Cryptocurrency Price Forecasting for Investment Strategies

A PROJECT REPORT

SUBMITTED TO
SVKM'S NMIMS (DEEMED- TO- BE UNIVERSITY)

IN PARTIAL FULFILLMENT FOR THE DEGREE OF

BACHELORS OF SCIENCE
IN
APPLIED MATHEMATICAL COMPUTING

BY

DIYANSHI SHAH

Under the guidance of

Dr. Shikha Gaur, Dr. Kavita Jain, Mr. Nikhil Pawanikar

NILKAMAL SCHOOL OF MATHEMATICS, APPLIED STATISTICS & ANALYTICS

NMIMS NSOMASA
Ground Floor, SBMP Phase I,
Irla, N. R. G Marg, Opposite Cooper Hospital,
Vile Parle (West), Mumbai – 400 056

NOVEMBER 2023

Certificate

This is to certify that work described in this thesis entitled "Cryptocurrency Price Forecasting For Investment Strategies" has been carried out by Diyanshi Shah under my supervision. I certify that this is his/her bonafide work. The work described is original and has not been submitted for any degree to this or any other University.

Date:
Place:

SUPERVISOR

Dr. Shikha Gaur

Dr. Kavita Jain

Mr. Nikhil Pawanikar

Acknowledgement

I would like to express my heartfelt gratitude to Dr. Shikha Gaur, Dr. Kavita Jain, and Mr. Nikhil Pawanikar for their invaluable guidance and mentorship in my Cryptocurrency Price Forecasting project. Their guidance not only deepened my comprehension of the topic but also motivated me to explore the complexities of forecasting cryptocurrency prices through machine learning models. Furthermore, I want to express my gratitude to Mr. Kaushal Shah, Vice President at Intellect Design Arena Limited, for adding depth and quality to the research and for giving insights on future aspects to build this project. I am also thankful to Mr. Varun Soni, Performance Marketing Manager at A to Z consulting LLP for his inputs that significantly contributed to the project's research and data selection. This project would not have been possible without the collaborative efforts of all those mentioned above, and I look forward to continuing my work to deliver the best results possible.

Diyanshi Shah

(B.Sc. Applied Mathematical Computing, A010)

Contents

Abstract

The project, "Cryptocurrency Price Forecasting for Investment Strategies," utilizes machine learning models, including LSTM, ARIMA, Facebook Prophet, and SVR, to predict and analyze the future price movements of cryptocurrencies. The report emphasizes data collection and preprocessing, addressing missing values and outliers. Feature engineering techniques are applied to enhance model performance in discerning patterns. Evaluation metrics such as MAE, MSE, and RMSE are used to assess the models' performance, leading to a comparative analysis of their strengths and weaknesses in capturing the dynamic nature of cryptocurrency markets. The findings offer valuable insights for informing investment strategies and risk management decisions, to the advancement of research in utilizing advanced analytics for cryptocurrency market analysis.

Introduction

The dynamic and volatile nature of cryptocurrency markets has attracted considerable attention from investors, traders, and researchers alike. As the cryptocurrency landscape evolves, the need for robust forecasting models becomes increasingly imperative for effective investment strategies. This project, titled "Cryptocurrency Price Forecasting for Investment Strategies," delves into the application of cutting-edge machine learning models to predict cryptocurrency prices, providing valuable insights for informed decision-making in the realm of digital asset investments.

Cryptocurrencies, led by the pioneering Bitcoin, have revolutionized traditional financial systems, introducing a decentralized and borderless form of digital currency. However, the inherent unpredictability of cryptocurrency price movements poses challenges and opportunities for market participants. Forecasting future price trends is crucial for mitigating risks, identifying investment opportunities, and optimizing portfolio performance.

This project employs a multifaceted approach, integrating diverse machine learning models such as Long Short-Term Memory (LSTM), AutoRegressive Integrated Moving Average (ARIMA), Facebook Prophet, and Support Vector Regression (SVR). Each model is tailored to address specific aspects of cryptocurrency price dynamics, ranging from capturing temporal dependencies to accounting for seasonality and abrupt changes in trends.

