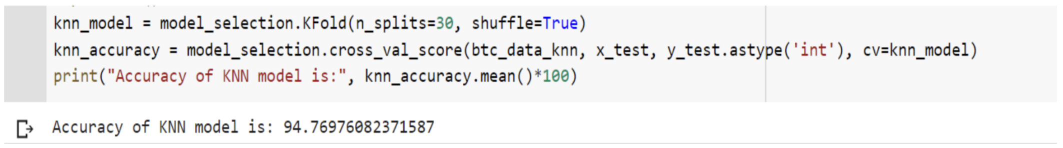


```
knn_regressor_model=KNeighborsRegressor(n_neighbors = 7)
btc_data_knn=knn_regressor_model.fit(x_train,y_train)
predicted_y=btc_data_knn.predict(x_test)
```

Figure 4: KNN model fitting

2.2 Effectiveness of this algorithm:

Running the KNN Regressor model for our train and test data, this KNN model gives a good accuracy of approximately 90%. The model was run against test data, and the accuracy of the model was predicted. This means that the predicted price by our model is almost 90% close to the actual price of the stocks. This shows that this model is a good option for predicting the price, unlike linear regression models.



```
knn_model = model_selection.KFold(n_splits=30, shuffle=True)
knn_accuracy = model_selection.cross_val_score(btc_data_knn, x_test, y_test.astype('int'), cv=knn_model)
print("Accuracy of KNN model is:", knn_accuracy.mean()*100)
```

```
Accuracy of KNN model is: 94.76976082371587
```

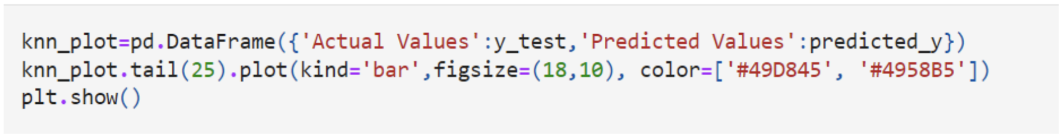
Figure 5: Accuracy of KNN model

2.3 Intelligence gained from this algorithm:

This algorithm predicts the value of the output variable and determines which class it belongs to by calculating the probabilities. From the graph, it can be inferred that the predicted price vs. the actual price has little difference and are close enough. For a type of dataset with a good amount of variance, the KNN model is a good choice for predicting the output. Thus we can infer that this model performs better than linear regression because linear regression tries to fit the model linearly while this model fits using the closest probability.

2.4 Results and Visualizations:

After we plot the graph for the KNN model, it can be seen that KNN models give good accuracy in predicting the weighted price. In the graph, the x-axis contains the timestamp, while the y-axis contains the price range. On a particular day, the difference between the predicted price the actual price is a little. This model proves to be a good fit for our data type and can be used for predicting the weighted price given the parameters Close, High, Low, Volume_BTC, and Volume_Currency.



```
knn_plot=pd.DataFrame({'Actual Values':y_test,'Predicted Values':predicted_y})
knn_plot.tail(25).plot(kind='bar',figsize=(18,10), color=['#49D845', '#4958B5'])
plt.show()
```

