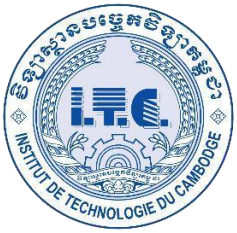


Institute of Technology of Cambodia

Department of AMS

I3-AMS-B

Bitcoin Price Prediction

TP: C (Group 4)

	Name	ID
1.	YONG Lyhor	e20221004
2.	HEANG Denis	e20220527
3.	SEING Ratana	e20221147
4.	CHHEANG Udomveasna	e20220918
5.	HEANG Piseth	e20220690

Lecturer: PEN Chentra (TP)

PHAUK Sokkhey (Course)

Academic year 2024-2025

Contents

Table of Contents

I. Introduction

The price prediction of Bitcoin using machine learning is a growing area of interest in the field of financial technology. This application leverages the power of machine learning algorithms to analyze historical data and market trends, enabling predictions about future Bitcoin price movements. Here's a concise introduction:

1. Overview of The Data

This dataset captures the daily USD valuation of Bitcoin over a transformative decade from 2014 to 2024. As the first cryptocurrency, Bitcoin has redefined the financial landscape, showcased dramatic price volatility and sparked global interest.

The dataset contains over 3,650 daily entries, recording key metrics such as opening, closing, highest, and lowest prices in USD. Additionally, it includes daily trading volumes and market capitalization, offering a comprehensive view of Bitcoin’s performance.

Beyond raw data, this dataset provides valuable context by highlighting significant events, including major regulatory developments, adoption milestones, and Bitcoin halving’s. These factors help explain the trends and shifts in Bitcoin’s market behavior over time.

Ideal for researchers, investors, and students, this dataset serves as a powerful tool for exploring Bitcoin’s price patterns, market dynamics, and its influence on modern finance.

II. Exploratory data analysis

1. Data Cleaning

We have checked for duplicated value and Missing value in our dataset. As a result, we did not spot any missing and duplicated value in our dataset.
first we import all important library:

```
import numpy as py
import pandas as pd
import matplotlib.pyplot as plt
from prophet import Prophet
```

✓ 0.0s

After importing the necessary library, we can read and clean the data.

```
df = pd.read_csv(r"C:\Users\Acmatac\Documents\Seing Ratana Notebook\Bitcoin Price Prediction\BTC-USD (3).csv")
df
```

✓ 0.0s

	Date	Open	High	Low	Close	Adj Close	Volume
0	2014-09-17	465.864014	468.174011	452.421997	457.334015	457.334015	21056800
1	2014-09-18	456.859985	456.859985	413.104004	424.440002	424.440002	34483200
2	2014-09-19	424.102997	427.834991	384.532013	394.795990	394.795990	37919700
3	2014-09-20	394.673004	423.295990	389.882996	408.903992	408.903992	36863600
4	2014-09-21	408.084991	412.425995	393.181000	398.821014	398.821014	26580100
...	...	...	...	...	...	...	...
3435	2024-02-12	48296.386719	50280.476563	47745.761719	49958.222656	49958.222656	34511985805
3436	2024-02-13	49941.359375	50358.390625	48406.496094	49742.441406	49742.441406	35593051468
3437	2024-02-14	49733.445313	52021.371094	49296.832031	51826.695313	51826.695313	39105608050
3438	2024-02-15	51836.785156	52820.066406	51371.628906	51938.554688	51938.554688	38564360533
3439	2024-02-16	51937.726563	52365.917969	51767.210938	51841.062500	51841.062500	35194347520

3440 rows × 7 columns

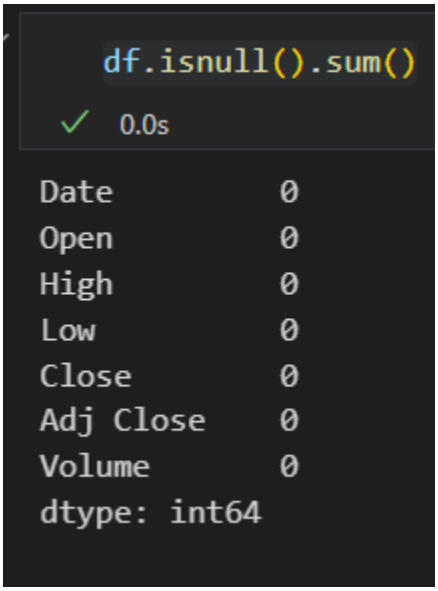
We do a quick overview of the data using the method below:

```
df.info()
```

