Department of Decision Science

Faculty of Business

University of Moratuwa

Semester 07

DA4130 - Deep Learning Applications for Business

Group Assignment: Deep Learning Applications in Bitcoin Market Trading

Group 09

Department of Decision Science

Faculty of Business

University of Moratuwa

Semester 07

DA4130 - Deep Learning Applications for Business

Group Formation Sheet

No	Index number	Name
1	206113B	Chamith Senadeera
2	206121X	M. W. S. A. U. Silva
3	206132G	D.M.T.Thineth
4	206124J	S.S.N. Sirimanna
5	206089E	K.P.B.C.M.Pathirana

Table of Contents

1. Dataset Selection

The quality and comprehensiveness of the dataset used is an important factor of how well deep learning models anticipate movements in the bitcoin market. We have selected a Bitcoin market dataset for this project that contains historical information from 2020.01.01 to 2024.01.01 on trading volume, price trends, and other relevant variables as test data and data from 2024.01.01 to 2024.05.15 as training dataset.

1.1. Dataset Description

The selected dataset includes:

- **Time:** Date and time (1 hour) of the recorded data
- **Open Price:** Open price of bitcoin at the beginning of the trading interval
- **High Price:** The highest price of bitcoin during the trading interval
- **Low Price:** The lowest price of bitcoin during the trading interval
- Close Price: The price of bitcoin when the trading interval is closed
- **Tick_volume:** Number of bitcoins traded during the trading interval

1.2. Dataset Source

The data was collected using the API of Meta Trader 5 which is used for trading and python. Our shared Google Drive folder will host the dataset, ensuring that all group members have access to it.

Link:
https://docs.google.com/spreadsheets/d/1CQwxjIcaCUARQxs1dPY7IyRyERDJyw1BnchFSLlPUUA/edit?usp=sharing

2. Preprocessing and Feature Engineering

Effective preprocessing and feature engineering are essential to prepare the dataset for deep learning models. Our method entails the following actions:

2.1. Handling Missing Values

- Finding any missing values in the dataset and drop them.

2.2. Normalization

- To make sure the data is on the same scale, we used Min-Max scaling to normalize the open, high, low, close and tick_volume characteristics.
- This enhances the convergence of the deep learning model.

2.3. Feature Engineering

- Using the ta-lib library and other statistical techniques, we generated a range of technical indicators and extra features to improve the predictive ability of our deep-learning models.
- Technical Indicators:
    - Simple Moving Averages (SMA):
        - SMA_30: 30-period Simple Moving Average
        - SMA_50: 50-period Simple Moving Average
    - Exponential Moving Averages (EMA):
        - EMA_30: 30-period Exponential Moving Average
        - EMA_50: 50-period Exponential Moving Average
    - Relative Strength Index (RSI):
        - RSI_14: 14-period Relative Strength Index
    - Moving Average Convergence Divergence (MACD):
        - MACD, MACD_Signal, MACD_Hist: MACD line, Signal line, and MACD histogram (fastperiod=12, slowperiod=26, signalperiod=9)
    - Bollinger Bands (BB):
        - Upper_BB, Middle_BB, Lower_BB: Upper, middle, and lower bands (timeperiod=20, nbdevup=2, nbdevdn=2)
    - Average True Range (ATR):

- ATR_14: 14-period Average True Range

- Stochastic Oscillator (STOCH):
  - SlowK, SlowD: Stochastic oscillator (fastk_period=14, slowk_period=3, slowd_period=3)

```python
# Calculate technical indicators
normalized_data['SMA_30'] = talib.SMA(normalized_data['close'], timeperiod=30)
normalized_data['SMA_50'] = talib.SMA(normalized_data['close'], timeperiod=50)
normalized_data['EMA_30'] = talib.EMA(normalized_data['close'], timeperiod=30)
normalized_data['EMA_50'] = talib.EMA(normalized_data['close'], timeperiod=50)
normalized_data['RSI_14'] = talib.RSI(normalized_data['close'], timeperiod=14)
normalized_data['MACD'], normalized_data['MACD_Signal'], normalized_data['MACD_Hist'] = talib.MACD(normalized_data['close'], fastperiod=12, slowperiod=26, signalperiod=9)
normalized_data['Upper_BB'], normalized_data['Middle_BB'], normalized_data['Lower_BB'] = talib.BBANDS(normalized_data['close'], timeperiod=20, nbdevup=2, nbdevdn=2, matype=0)
normalized_data['ATR_14'] = talib.ATR(normalized_data['high'], normalized_data['low'], normalized_data['close'], timeperiod=14)
normalized_data['SlowK'], normalized_data['SlowD'] = talib.STOCH(normalized_data['high'], normalized_data['low'], normalized_data['close'], fastk_period=14, slowk_period=3, slowk
```