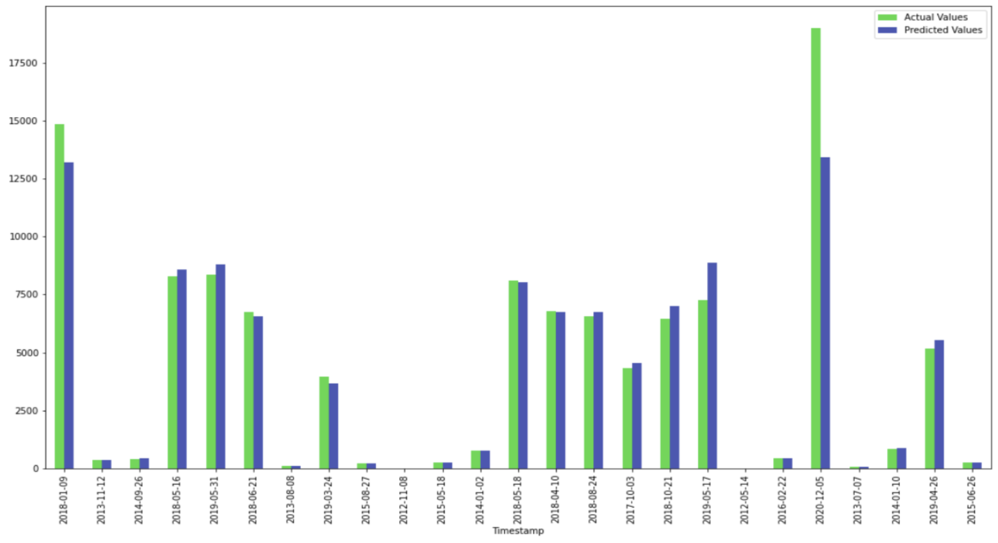


Figure 6: Actual v/s predicted values

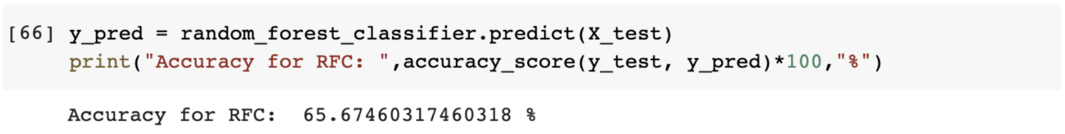
3 Random Forest Classification

3.1 Reason for choosing this algorithm:

The random forest classifier uses averaging to increase predictive accuracy and reduce overfitting. It does this by fitting several decision tree classifiers to different subsamples of the dataset. Since the randomness of this model strongly resonates with the cryptocurrency market, we think it's a good idea to use it to fit the time series. We have some predictions based on the SMA strategy introduced in phase 1 (golden and death cross). We divide train test data into an 85:15 ratio and use 'Trade_signal' calculated in phase 1 as true labels.

3.2 Effectiveness of this algorithm:

The effectiveness of this model is calculated in terms of the accuracy of the binary classification, which is shown as follows:



```
[66] y_pred = random_forest_classifier.predict(X_test)
     print("Accuracy for RFC: ",accuracy_score(y_test, y_pred)*100,"%")

     Accuracy for RFC:  65.67460317460318 %
```

Figure 7: Accuracy for random forest

Usually, a trading strategy is considered meaningful and would help investors realize profits if the accuracy of trading calls is greater than 60%. Using a random forest classifier, we can achieve around 64-65% accuracy using the golden/death cross identification and prediction of calls. This model can be considered moderately effective among all other models in the current market.

3.3 Intelligence gained from this algorithm:

This model acts as a decision support system for a trader daily. Since we output the trading signals, the investors will be aware of some highly volatile conditions beforehand and can avoid huge losses and, in inverse conditions, be highly profitable. As we can see in the two graphs shown below, the end part of the prediction signifies the BUY and hold option, and we can observe the price of bitcoin skyrocket. On the other hand, with the confusion matrix, we can realize that some trading signals are incorrect, and it is situational if the investor will have an overall profit or loss according to the volume of each trade. If the volume is the same across all trades, then we will be in profit after following this model.