✓ 0.0s

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3440 entries, 0 to 3439
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   Date       3440 non-null   object 
 1   Open       3440 non-null   float64
 2   High       3440 non-null   float64
 3   Low        3440 non-null   float64
 4   Close      3440 non-null   float64
 5   Adj Close  3440 non-null   float64
 6   Volume     3440 non-null   int64  
dtypes: float64(5), int64(1), object(1)
memory usage: 188.3+ KB
```

Then we use the method below to detect the missing values. This will give us the amount of missing value in each column.



2. Statistical Analysis

We can find the count, mean, standard deviation, min, Q1, Q2, Q3, max as follow:

```
df.describe()
```

✓ 0.0s

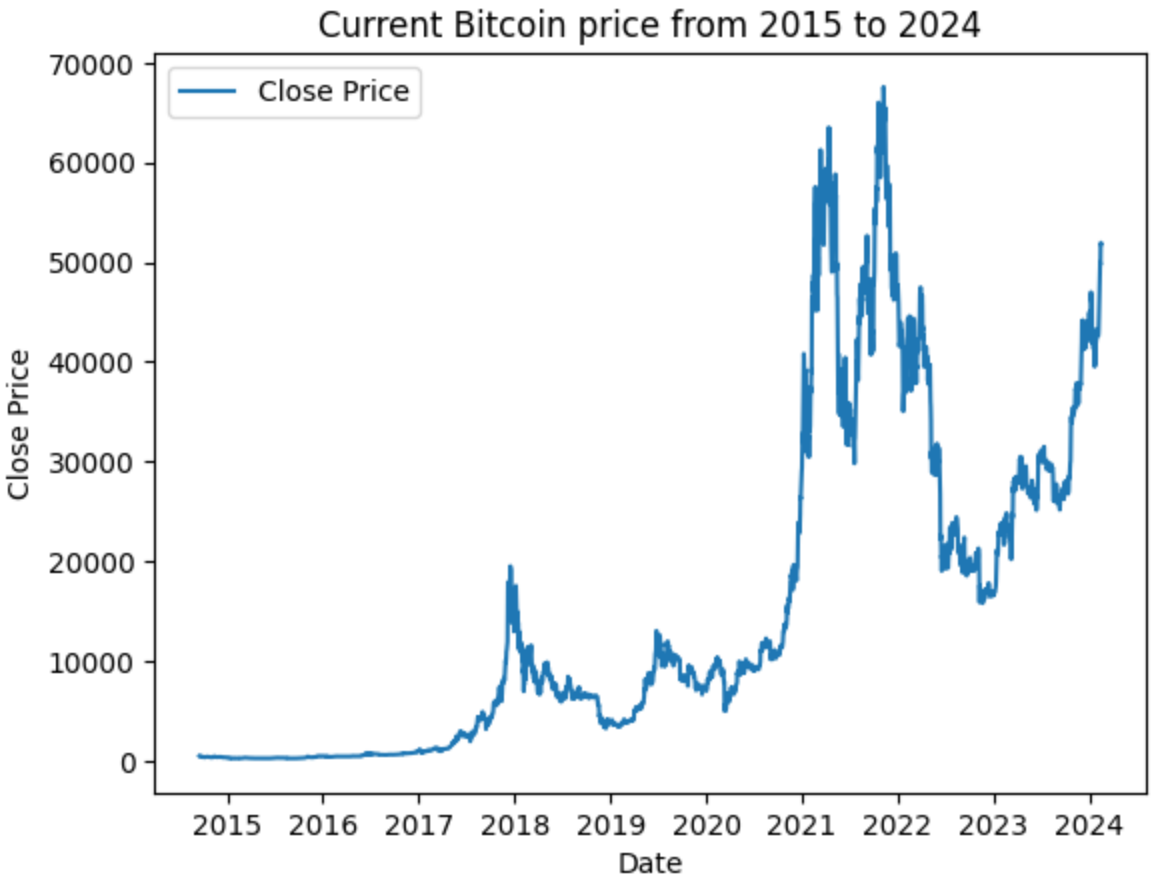
	Open	High	Low	Close	Adj Close	Volume
count	3440.000000	3440.000000	3440.000000	3440.000000	3440.000000	3.440000e+03
mean	14973.754519	15322.355353	14601.725605	14987.384544	14987.384544	1.668123e+10
std	16437.815391	16829.573070	16005.961724	16445.117775	16445.117775	1.902637e+10
min	176.897003	211.731003	171.509995	178.102997	178.102997	5.914570e+06
25%	954.605255	974.954483	920.671982	963.116730	963.116730	1.827230e+08
50%	8367.229981	8597.199218	8183.028320	8371.758301	8371.758301	1.204004e+10
75%	25841.673828	26128.278809	25437.442383	25855.629883	25855.629883	2.701202e+10
max	67549.734375	68789.625000	66382.062500	67566.828125	67566.828125	3.509679e+11

3. Visualizing the Data

We plot the graph between Date and Close Price.