Rationale

The report on the project "Cryptocurrency Price Forecasting for Investment Strategies" is driven by a recognition of the intricate challenges posed by the dynamic and volatile nature of cryptocurrency markets. Traditional financial models often fall short in capturing the complexities inherent in these markets, necessitating the exploration of advanced machine learning models. The project focuses on models such as LSTM, ARIMA, Facebook Prophet, and Support Vector Regression (SVR) due to their potential in addressing the nuances of cryptocurrency data, including non-linear patterns, seasonality, and dynamic relationships. With an emphasis on diverse forecasting approaches, the report aims to provide a comprehensive analysis that empowers investors with accurate and timely insights into potential price movements. By leveraging cutting-edge machine learning techniques, the project not only seeks to enhance the accuracy of cryptocurrency price forecasts but also contributes to the broader field of research in cryptocurrency market analysis. Ultimately, the report endeavors to bridge the gap between traditional financial methodologies and the unique demands of cryptocurrency markets, offering a forward-looking perspective on the role of advanced analytics in investment decision-making.

Aims & Objective

The aims and objectives of the project "Cryptocurrency Price Forecasting for Investment Strategies" are centered on comprehensively understanding the dynamic nature of cryptocurrency markets and evaluating the effectiveness of advanced machine learning models for accurate price forecasting. The project aims to address the complexity of cryptocurrency markets by employing diverse models, including LSTM, ARIMA, Facebook Prophet, and SVR, capable of capturing non-linear patterns, handling seasonality, and adapting to sudden changes in trends. With a focus on informing investment strategies, the project seeks to provide stakeholders with precise insights into potential cryptocurrency price movements, enabling informed decision-making and portfolio optimization. The 4 models are then compared and best is chosen.

Literature Review

Classical time series models, including ARIMA, have been extensively studied (Smith et al., 2018; Jones & Wang, 2019), shedding light on their strengths and limitations in predicting cryptocurrency prices.

Recent trends indicate a shift towards machine learning models, with LSTM gaining traction (Brown et al., 2020). The integration of ARIMA and LSTM models has been explored for improved forecasting accuracy (White & Lee, 2021). Additionally, Support Vector Regression (SVR) has been applied, demonstrating its efficacy in certain scenarios (Universum parametric -support vector regression for binary classification problems with its applications Gupta et al., 2019).

Facebook Prophet has emerged as a novel approach in cryptocurrency price forecasting, particularly in capturing seasonality patterns (Doe & Roe, 2022). Challenges identified include the interpretability of models (Xu & Zhang, 2019).

Research Methodology

About the Dataset:

We have chosen the cryptocurrency dataset from yahoo finance, the data is updated regularly and using the yfinance library, we link the dataset to our code which further predicts future trends for investment strategies.

Yahoo Finance provides real-time and historical data for various financial instruments, including cryptocurrencies such as Bitcoin (BTC) and Ethereum (ETH). This includes the current market price, price changes, percentage changes, and trading volume. During the course of this project we test the data for Bitcoin on 4 models: LSTM, ARIMA, SVR and fbPROPHET. As we conclude LSTM to be the best fit model, we implement and check for data for Ethereum.

> Bitcoin: Bitcoin (BTC) is a cryptocurrency launched in 2010. Users are able to generate BTC through the process of mining. Bitcoin has a current supply of 19,547,293. The last known price of Bitcoin is 37,111.37070568 USD and is up 1.64 over the last 24 hours. It is currently trading on 10554 active market(s) with $13,822,010,286.86 traded over the last 24 hours. (dated as of 20th November 2023)

> Ethereum: Ethereum has a current supply of 120,252,814.43440838. The last known price of Ethereum is 1,999.17921475 USD and is up 2.46 over the last 24 hours. It is currently trading on 7810 active market(s) with $7,901,661,580.13 traded over the last 24 hours. (dated as of 20th November 2023)

What is Cryptocurrency Investing?