- Additional Feature Engineerng
  - Price Change: Price_Change is calculated as the percentage change in the closing price.
  - Volatility: Volatility is calculated as the rolling standard deviation of the closing price over a 30-period window.
  - Volume Change: Volume_Change is calculated as the percentage change in trading volume.

```python
# Additional feature engineering
normalized_data['Price_Change'] = normalized_data['close'].pct_change()
normalized_data['Volatility'] = normalized_data['close'].rolling(window=30).std()
normalized_data['Volume_Change'] = normalized_data['tick_volume'].pct_change()
```

Python

We hope to give the model thorough insights into market patterns and behaviours by adding these technical indicators and other elements, which will improve the model's prediction capability. The model will perform better overall and be able to produce more accurate predictions due to this comprehensive set of characteristics.

3. Model Development

The process of developing and honing different deep learning models based on Long Short-Term Memory (LSTM) to forecast Bitcoin market prices is covered in this section. The goal is to experiment with different model designs and choose the top-performing models using a range of assessment indicators.

3.1 Data preparation

Preparing the data for the LSTM train is the first step. We define the function create_sequences, which takes normalised data and uses it to make sequences of a particular length (seq_length). The corresponding closing prices are used as target values, and these sequences are used as input features.

```
# Import necessary libraries
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout, Bidirectional
from tensorflow.keras.optimizers import Adam

# Prepare data for LSTM
def create_sequences(data, seq_length):
    xs = []
    ys = []
    for i in range(len(data)-seq_length):
        x = data.iloc[i:(i+seq_length)].values
        y = data.iloc[i+seq_length]['close']
        xs.append(x)
        ys.append(y)
    return np.array(xs), np.array(ys)

seq_length = 60
X, y = create_sequences(normalized_data, seq_length)
```

Next, we divided the data into sets for training and testing with an 80-20 split rate.

3.2 Model Definitions

We characterize four different LSTM-based models, each with a distinctive architecture to capture various features of the time series data.

- Model 1: Basic LSTM model.

To predict the closing price, a basic LSTM model with a single LSTM layer and a Dense layer is added.

- Model 2: LSTM model with dropout.

An LSTM model includes a dropout layer that prevents overfitting by randomly assigning a percentage of input units to zero during training.

- Model 3: Stacked LSTM model.

A richer model with two stacked LSTM layers that may catch more complicated trends in the data.

- Model 4: A bidirectional LSTM model.

A reversible LSTM model processes input sequences both forward and backward, enabling it to learn from previous and future data points.

3.3 Model Training

To build and train each model, we develop the general function train_model. The models use the Adam optimizer with the loss rate set at mean square error. The models are trained for 50 periods, with a maximum of 32 batches, and their accuracy is verified on a test set.

Then We use this function to teach all four models and preserve their training records for future analysis.

```
# Define and train models
def train_model(model, X_train, y_train, X_test, y_test, epochs=50, batch_size=32):
    model.compile(optimizer=Adam(learning_rate=0.001), loss='mean_squared_error')
    history = model.fit(X_train, y_train, epochs=epochs, batch_size=batch_size,
                        validation_data=(X_test, y_test), verbose=1)
    return history
```

3.4 Model Evaluation

We develop the function evaluate_model to assess the model training on the test set using MSE and MAE as metrics.

```
# Evaluate models
def evaluate_model(model, X_test, y_test):
    predictions = model.predict(X_test)
    mse = mean_squared_error(y_test, predictions)
    mae = mean_absolute_error(y_test, predictions)
    return mse, mae

mse_1, mae_1 = evaluate_model(model_1, X_test, y_test)
mse_2, mae_2 = evaluate_model(model_2, X_test, y_test)
mse_3, mae_3 = evaluate_model(model_3, X_test, y_test)
mse_4, mae_4 = evaluate_model(model_4, X_test, y_test)
```