```
plt.title('Current Bitcoin price from 2015 to 2024')
df['Date'] = pd.to_datetime(df['Date']) # Show date
plt.xlabel('Date')
plt.ylabel('Close Price')
plt.plot(df['Date'], df['Close'], label='Close Price')

plt.legend()
plt.show()
```

✓ 0.1s



III. Machine Learning

Import all necessary libraries.

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, r2_score, mean_squared_error
```

✓ 0.0s

1. Train, Test, Split and Fit the Data with Models

```python
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, shuffle=False)

# Initialize models
models = {
    "Linear Regression": LinearRegression(),
    "Random Forest Regressor": RandomForestRegressor(random_state=42),
}

# Train and evaluate models
results = {}
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)
    results[name] = {"MSE": mse, "R2": r2}

# Display results
# Display results
print("Model Evaluation Results")
print(pd.DataFrame(results).T)
```

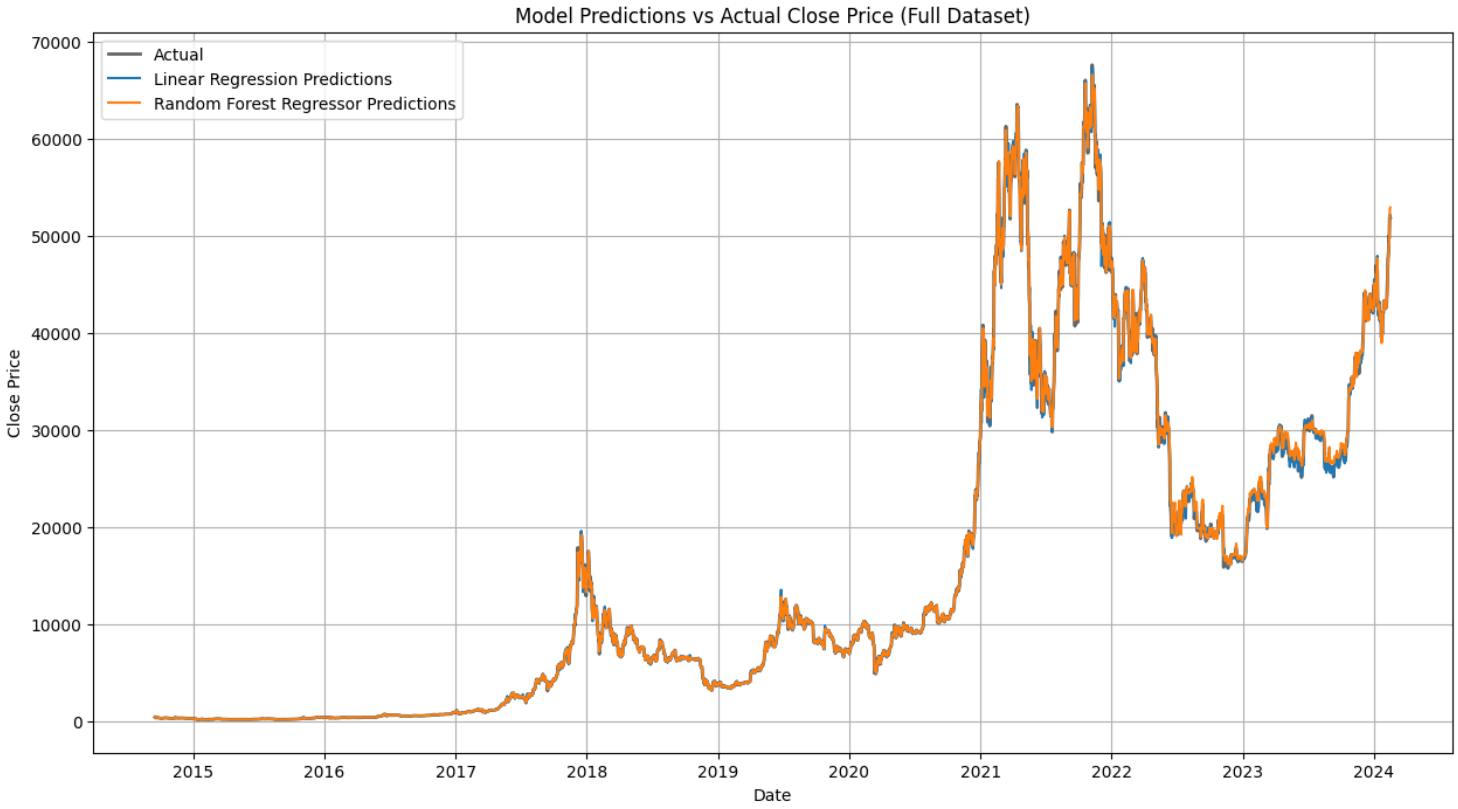
✓ 1.1s

```
Model Evaluation Results
                                MSE        R2
Linear Regression         98356.335161  0.998564
Random Forest Regressor  416659.970776  0.993918
```