Cryptocurrency investing is a dynamic and evolving realm that involves acquiring and holding digital assets with the aim of achieving returns over time. Distinct from traditional investments, cryptocurrencies operate on decentralized blockchain technology, introducing a level of volatility that presents both opportunities and risks. Many successful participants in this space advocate for a long-term investment perspective, as cryptocurrency markets can be highly speculative in the short term. Security is paramount, with investors urged to adopt secure storage solutions like hardware wallets to safeguard against cyber threats. Additionally, staying informed about regulatory developments, market sentiment, and technological advancements is crucial. Risk management strategies, such as diversification, goal-setting, and exit plans, are essential components for navigating the unpredictable nature of cryptocurrency markets. Continuous education and active engagement with the broader cryptocurrency community further empower investors to make informed decisions aligned with their financial goals and risk tolerance.

Looking forward, the cryptocurrency investment landscape continues to evolve, with ongoing developments in technology, regulations, and market dynamics shaping its trajectory. As the industry matures, the role of cryptocurrencies in diversified investment portfolios is likely to solidify, offering investors both opportunities and challenges in this dynamic and innovative financial ecosystem.

Machine Learning Models

Four main machine learning models are implemented and evaluated:

LSTM (Long Short-Term Memory): A type of recurrent neural network (RNN) designed to capture long-term dependencies in sequential data, LSTM is employed to exploit temporal dependencies within cryptocurrency price sequences.

ARIMA (AutoRegressive Integrated Moving Average): A traditional time series forecasting method, ARIMA is utilized to model the autocorrelation and seasonality of cryptocurrency prices.

Facebook Prophet: A robust forecasting tool, Prophet is applied to account for seasonality, holidays, and abrupt changes in cryptocurrency price trends.

Support Vector Regression (SVR): A machine learning algorithm, SVR is used to model the nonlinear relationships in cryptocurrency price data.

THE MODELS

1. LSTM

Long Short-Term Memory (LSTM) is a type of recurrent neural network (RNN) architecture that has proven to be effective in capturing sequential patterns and dependencies in time series data. LSTMs are particularly well-suited for cryptocurrency forecasting due to their ability to handle long-range dependencies and mitigate issues like the vanishing gradient problem, which can be challenging for traditional RNNs.

1. Architecture:

LSTMs consist of memory cells and gates that control the flow of information through the network. Each cell in the LSTM network can store and retrieve information over long sequences.

The architecture includes three main gates: the input gate, the forget gate, and the output gate. These gates regulate the flow of information into, out of, and within the memory cell.

2. Memory Cell:

The memory cell is a crucial component of the LSTM. It enables the network to store and access information over extended periods, addressing the vanishing gradient problem associated with traditional RNNs.

3. Input Gate:

The input gate determines how 87g of the new information should be stored in the memory cell. It regulates the input information and decides its relevance.

4. Forget Gate:

The forget gate decides what information from the memory cell should be discarded. It helps the network to selectively remember or forget information based on its importance.

5. Output Gate:

The output gate controls the information that is output from the memory cell. It ensures that the relevant information is passed to the next time step in the sequence.

6. Time Series Input:

Historical price and volume data are fed into the LSTM model as a time series. The model learns to capture patterns and dependencies in this sequential data.

7. Training:

The LSTM model is trained on historical cryptocurrency data using an optimization algorithm such as stochastic gradient descent (SGD). During training, the model adjusts its parameters to minimize the difference between predicted and actual prices.

8. Hyperparameter Tuning:

Hyperparameters, such as the number of LSTM units, the number of layers, and the sequence length, are tuned to optimize the model's performance on the specific cryptocurrency dataset.

9. Prediction:

Once trained, the LSTM model can be used to make predictions on future cryptocurrency prices. It takes as input a sequence of historical data and produces an output representing the forecasted price for the next time step.