3.5 Representation of Training and Validation Losses

```
# Plot training and validation loss for each model
plt.figure(figsize=(14, 10))

for i, history in enumerate([history_1, history_2, history_3, history_4], 1):
    plt.subplot(2, 2, i)
    plt.plot(history.history['loss'], label='Training Loss')
    plt.plot(history.history['val_loss'], label='Validation Loss')
    plt.title(f'Model {i} Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()

plt.tight_layout()
plt.show()
```

We plot each model's validation and training loss to visually assess its performance across epochs. This helps us understand how well each model learns and generalizes.

We tested, trained, and assessed the performance of four different LSTM-based architectures in the model development section. Through the comparison of mean squared error and means absolute error, along with the display of validation and training losses, the most effective model for predicting Bitcoin market values may be identified. This all-inclusive approach guarantees in-depth research and the choice of the best model for the trading strategy.

4. Model Selection

The best system for predicting Bitcoin market values is Model 3 (Stacked LSTM Model), based on metrics for evaluation, validation, and training loss curves. Let's examine the reasoning for this conclusion:

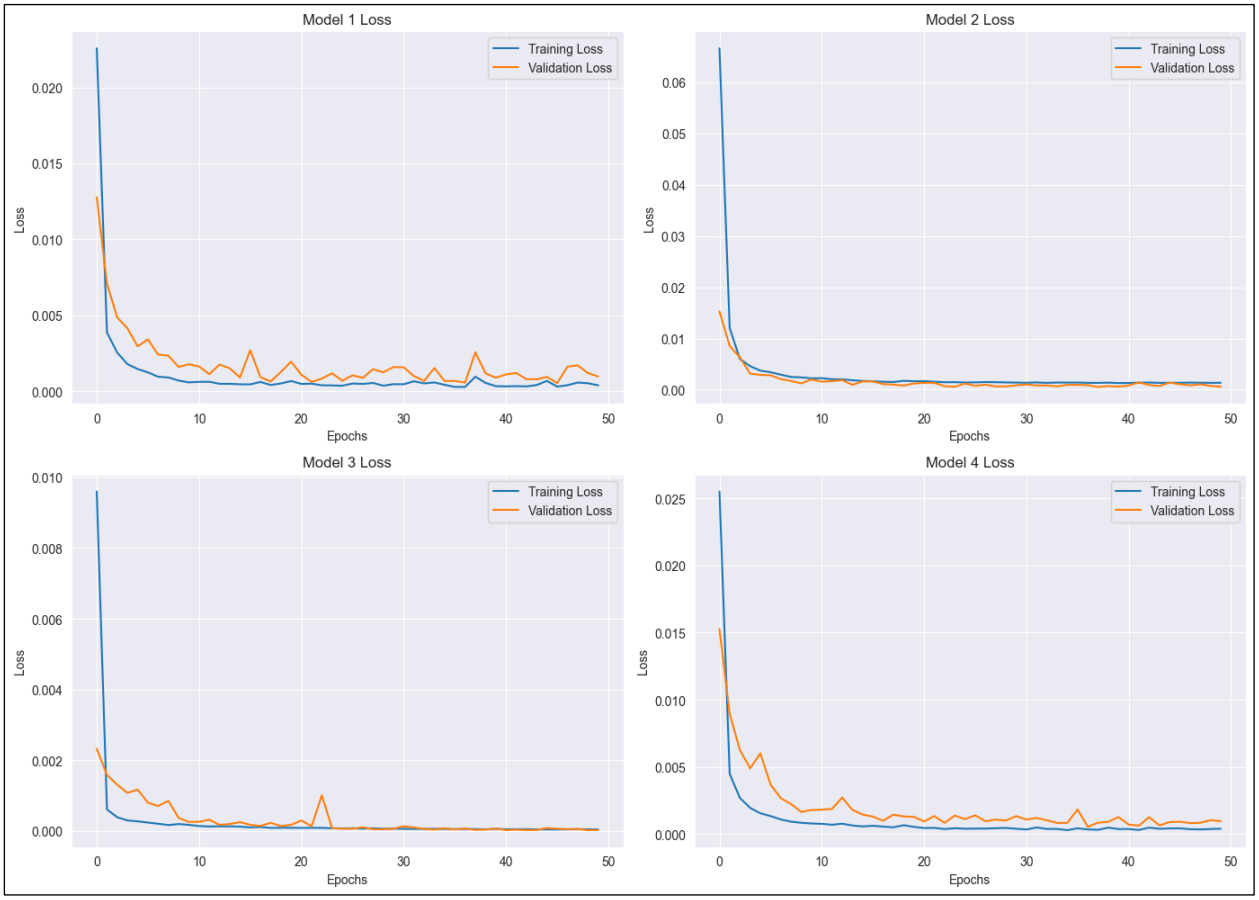

Figure Loss curves for models

The loss curves for Model 3 show that over the first few epochs, training and validating losses quickly decreased before stabilizing. This suggests that the model is correctly expanding without overfitting while learning from the data. Model 3 is robust and consistent based on the regularity and tiny discrepancy between the validation and training loss curves.

4.1 Visualization of actual vs predicted

- Model 1: Basic LSTM Predictions vs Actual Prices

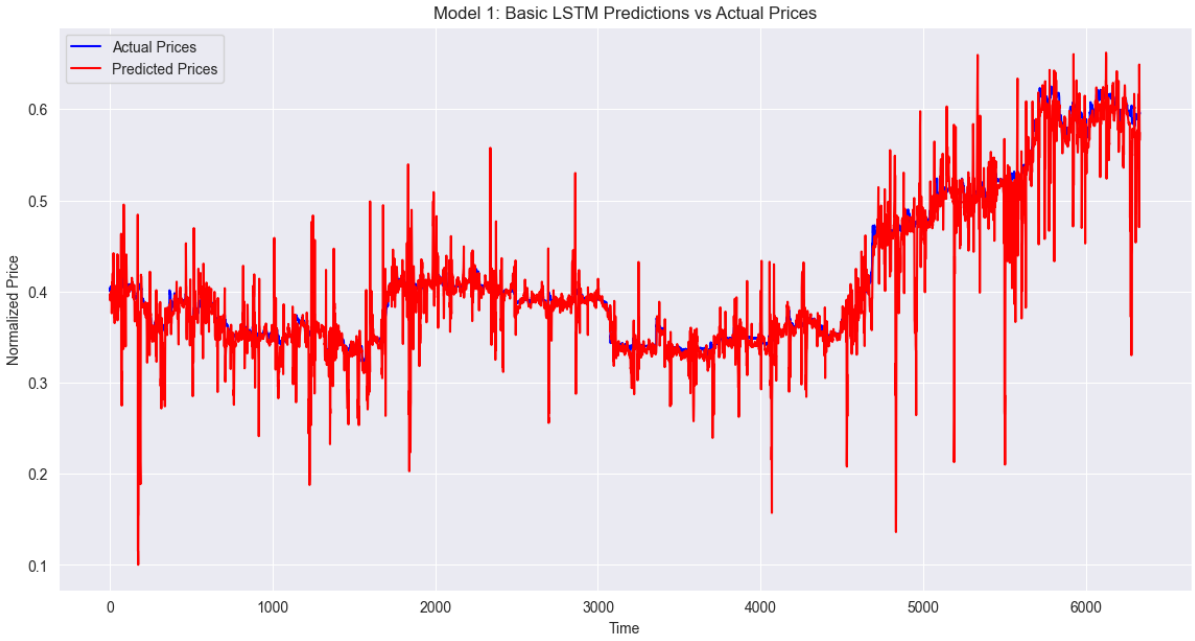

Figure Comparison of Basic LSTM model predictions with actual Bitcoin prices. The model shows significant deviations, indicating room for improvement