3.4 Results and Visualizations:

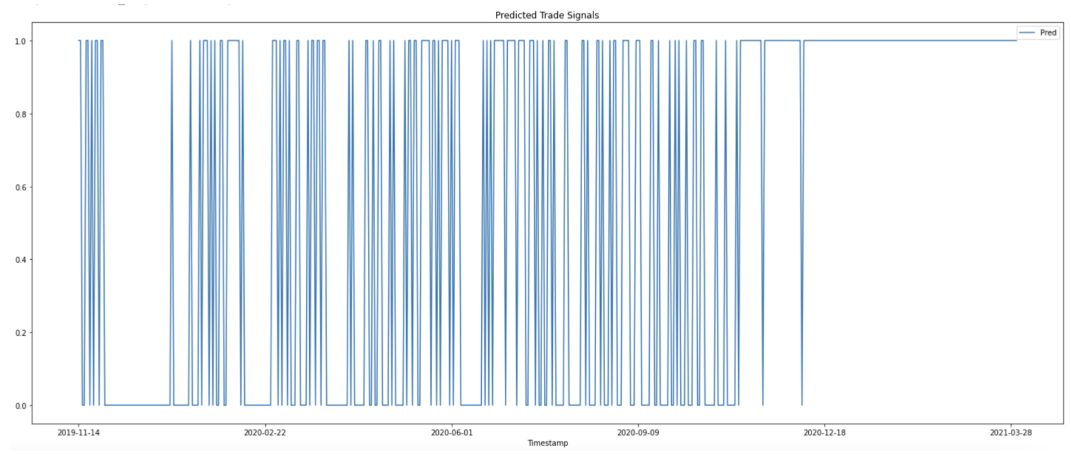


Figure 8: Output trade signals on test

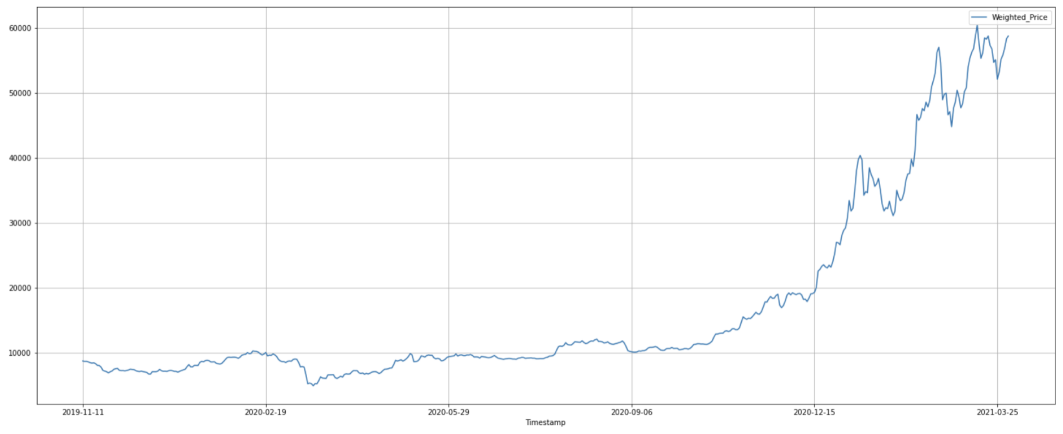


Figure 9: Testing data

In the 1st graph, the '1' trade signal signifies 'BUY,' and the '0' trade signal signifies 'SELL.' In the 2nd graph, we simply have the test data for the weighted price of bitcoin to compare the change in the market accordingly. We have thus generated a simple trading strategy using SMA10 (10-day simple moving average) indicator and random forest model.