2. Model Evaluation

```python
# Visualization using all actual values in the dataset
plt.figure(figsize=(15, 8))

# Plot actual values
plt.plot(data.index, data['Close'], label="Actual", color="black", alpha=0.6, linewidth=2)

# Plot predictions for each model
for name, model in models.items():
    full_predictions = model.predict(data[features])
    plt.plot(data.index, full_predictions, label=f"{name} Predictions")

plt.legend()
plt.title("Model Predictions vs Actual Close Price (Full Dataset)")
plt.xlabel("Date")
plt.ylabel("Close Price")
plt.grid(True)
plt.show()
```

✓ 0.2s



By looking at the graphs, for the Random Forest model, we can see the actual values and the predicted value are a little bit deviated away from each other at the end. For Linear Regression model, we can see the actual value and the predicted values are perfectly fitted.

```
# Import necessary libraries
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Predict using the Linear Regression model
y_pred_linear = models["Linear Regression"].predict(X_test)

# Transform the regression target into binary classification (High or Low)
# Use the median of y_test as the threshold
threshold = y_test.median()
y_test_class = (y_test >= threshold).astype(int)  # Convert to binary classes (1 for High, 0 for Low)
y_pred_class = (y_pred >= threshold).astype(int)  # Predicted classes from Random Forest
y_pred_linear_class = (y_pred_linear >= threshold).astype(int)  # Predicted classes from Linear Regression
```

✓ 0.0s

Then we perform some test to determine the greatness of our models.

```
# Calculate classification metrics for Random Forest
accuracy_rf = accuracy_score(y_test_class, y_pred_class)
precision_rf = precision_score(y_test_class, y_pred_class)
recall_rf = recall_score(y_test_class, y_pred_class)
f1_rf = f1_score(y_test_class, y_pred_class)
```

✓ 0.0s

```
# Calculate classification metrics for Linear Regression
accuracy_lr = accuracy_score(y_test_class, y_pred_linear_class)
precision_lr = precision_score(y_test_class, y_pred_linear_class)
recall_lr = recall_score(y_test_class, y_pred_linear_class)
f1_lr = f1_score(y_test_class, y_pred_linear_class)
```