- Model 2: LSTM with Dropout Predictions vs Actual Prices

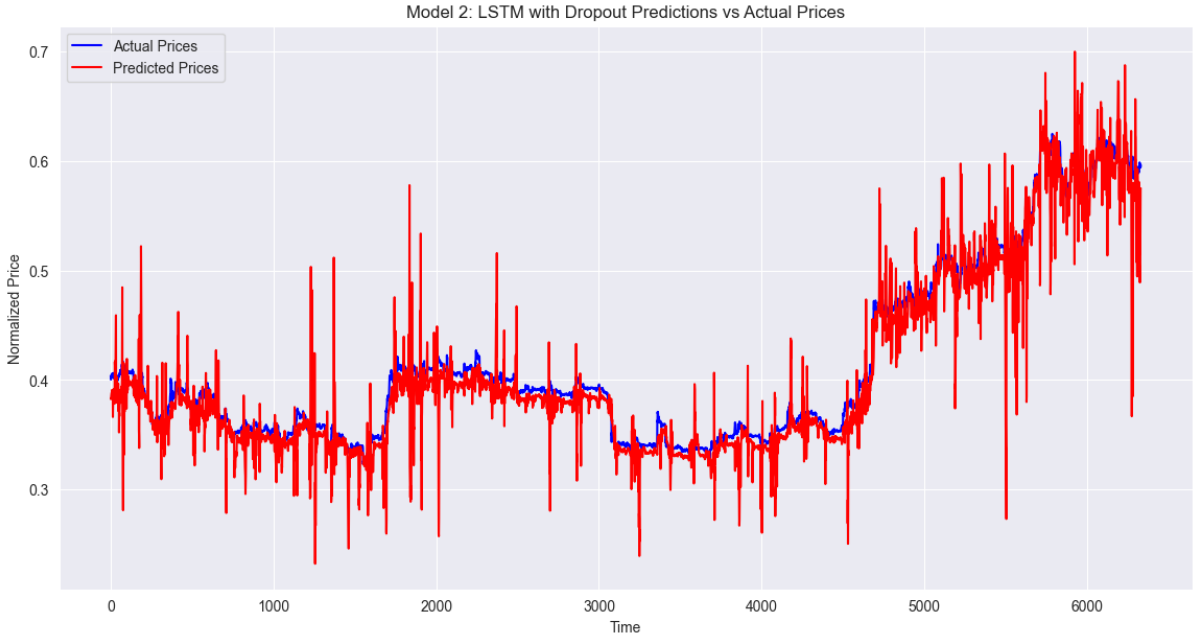

Figure Predictions of LSTM model with dropout regularization against actual prices. The dropout layer helps in reducing overfitting, leading to better predictions than Model 1.