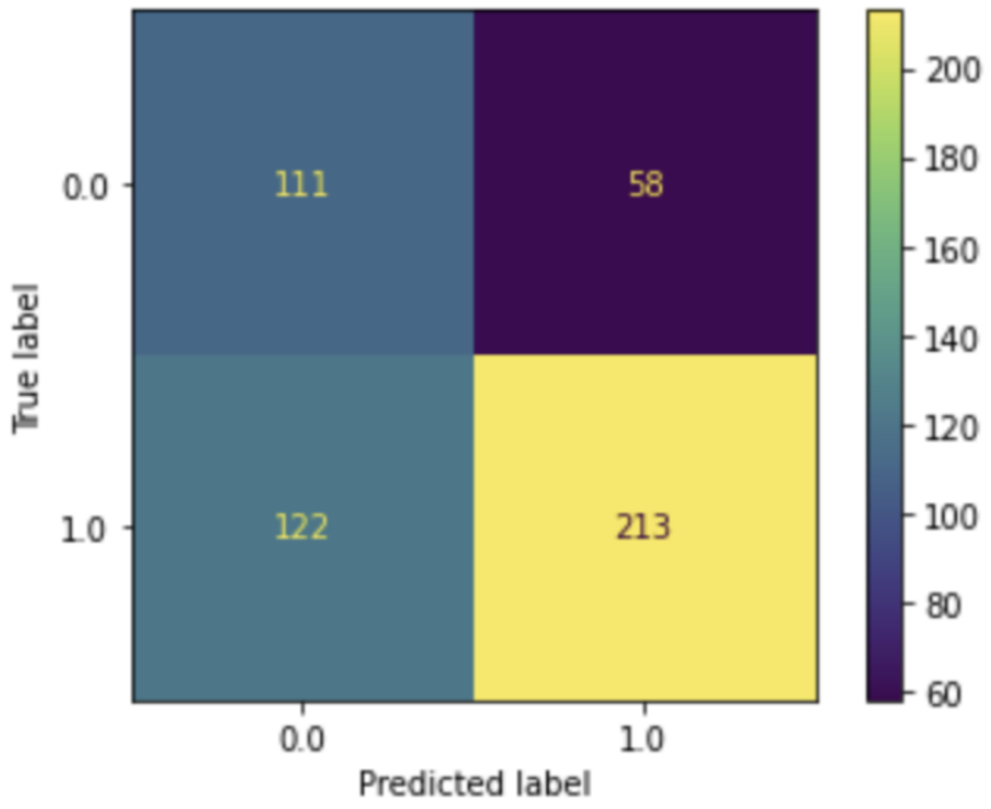


Figure 10: Confusion matrix

Above is the confusion matrix to realize the trade signal's true positives and negatives as well as false positives and negatives.

4 Long Short Term Memory

4.1 Reason for choosing this algorithm:

Long Short Term Memory(LSTM) has units of a recurrent neural network with three gates: input gate, output gate, and forget gate, along with a cell. These three gates control the flow of information into and out of the cell, and the cell remembers values across arbitrary time intervals. We can leverage this memorizing factor of LSTM to fit a model and predict bitcoin's price. Hence we chose this advanced algorithm for bitcoin's time series analysis.

4.2 Effectiveness of this algorithm:

LSTM has been observed to be very effective in fitting the bitcoin price for approximately nine years, which is our training data. The model is seen to perform accurately until the price of bitcoin shows some movement that was never seen before. This includes price manipulation by whales, market news, and happenings around the globe, including the COVID-19 pandemic and recession. For the training data, we calculated the model's effectiveness by plotting loss over each epoch and found a sudden decrease over initial epochs and stable loss 0.0017 while the model converges.

4.3 Intelligence gained from this algorithm:

Before discussing about the intelligence gained, let's have a look at the primary modulation of sequential LSTM used with the following parameters:
- Units: 128 (number of hidden units)
- Dense: 1 (output layer)
- Dropout: 0.2
- Activation function: sigmoid
- Optimizer: adam
- Loss: mean squared loss (MSE)
- Number of epochs: 100
- Batch size: 50

We can infer that LSTM cannot be effective in volatile situations as described above; hence, we see the predicted price lags more than the actual price in the last days of the test data frame. However, LSTM can be the best fit for a more stable stock prediction. Further intelligence gained is that the activation function, if changed to 'relu,' can overfit the model and give desired results for the current volatile data. This may be good for the cryptocurrency price but had a trade-off of overfitting and underperforming under normal market conditions.