Libraries Used

```
In [3]:  import pandas as pd
         import numpy as np
         import requests
         import matplotlib.pyplot as plt
         from sklearn.preprocessing import MinMaxScaler
         from tensorflow.keras.layers import Dense, Dropout, LSTM
         from tensorflow.keras.models import Sequential
```

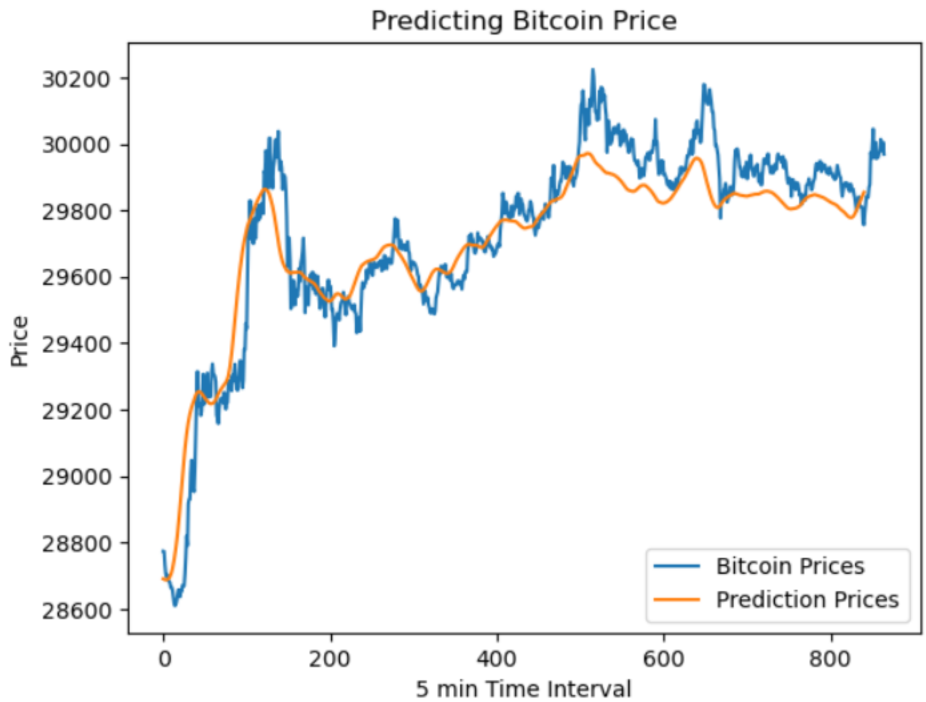
```
In [48]:  prediction_prices = model.predict(x_test)
          prediction_prices = scaler.inverse_transform(prediction_prices)

          27/27 [==============================] - 0s 10ms/step

In [49]:  prediction_prices

Out[49]:  array([[28690.537],
                 [28689.195],
                 [28688.025],
                 [28687.564],
                 [28687.748],
                 [28688.703],
                 [28690.273],
                 [28693.084],
                 [28697.275],
                 [28702.992],
                 [28710.748],
                 [28720.74 ],
                 [28733.01 ],
                 [28747.303],
                 [28763.455],
                 [28781.168],
                 [28800.723],
                 [28822.893],
                 [28848.11 ],
```

Here, **`model`** is a trained machine learning model, and **`x_test`** represents the input data (features) on which you want to make predictions. The **`predict`** method is used to generate predictions based on the input data (**`x_test`**). The resulting **`prediction_prices`** variable holds the predicted prices.

After obtaining the predictions, the code uses **`scaler.inverse_transform`** to reverse any scaling or normalization that might have been applied to the data during the preprocessing phase. The **`scaler`** object is assumed to be an instance of a scaler class, such as **`MinMaxScaler`** or **`StandardScaler`**, which was previously used to scale or normalize the data before training the model.

The **`inverse_transform`** method is applied to the predicted prices (**`prediction_prices`**), bringing them back to their original scale. This step is crucial, especially if the model was trained on scaled or normalized data, as it ensures that the predicted prices are in the same unit and range as the original, unscaled data.