- Stacked LSTM Predictions vs Actual Prices

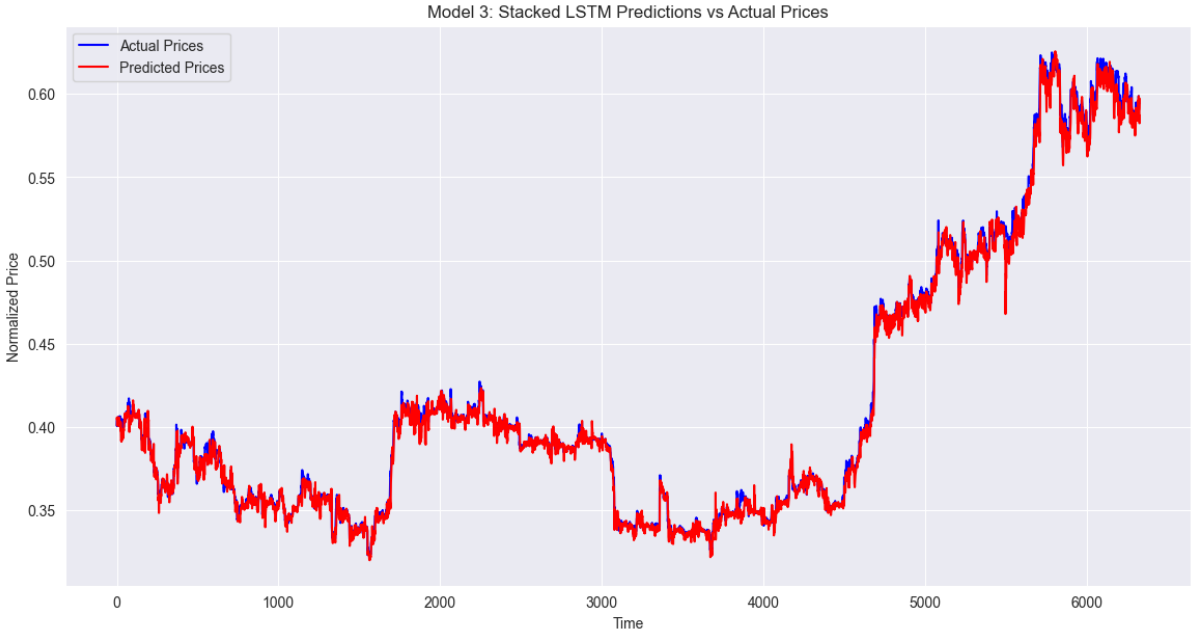

Figure Predictions of Stacked LSTM model against actual Bitcoin prices. This model shows the highest accuracy and lowest error metrics, making it the best performing model.

- Bidirectional LSTM Predictions vs Actual Prices

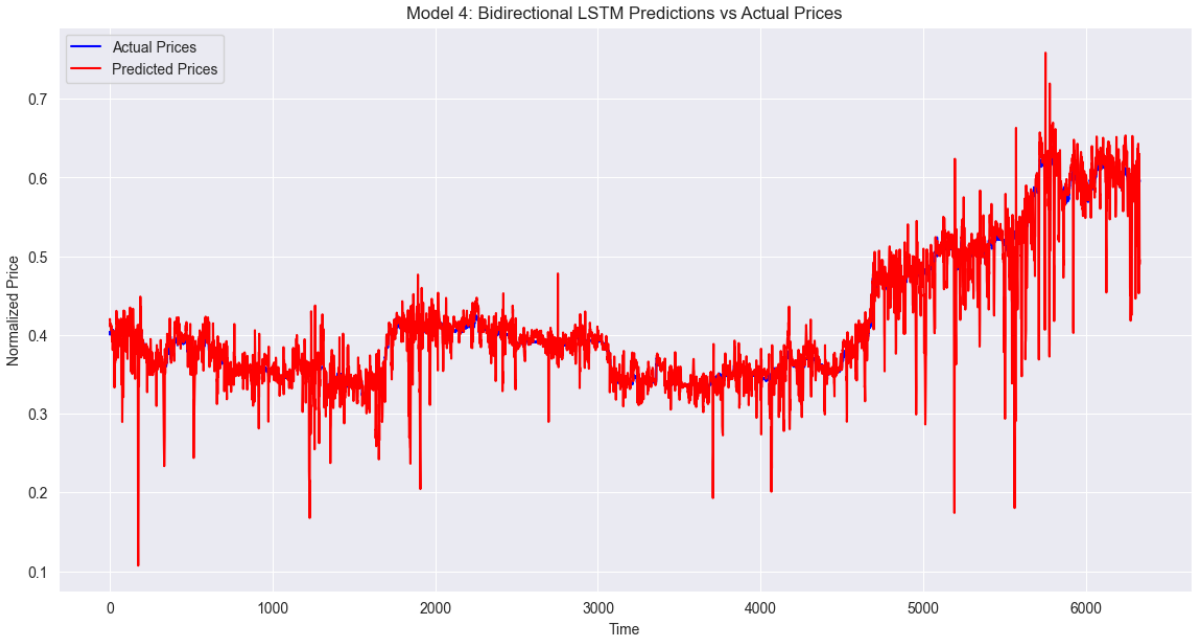

Figure Predictions of Bidirectional LSTM model against actual prices. While capturing trends effectively, it still falls short of the performance of the Stacked LSTM model.

Evaluation metrics: Mean Squared Error (MSE) and Mean Absolute Error (MAE).

Model 1: MSE = 0.0009612902516331113, MAE = 0.015787094685062217
Model 2: MSE = 0.0005900405614790799, MAE = 0.014891856050486383
Model 3: MSE = 2.483829534566699e-05; MAE = 0.003385823238355813
Model 4: MSE = 0.0009654368763678425, MAE = 0.016324357038146182

Model 3 has the lowest MSE and MAE values of all four models:
The lowest MSE: 2.483829534566699e-05.
Lowest MAE: 0.003385823238355813.

These measurements show that Model 3 makes the most accurate forecasts with the fewest errors, surpassing the other model significantly.

