

Bitcoin Price Prediction Using Machine Learning

Anoop Lashiyal (002849957)

Pranavi Yarlagaadda (002848553)

1. Introduction

1.1 Problem Statement

Predicting Bitcoin prices is a challenging task due to its highly volatile and non-linear nature. Accurately forecasting price trends can help investors, traders, and stakeholders make informed decisions.

1.2 Objectives

- Develop a machine learning model to predict Bitcoin price trends and values.
- Compare traditional machine learning models and deep learning techniques to identify the best-performing approach.

1.3 Motivation

Cryptocurrencies like Bitcoin have gained immense popularity as an alternative investment. However, price fluctuations make them highly unpredictable. This project explores advanced techniques to forecast Bitcoin prices by leveraging historical price data and time-series modeling.

2. Data Collection and Preprocessing

2.1 Dataset

- **Source:** We downloaded the data from kaggle.
- **Features:**
 - Timestamp
 - Open Price
 - Close Price
 - High Price
 - Low Price

- Volume

2.2 Data Preprocessing

- **Scaling:** Min-MaxScaler was applied to normalize the data between 0 and 1.
- **Feature Selection:** Selected the closing price as the primary target for predictions.
- **Sequence Creation:** For LSTM, data was segmented into sequences of 30-day windows as inputs.


2.3 Exploratory Data Analysis

- Display the first few rows of a DataFrame



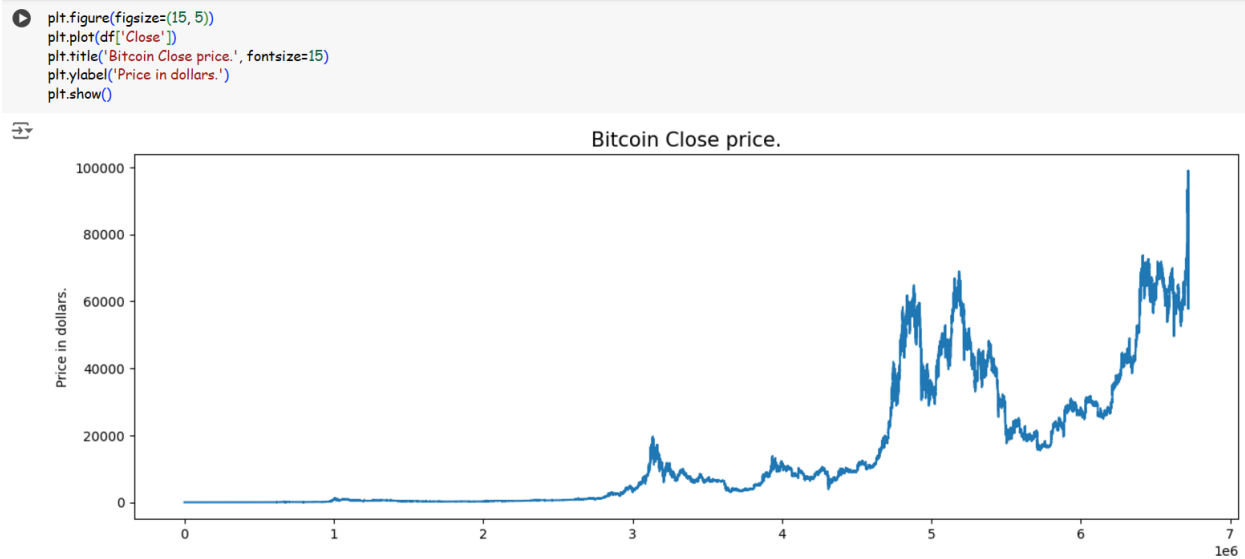
	Timestamp	Open	High	Low	Close	Volume
0	1.325412e+09	4.58	4.58	4.58	4.58	0.0
1	1.325412e+09	4.58	4.58	4.58	4.58	0.0
2	1.325412e+09	4.58	4.58	4.58	4.58	0.0
3	1.325412e+09	4.58	4.58	4.58	4.58	0.0
4	1.325412e+09	4.58	4.58	4.58	4.58	0.0