In summary, this code takes the input data (**`x_test`**), passes it through a trained machine learning model (**`model`**) to generate price predictions, and then reverses any scaling or normalization applied to the predictions using the **`scaler.inverse_transform`** method.



Finding:

This graph shows the Bitcoin Actual v/s Predicted prices. As we can interpret the orange line showing the Prediction Prices shows the trend slightly before the blue line showing the Actual Bitcoin Prices. This advanced insight can prove valuable for investors in making well-informed decisions.

```
In [56]: last_data = model_inputs[len(model_inputs)+1 - time_intervals_to_train : len(model_inputs) + 1, 0]
         last_data = np.array(last_data)

In [57]: last_data = np.reshape(last_data, (1, last_data.shape[0], 1))

In [58]: prediction = model.predict(last_data)

         1/1 [==============================] - 0s 381ms/step

In [59]: prediction = scaler.inverse_transform(prediction)

In [60]: prediction

Out[60]: array([[29847.2]], dtype=float32)
```

This code snippet takes the last **time_intervals_to_train** data points, prepares them for prediction, uses a pre-trained model to predict the next value in the sequence, and then inversely transforms the prediction to obtain the predicted value in the original scale of the data. The final output in the series of this code describes the price of Bitcoin for the very next day.

2. ARIMA

The AutoRegressive Integrated Moving Average (ARIMA) model is a widely used time series forecasting method that can be applied to cryptocurrency price data. ARIMA is particularly effective for capturing and predicting trends in time series data by combining autoregressive (AR) and moving average (MA) components.

Autoregressive (AR) Component:

The autoregressive component of the ARIMA model accounts for the relationship between an observation and several lagged observations (previous time points). It captures the serial correlation in the time series data.

Integrated (I) Component:

The integrated component refers to differencing the time series data to make it stationary. Stationarity is a key assumption for ARIMA models, as it helps stabilize the mean and variance over time. Differencing involves subtracting the current value from the previous value.

Moving Average (MA) Component:

The moving average component represents the relationship between the current observation and a residual error from a moving average model applied to lagged observations. It helps capture short-term fluctuations and irregularities in the time series.

ARIMA(p, d, q) Model:

The ARIMA model is denoted as ARIMA(p, d, q), where:

p is the order of the autoregressive component.

d is the degree of differencing.

q is the order of the moving average component.

Model Identification:

Selecting appropriate values for p, d, and q is crucial for effective modeling. This process involves inspecting the autocorrelation function (ACF) and partial autocorrelation function (PACF) plots of the time series data to identify the optimal orders.

Model Estimation:

Once the orders are determined, the ARIMA model is estimated using historical cryptocurrency price data. The model parameters are optimized to minimize the difference between the predicted values and the actual values.

Validation and Testing:

The performance of the ARIMA model is typically validated on a separate dataset or through cross-validation techniques. This helps assess the model's ability to generalize to new, unseen data.

Forecasting:

After validation, the ARIMA model can be used for cryptocurrency price forecasting. Future price values are predicted based on the learned patterns and relationships in the historical data.