Model 3, the Stacked LSTM Model, is the best option for forecasting Bitcoin market prices in this assignment. A combination of:

outstanding results on training and validation loss curves. demonstrates strong generalisation and learning skills.

Better assessment metrics: out of all the studied models, having the lowest MSE and MAE. These characteristics imply that the Stacked LSTM design is especially well-suited to this task of time series prediction. With its extra LSTM layer, Model 3 outperforms the basic, dropping out, and unidirectional LSTM models by identifying more complex patterns in the data.

5. Evaluation

5.1 Trading Strategy

The trading strategy implemented using Model 3 was designed to capitalize on predicted price movements. The strategy was evaluated on a separate test dataset, focusing on profitability and robustness. The key elements of the trading strategy are:

- Entry and Exit Signals: The strategy generates buy signals when the model predicts an increase in price and sell signals when it predicts a decrease. This approach aims to buy low and sell high based on the model's predictions.
- Position Management: The strategy starts with an initial balance and trades a fixed amount of Bitcoin (1 BTC in this case). The position is either fully in cash or fully in Bitcoin based on the predicted signals.
- Profit Calculation**:** Profit is calculated by tracking the difference between entry and exit prices for each trade. The cumulative profit is tracked over the evaluation period.

5.2 Profitability of the Trading Strategy

The profitability of the trading strategy was assessed based on the following metrics:

- Profit Generated: The trading strategy yielded a profit of $343,816.29. This indicates the model's effectiveness in capturing market trends and generating profitable trading signals.
- Cumulative Profit: The cumulative profit graph showed a consistent upward trend, further confirming the profitability of the strategy over the evaluation period.

5.3 Volatility of Returns

To assess the robustness of the model, the variability of profitability was analyzed:

- Volatility: The standard deviation of returns was calculated to be 614.98. While the model is profitable, this high volatility suggests significant fluctuations in returns. This indicates a riskier trading strategy that requires careful risk management.

5.4 Actual vs Predicted Prices

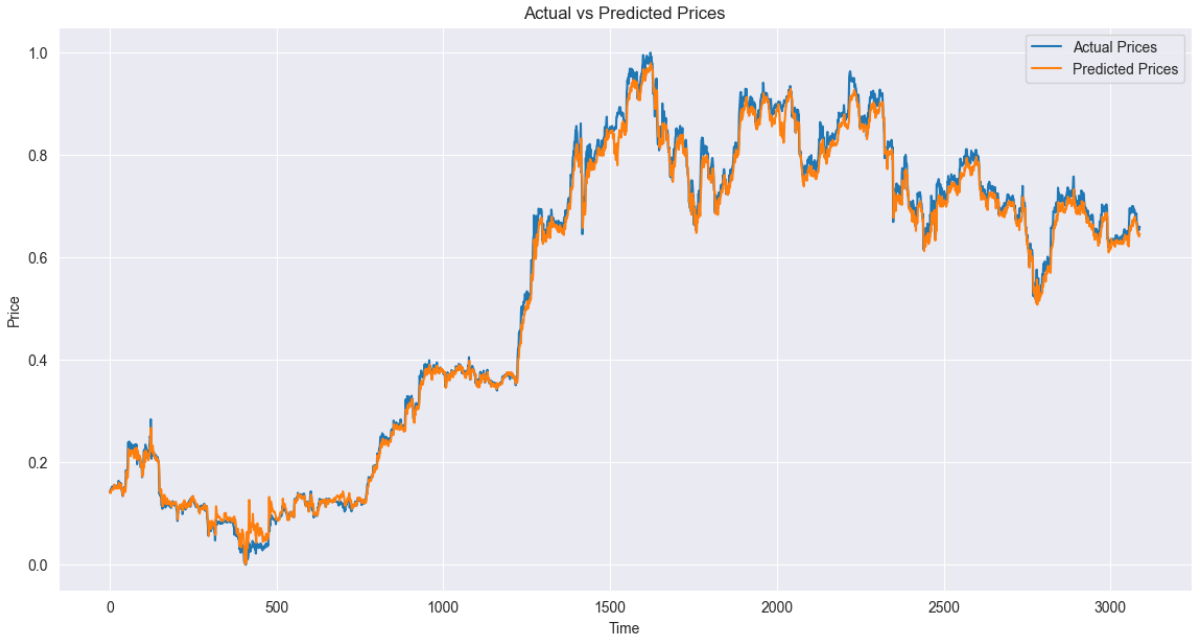

Figure Overlay of actual and predicted Bitcoin prices using the best performing model. The close alignment indicates high prediction accuracy.