4.4 Results and Visualizations:

Visualizing training and testing data over split ratio of 85:15

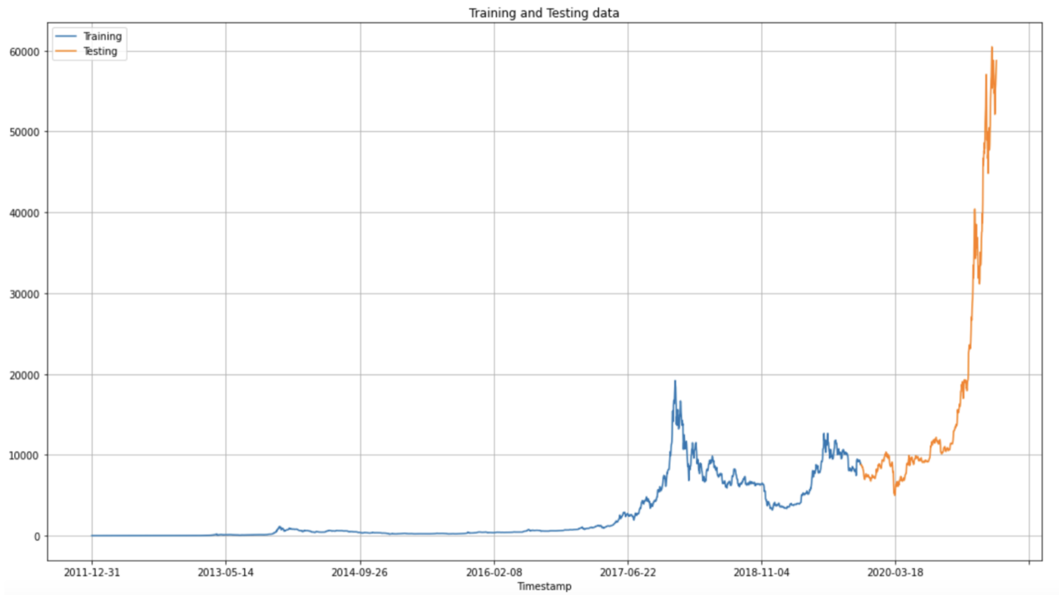
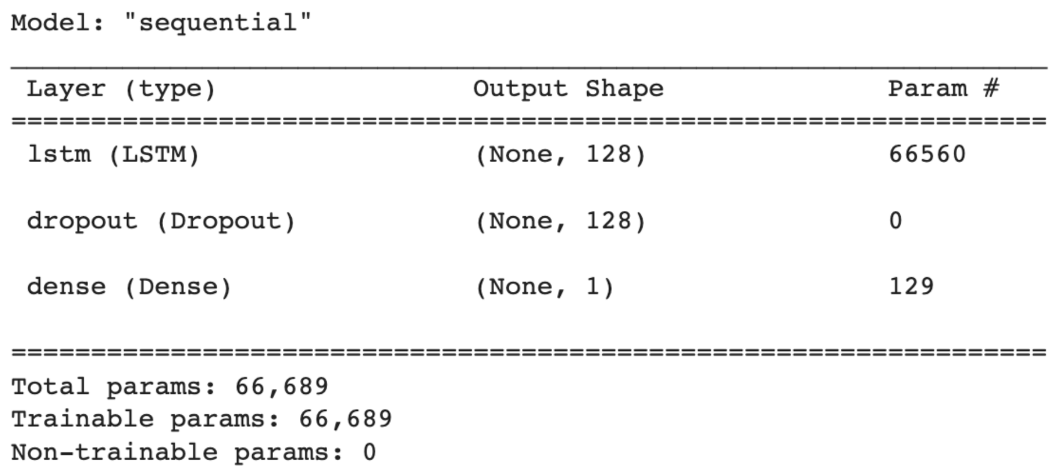


Figure 11: Training and testing data

Finding number of trainable parameters through model summary after setting all hyper parameters