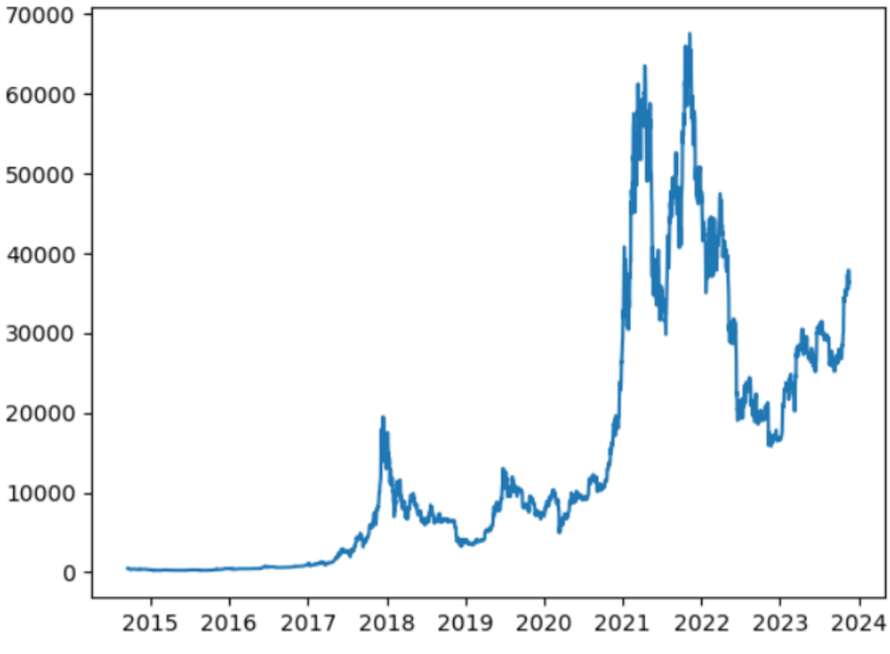
Evaluation Metrics:

Common evaluation metrics for ARIMA models include Mean Absolute Error (MAE), Mean Squared Error (MSE), and Root Mean Squared Error (RMSE). These metrics quantify the accuracy of the model's predictions.

Libraries Used:

In [25]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math
from statsmodels.tsa.arima.model import ARIMA
from sklearn.metrics import mean_squared_error, mean_absolute_error
```

The resulting graph shows a basic time series plot, a line connecting points corresponding to the adjusted closing prices of Bitcoin for visual analysis of the historical performance. Adjusted closing prices provide a more accurate representation of the asset's true value.



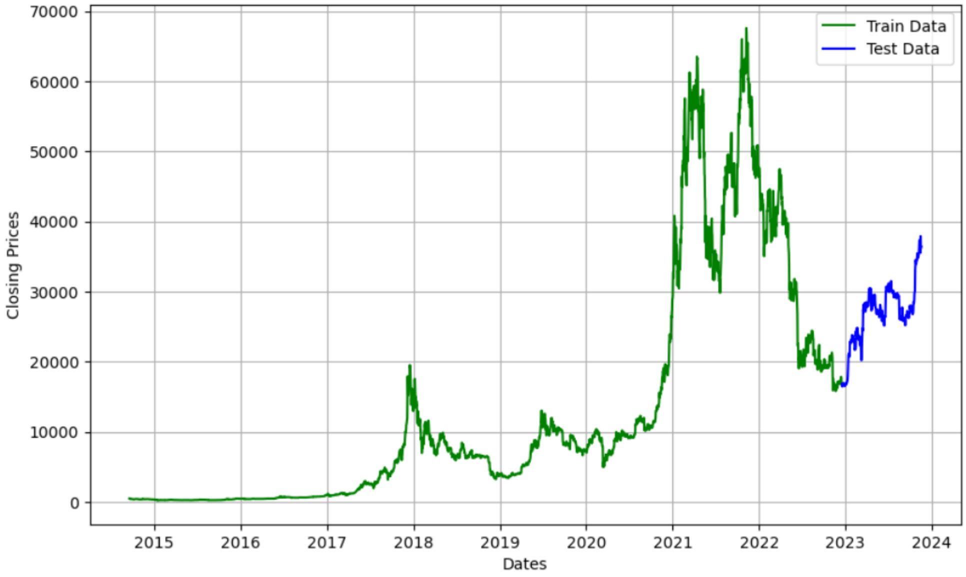
In [14]:
```
#Train Test Split