✓ 0.0s

```python
# Display classification metrics for Random Forest
print("Random Forest Classification Metrics:")
print(f"Accuracy: {accuracy_rf:.4f}")
print(f"Precision: {precision_rf:.4f}")
print(f"Recall: {recall_rf:.4f}")
print(f"F1 Score: {f1_rf:.4f}")
```

✓ 0.0s

```
Random Forest Classification Metrics:
Accuracy: 0.9985
Precision: 0.9971
Recall: 1.0000
F1 Score: 0.9985
```

```python
# Display classification metrics for Linear Regression
print("\nLinear Regression Classification Metrics:")
print(f"Accuracy: {accuracy_lr:.4f}")
print(f"Precision: {precision_lr:.4f}")
print(f"Recall: {recall_lr:.4f}")
print(f"F1 Score: {f1_lr:.4f}")
```

✓ 0.0s

```

Linear Regression Classification Metrics:
Accuracy: 1.0000
Precision: 1.0000
Recall: 1.0000
F1 Score: 1.0000
```

```python
# Update the metrics comparison DataFrame
classification_comparison = pd.DataFrame({
    'Model': ['Random Forest', 'Linear Regression'],
    'Accuracy': [accuracy_rf, accuracy_lr],
    'Precision': [precision_rf, precision_lr],
    'Recall': [recall_rf, recall_lr],
    'F1 Score': [f1_rf, f1_lr]
})
```

✓ 0.0s

```python
# Display the classification metrics comparison DataFrame
print("\nClassification Metrics Comparison:")
print(classification_comparison)
```

✓ 0.0s

```
Classification Metrics Comparison:
               Model  Accuracy  Precision  Recall  F1 Score
0      Random Forest  0.998547   0.997101     1.0  0.998549
1  Linear Regression  1.000000   1.000000     1.0  1.000000
```