- Data Description



	Timestamp	Open	High	Low	Close	Volume
count	6.719280e+06	6.719281e+06	6.719281e+06	6.719281e+06	6.719281e+06	6.719281e+06
mean	1.527023e+09	1.433533e+04	1.434155e+04	1.432893e+04	1.433536e+04	5.474763e+00
std	1.164142e+08	1.924336e+04	1.925088e+04	1.923573e+04	1.924346e+04	2.296880e+01
min	1.325412e+09	3.800000e+00	3.800000e+00	3.800000e+00	3.800000e+00	0.000000e+00
25%	1.426212e+09	4.118900e+02	4.120000e+02	4.117000e+02	4.119000e+02	1.825500e-02
50%	1.527012e+09	6.221860e+03	6.225000e+03	6.218260e+03	6.221630e+03	5.000000e-01
75%	1.627812e+09	2.298800e+04	2.299400e+04	2.298200e+04	2.298800e+04	3.189348e+00
max	1.732579e+09	9.899600e+04	9.912100e+04	9.896400e+04	9.899300e+04	5.853852e+03

- Close price graph plot



- Checking if null values are present and removing the null values

```
# checking if we have any null values in the dataset
df.isnull().sum()
```

```
Timestamp    0
Open         0
High         0
Low          0
Close        0
Volume       0

dtype: int64
```

✓
0s



```
# Removing the row that has null value  
df = df.dropna(subset=['Timestamp'])
```

✓
0s

```
[10] # checking if the row is removed  
df.isnull().sum()
```



0

Timestamp	0
Open	0
High	0
Low	0
Close	0
Volume	0

dtype: int64

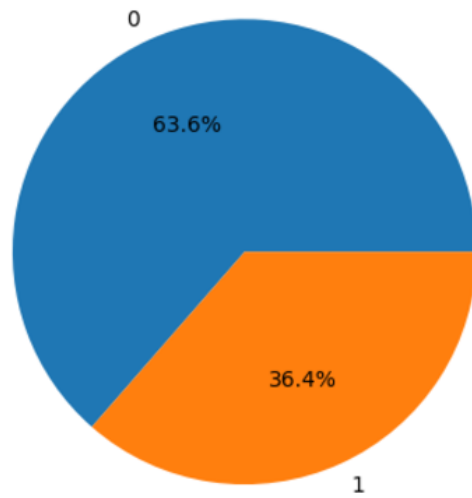
2.4 Feature Engineering

- Feature engineering involves creating new, meaningful features from existing data. These additional features can significantly enhance the model's performance and provide deeper insights into the dataset.
- We added open-close, low-high, and target features.
- The heatmap indicates that the features are not highly correlated.

Adding the target feature which is a signal whether to buy or not we will train our model to predict this.

```
[14] df['open-close'] = df['Open'] - df['Close']  
      df['low-high'] = df['Low'] - df['High']  
      df['target'] = np.where(df['Close'].shift(-1) > df['Close'], 1, 0)
```

```
plt.pie(df['target'].value_counts().values,  
        labels=[0, 1], autopct='%1.1f%%')  
plt.show()
```



- The heatmap of the following features is as follows

Timestamp	1	0	0	0	0	0	0	0	0
Open	0	1	1	1	1	0	0	0	0
High	0	1	1	1	1	0	0	0	0
Low	0	1	1	1	1	0	0	0	0
Close	0	1	1	1	1	0	0	0	0
Volume	0	0	0	0	0	1	0	0	0
open-close	0	0	0	0	0	0	1	0	0
low-high	0	0	0	0	0	0	0	1	0
target	0	0	0	0	0	0	0	0	1
	Timestamp	Open	High	Low	Close	Volume	open-close	low-high	target

3. Methodology

3.1 Machine Learning Models Tested

1. Logistic Regression:

- Objective: Predict binary price movements (up or down).
- Performance: Accuracy ~ 52%.