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 128)               66560

 dropout (Dropout)           (None, 128)               0

 dense (Dense)               (None, 1)                 129

=================================================================
Total params: 66,689
Trainable params: 66,689
Non-trainable params: 0
```

Figure 12: Summary of model

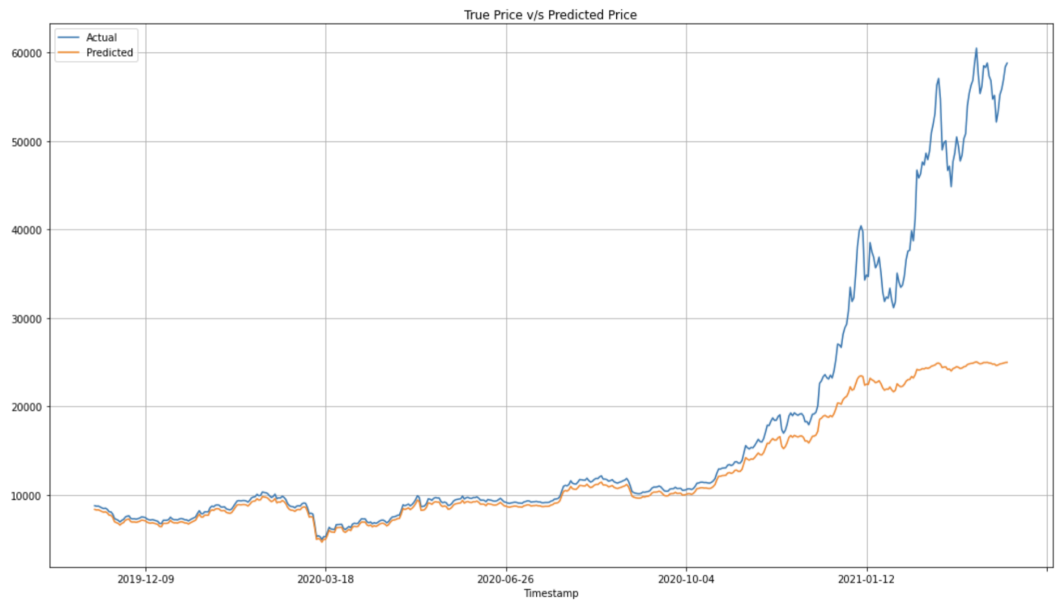


Figure 13: Visualization of actual vs predicted price on test data

As we can observe, the model performs really well on earlier phase of testing data but fails to keep up with rising BTC price due to uncertain circumstances in crypto market.

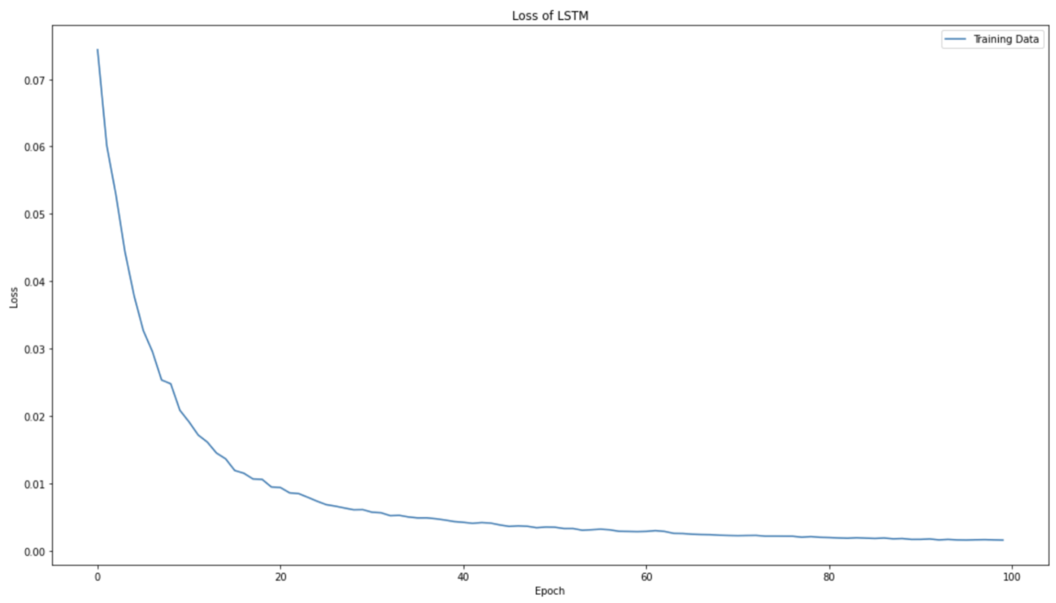


Figure 14: Loss over 100 epochs

The graph of decreasing loss is as expected for vanilla LSTM model.

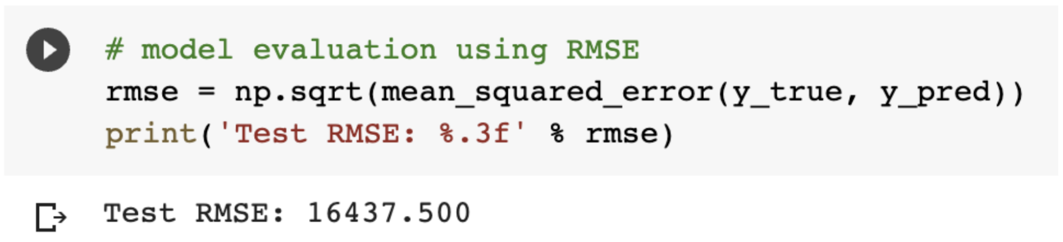
5 FB Prophet

5.1 Reason for choosing this algorithm:

The prophet library is an open-source library by Facebook(Meta) and is mainly used for automatic time series forecasts. The reason for choosing this model to fit the bitcoin data is its ability to predict seasonality and trend. Along with the price prediction, it also provides upper and lower bounds of the forecast that can give us a range of predictions, and we can cut our losses if the market price goes the other way.

5.2 Effectiveness of this algorithm:

Since the prophet model gives a forecast and region for the possibility of price data, we use RMSE (Root Mean Squared Error) for evaluating the model on testing data. It can range between 0 to infinity, and the lower, the better.