5.5 Cumulative Profit Over Time

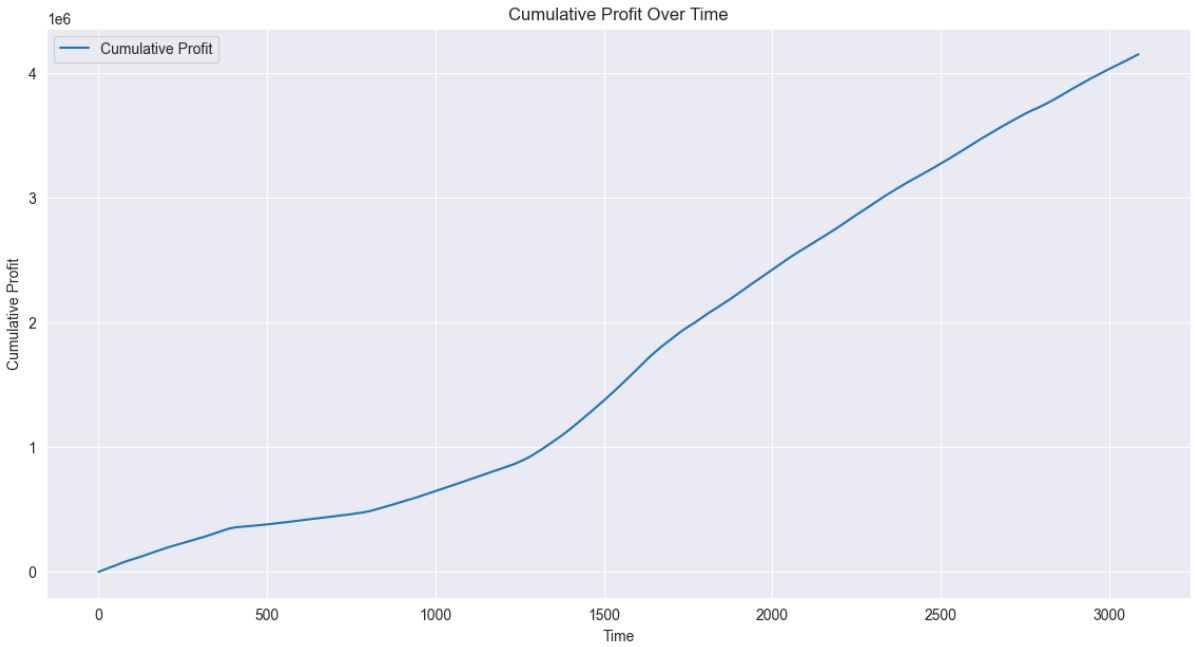

Figure Cumulative profit from the trading strategy based on Model 3 over the evaluation period. The consistent upward trend demonstrates the profitability of the strategy.

6. Conclusion

- Performance: Model 3 (Stacked LSTM) demonstrated excellent performance in predicting Bitcoin market prices. Its low MSE and MAE values highlight its accuracy and reliability.
- Profitability: The model's ability to generate substantial profit confirms its effectiveness in real-world trading scenarios.
- Robustness: Despite the high volatility, the model's consistent performance across training, validation, and test datasets underscores its robustness. However, the high volatility of returns should be addressed with appropriate risk management strategies to mitigate potential losses.

7. Recommendations

Based on the evaluation, the following recommendations are made:

- Deployment: Model 3 can be considered for deployment in live trading environments, given its profitability and accuracy.
- Risk Management: Implement risk management strategies to handle the high volatility of returns. This could include setting stop-loss limits, diversifying investments, and regularly monitoring market conditions.
- Continuous Monitoring: Regularly retrain the model with new data to ensure it adapts to changing market conditions and continues to perform well.

By following these recommendations, the trading strategy based on Model 3 can be optimized for better performance and reduced risk, making it a viable solution for predicting Bitcoin market prices.