2. Support Vector Classifier (SVC):

- Objective: Improve accuracy with non-linear kernels.
- Performance: Accuracy ~ 48%.

3. XGBoost:

- Objective: Use boosting for price trend prediction.
- Performance: Accuracy ~ 92%.
- But it had the issue of overfitting.

```
models = [LogisticRegression(), SVC(kernel='poly', probability=True), XGBClassifier()]

for i in range(3):
    models[i].fit(X_train, Y_train)

print(f'{models[i]} : ')
print('Training Accuracy : ', metrics.roc_auc_score(Y_train, models[i].predict_proba(X_train)[:,:1]))
print('Validation Accuracy : ', metrics.roc_auc_score(Y_valid, models[i].predict_proba(X_valid)[:,:1]))
print()
```

```
LogisticRegression() :
Training Accuracy : 0.5272712493564907
Validation Accuracy : 0.5187429004165088

SVC(kernel='poly', probability=True) :
Training Accuracy : 0.4828745224483161
Validation Accuracy : 0.5278844593498134

XGBClassifier(base_score=None, booster=None, callbacks=None,
               colsample_bylevel=None, colsample_bynode=None,
               colsample_bytree=None, device=None, early_stopping_rounds=None,
               enable_categorical=False, eval_metric=None, feature_types=None,
               gamma=None, grow_policy=None, importance_type=None,
               interaction_constraints=None, learning_rate=None, max_bin=None,
               max_cat_threshold=None, max_cat_to_onehot=None,
               max_delta_step=None, max_depth=None, max_leaves=None,
               min_child_weight=None, missing=nan, monotone_constraints=None,
               multi_strategy=None, n_estimators=None, n_jobs=None,
               num_parallel_tree=None, random_state=None, ...):
Training Accuracy : 0.9229563497439509
Validation Accuracy : 0.46156758803483533
```

3.2 Transition to Deep Learning

- Recognizing the limitations of traditional models in handling sequential data, **Long Short-Term Memory (LSTM)** networks were chosen for their ability to model temporal dependencies.

4. Implementation

4.1 LSTM Architecture

- **Input Layer:** 30-day sequences of scaled price data.
- **Hidden Layers:**
 - 2 LSTM layers with **units = 50 and 100**.
 - Dropout layers to prevent overfitting.
- **Output Layer:** Single neuron for price prediction.
- **Activation Function:** Linear.
- **Loss Function:** Mean Squared Error (MSE).
- **Optimizer:** Adam.