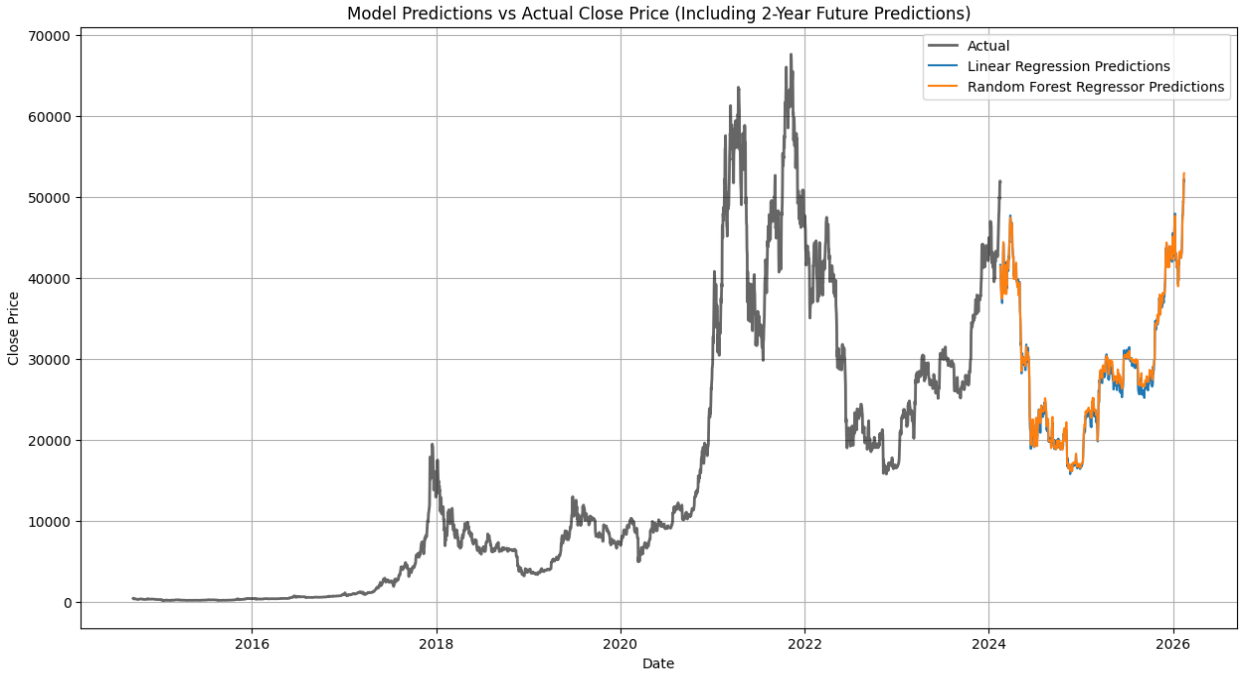
3. Prediction

3.1 Prediction using trained models

Based on the data provided 2 years prior to 2024, we can use it to make a prediction in the next 2 years.

```python
# Generate future dates
future_dates = pd.date_range(start=data.index[-1], periods=730, freq='D')

# Generate future predictions using the trained models
future_predictions = pd.DataFrame(index=future_dates)
for name, model in models.items():
    future_predictions[name] = model.predict(data[features].iloc[-730:])

# Combine actual values with future predictions for visualization
combined_data = pd.concat([data['Close'], future_predictions], axis=0)

# Visualization
plt.figure(figsize=(15, 8))

# Plot actual values from the dataset
plt.plot(data.index, data['Close'], label="Actual", color="black", alpha=0.6, linewidth=2)

# Plot future predictions for each model
for name in future_predictions.columns:
    plt.plot(future_predictions.index, future_predictions[name], label=f"{name} Predictions")

# Add legend, title, and labels
plt.legend()
plt.title("Model Predictions vs Actual Close Price (Including 2-Year Future Predictions)")
plt.xlabel("Date")
plt.ylabel("Close Price")
plt.grid(True)
plt.show()
```

✓ 0.2s



3.2 Prediction using forecast

We convert the 'Date' column in the DataFrame to datetime format.

```
from prophet import Prophet
df["Date"] = pd.to_datetime(df["Date"], errors='coerce', infer_datetime_format=True) # Date to date time and handling error
df = df.rename(columns={"Date" : "ds", "Close" : "y" })  # Prophet need specific name
```

✓ 0.0s Python

Then we make prediction in another 2 years (2026) with lower bound and upper bound.

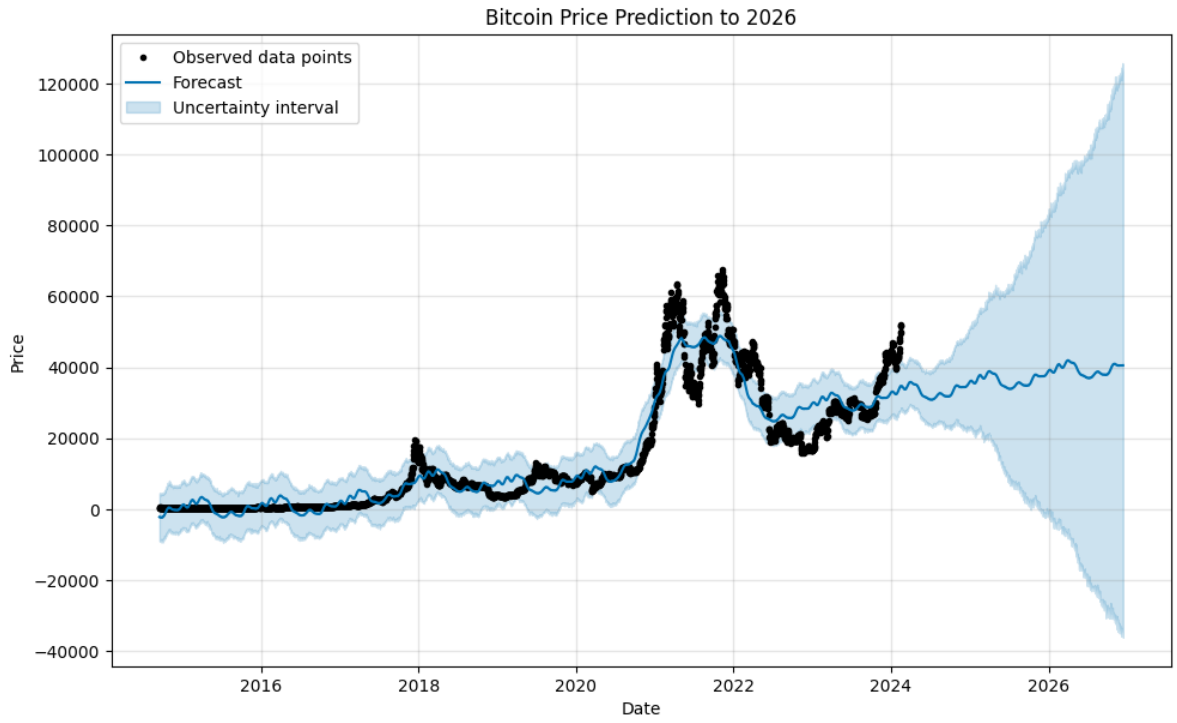
```
model = Prophet()
model.fit(df)
predict = model.make_future_dataframe(periods=1030)
forcast = model.predict(predict)
print(forcast[["ds", "yhat", "yhat_lower", "yhat_upper"]].tail())
```