```
# model evaluation using RMSE
rmse = np.sqrt(mean_squared_error(y_true, y_pred))
print('Test RMSE: %.3f' % rmse)
```

```
Test RMSE: 16437.500
```

Figure 15: RMSE for prophet

Effectiveness is further estimated by the 'yhat,' 'yhat_lower', and 'yhat_higher' values generated in the forecasting data frame. The 'yhat' is the actual forecast value, and 'yhat_lower' and 'yhat_upper' are the uncertainty estimators interval for our prophet model.

5.3 Intelligence gained from this algorithm:

We used the same ratio for splitting training and testing data: 85:15 and 'Weighted_Price' for forecasting. Primary intelligence gained from applying the prophet model to the bitcoin dataset is that the slope direction of the upper bound estimate is true; however, this model did perform poorly when compared to the test data. The shaded region, as shown in the results below gives a trader an accurate estimate of their risks and further be extremely useful to evaluate our trading strategy and investments in terms of returns to risk ratio. One last thing we can conclude from the results is that the region of the bounds of time series is rigid and needs to be more flexible to react to changing trends in the market appropriately.

5.4 Results and Visualizations:

```
# Visualize model components
prophet_model.plot_components(forecasting)
```

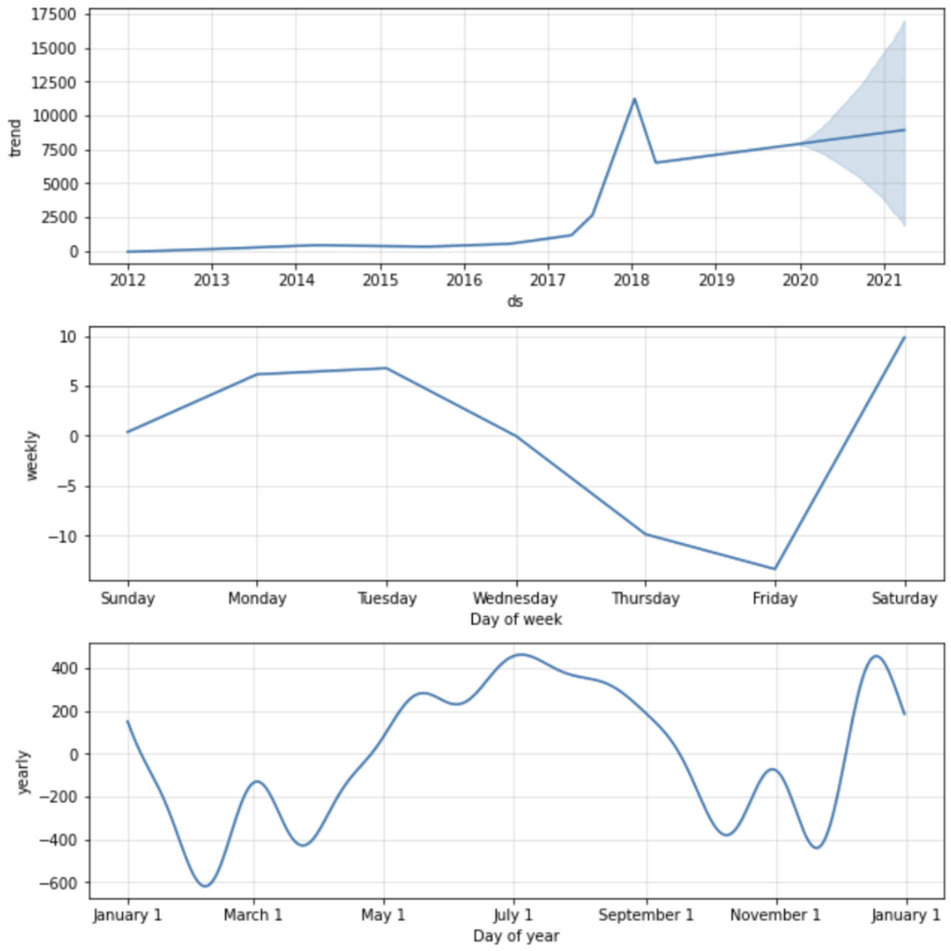


Figure 16: Prophet components