4.2 Hyperparameter Tuning

- Conducted hyperparameter optimization using Keras Tuner to find the optimal configuration:
 - LSTM units: [32, 64, 128].
 - Dropout rate: [0.1, 0.5].
 - Learning rate: [1e-4, 1e-2].
 - Batch size: [16, 32, 64].

```
[ ] train_size = int(len(X) * 0.6)
X_train, X_test = X[:train_size], X[train_size:]
y_train, y_test = y[:train_size], y[train_size:]
```

```
▶ #Build the LSTM Model
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout

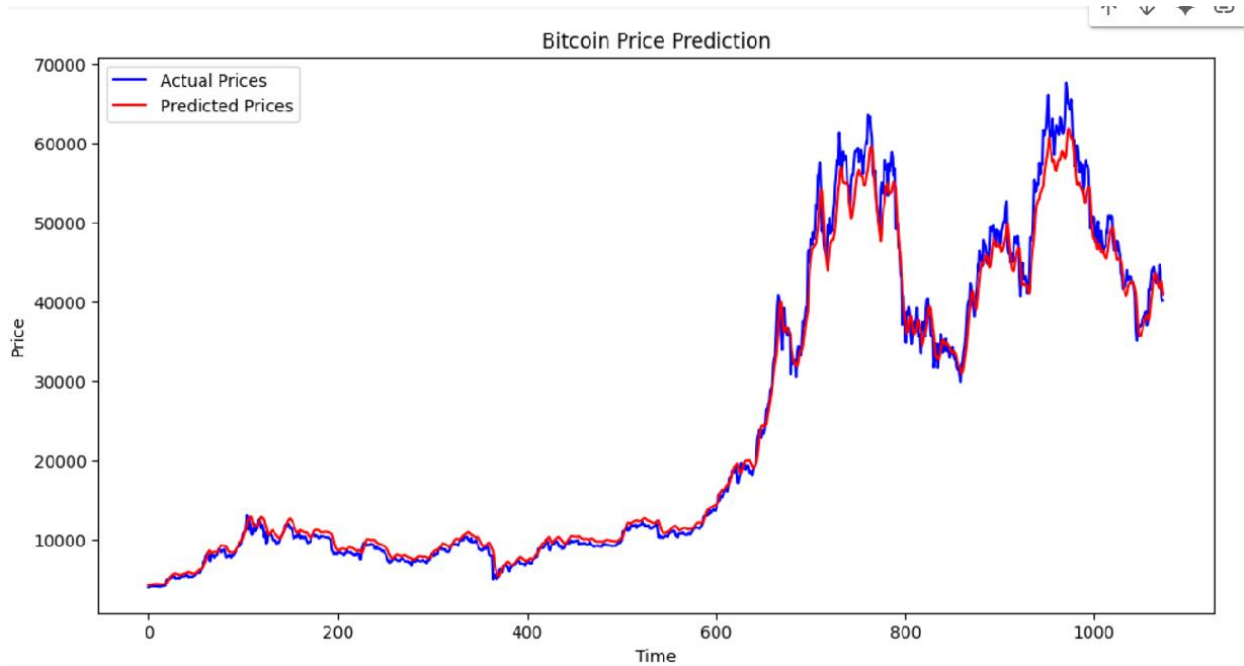
model = Sequential([
    LSTM(50, return_sequences=True, input_shape=(X_train.shape[1], X_train.shape[2])),
    Dropout(0.2),
    LSTM(50, return_sequences=False),
    Dropout(0.2),
    Dense(25),
    Dense(1)
])

model.compile(optimizer='adam', loss='mean_squared_error')
```


5. Evaluation

5.1 Observations

- LSTM performed better than Logistic Regression, SVC, and XGBoost in capturing temporal dependencies.
- Error metrics revealed that while LSTM captured trends, price deviations were still significant.



5.2 Comparison of the models implemented

Model	Accuracy	Strength	Limitations
Logistic Regression	52%	Simple, interpretable	Poor for non-linear relationships
SVC	48%	Handles non-linear data with kernels	Limited scalability, no sequential focus
XGBoost	92%	High efficiency, boosting performance	Not ideal for time-series dependencies
LSTM	80%	Captures sequential and temporal patterns	Complex, needs more computational resources

6. Insights and Limitations

6.1 Insights

- Feature engineering played a critical role in extracting meaningful patterns from historical data.
- LSTM successfully identified sequential trends, proving its suitability for time-series tasks.

6.2 Limitations

1. **High Volatility:** Bitcoin's unpredictable nature affects precision.
2. **Data Size:** Limited historical data may have constrained the model's learning.
3. **Accuracy:** While better than traditional models, 48% accuracy highlights room for improvement.

7. Conclusion

- The accuracy of identifying the trend of bitcoin price is 80% and the mean absolute percentage error is 5.71%.
- The project demonstrated that deep learning models, particularly LSTM, are more effective than traditional machine learning models for Bitcoin price prediction. However, further enhancements in data and methodology are needed to achieve higher accuracy and reliability in this highly volatile domain.