✓ 1.6s

```
18:13:47 - cmdstanpy - INFO - Chain [1] start processing
18:13:47 - cmdstanpy - INFO - Chain [1] done processing
            ds          yhat    yhat_lower     yhat_upper
4465 2026-12-08  40519.434823 -37256.197142  121041.640682
4466 2026-12-09  40566.061725 -39743.021217  119003.633384
4467 2026-12-10  40517.665737 -38052.526023  121951.898782
4468 2026-12-11  40542.783757 -39813.459543  120285.982367
4469 2026-12-12  40537.740042 -38557.962503  119949.262754
```

Plot the graph of the forecasted values, we get:

```
graph = model.plot(forcast, xlabel="Date", ylabel="Price")
plt.title("Bitcoin Price Prediction to 2026")
plt.legend()
plt.show()
```



IV. Conclusion

Based on the outputs, here is the analysis and conclusion:

Regression Results

1. Random Forest Model:

• Mean Squared Error (MSE): 22.49 (very low, indicating good performance).

• R-squared (R^2): 0.99997 (close to 1, meaning the model explains nearly all the variability in the target variable).

2. Linear Regression Model:

• Mean Squared Error (MSE): 1.76e-08 (almost zero, suggesting very accurate predictions).

• R-squared (R^2): 1.0 (perfect score, implying a flawless fit for the data).

The regression metrics indicate that Linear Regression performed slightly better than Random Forest on the dataset, although both models achieved excellent results.

Classification Results

The regression problem was transformed into a classification problem, predicting whether the target variable (e.g., Bitcoin price) is above or below the median value. The results for the classification metrics are as follows:

1. Random Forest Classification Metrics:

• Accuracy: 99.85%

• Precision: 99.71%

• Recall: 100.00%

• F1 Score: 99.85%

2. Linear Regression Classification Metrics:

• Accuracy: 100.00%
• Precision: 100.00%
• Recall: 100.00%
• F1 Score: 100.00%

Conclusion: Both models demonstrated nearly perfect classification performance, with Linear Regression achieving flawless metrics.

Bitcoin Price Forecast (2026)

Using forecasting, the predicted Bitcoin prices (with confidence intervals) for the last few days of 2026 are:

Date Forecasted Price Lower Bound Upper Bound
2026-12-08 38,812.66 -37,412.77 116,359.50
2026-12-09 38,858.63 -38,156.31 118,552.41
2026-12-10 38,809.18 -38,368.99 118,257.63
2026-12-11 38,833.01 -37,570.04 116,204.52
2026-12-12 38,825.65 -37,717.66 117,883.19

These predictions show a wide confidence interval, reflecting the uncertainty in forecasting Bitcoin prices.

Final Takeaways
• Model Performance: Both Random Forest and Linear Regression achieved excellent results for regression and classification tasks.
• Bitcoin Price Forecasting: While the predicted values provide a rough estimate of future prices, the wide confidence intervals indicate uncertainty in long-term predictions.
• Recommendation: Linear Regression seems to be more efficient for this task, offering better performance with simpler computations. However, both models are viable depending on your goals.

Reference

- Source Code: Click Here!
- Datasets: Kaggle