```
# Visualizing results
prophet_model.plot(forecasting)
plt.title('Price Forecasting for Bitcoin')
plt.ylabel('Price (USD)')
plt.xlabel('Date')
plt.show()
```

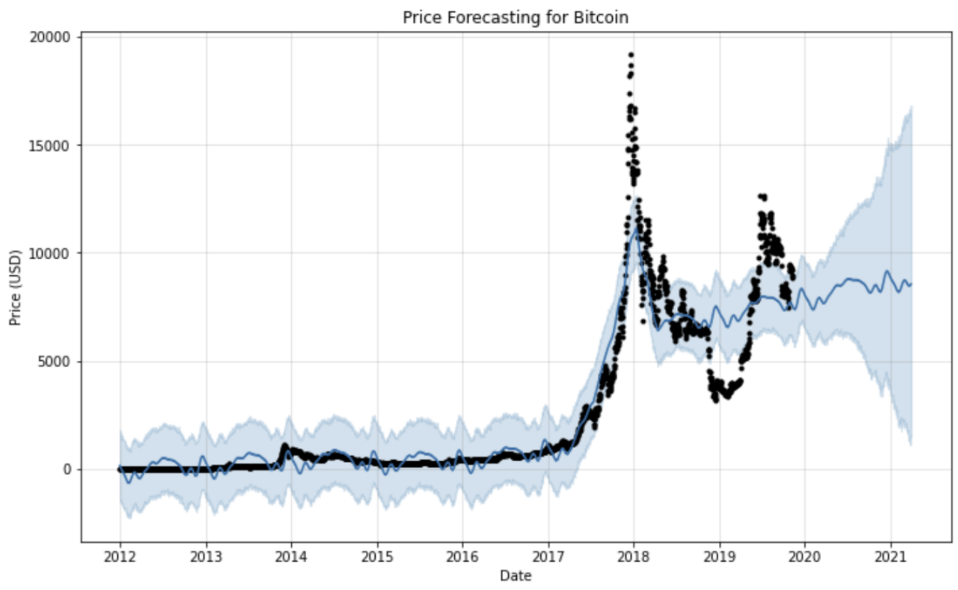


Figure 17: Forecast using prophet

As we can see, the shaded region shows the upper and lower bounds of the forecasted bitcoin price, and the solid blue line shows the predicted value. We can also observe from the region that bitcoin has a higher chance of an upper trend, and for true values in our test data, we can see the actual price of bitcoin follows the upper trend.

6 REFERENCES

1. https://keras.io/api/layers/recurrent_layers/lstm/
2. https://medium.com/analytics-vidhya/how-well-can-machine-learning-models-predict-the-price-of-bitcoin-f036fdecdc03
3. Stock Trading With Random Forests, Trend Detection Tests and Force Index Volume Indicators
4. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.plot_confusion_matrix.html
5. https://towardsai.net/p/machine-learning/netflix-stock-prediction-model-a-comparative-study-of-linear-regression-k-nearest-neighbor-knn-4527ff17939b