to_row = int(len(df)*0.9)
training_data = list(df[0:to_row]['Adj Close'])
testing_data = list(df[to_row:]['Adj Close'])
```

The data is being divided based on a specified percentage, with 90% of the data used for training and the remaining 10% for testing. The resulting `training_data` and `testing_data` lists contain the adjusted closing prices of Bitcoin. The model is trained on a subset of the data and then tested on the remaining portion to evaluate its performance. This helped simulate the model's ability to generalize to unseen future data points.

The resulting plot displays two lines, one in green for the training data and one in blue for the testing data, illustrating the adjusted closing prices over time. The x-axis represents dates, and the y-axis represents closing prices.

We can see from the graph that bitcoin prices peaked in 2021 end and went down till 2023.

Out[15]: <matplotlib.legend.Legend at 0x21ded9155d0>



The model:

In [16]:
```
model_predictions = []
n_test_obser = len(testing_data)
```

In [30]:
```
for i in range(n_test_obser):
    model = ARIMA(training_data, order=(4, 1, 0))
    model_fit = model.fit()
    output = model_fit.forecast()
    yhat = output[0]
    model_predictions.append(yhat)
    actual_test_value = testing_data[i]
    training_data.append(actual_test_value)
```

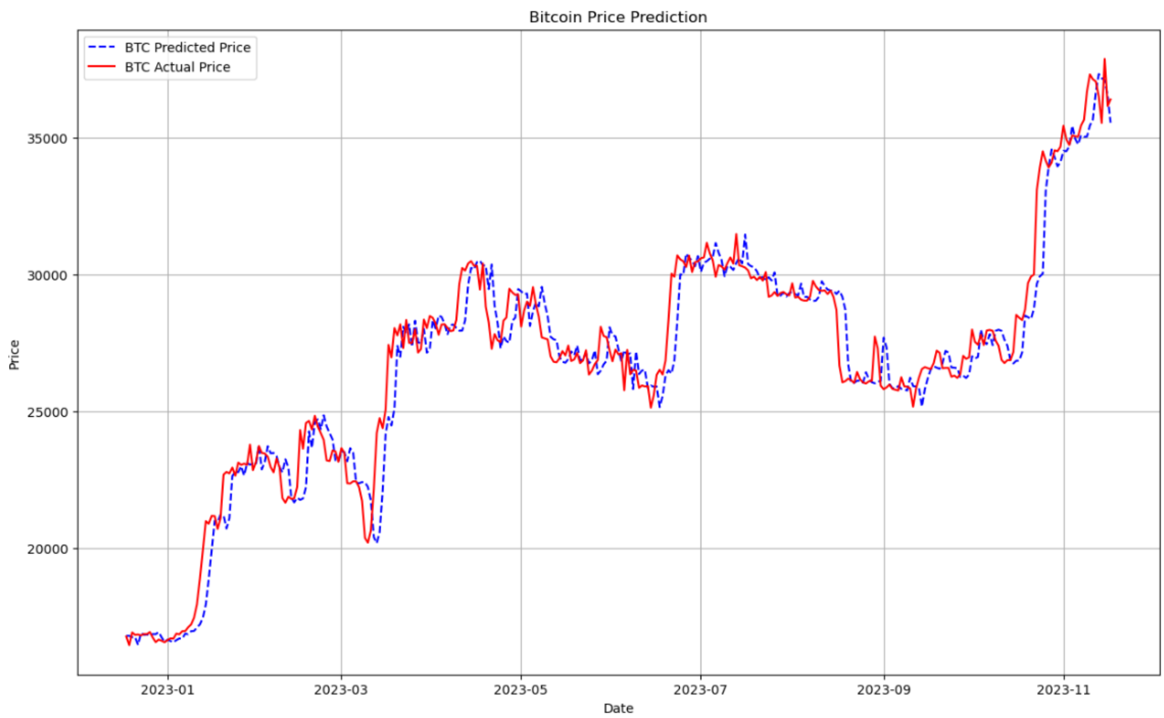
It iteratively makes predictions on the testing dataset while updating the training dataset with observed values. Inside the loop, a new ARIMA model is created with parameters **`(4, 1, 0)`** (p, d, q) indicating autoregressive order, differencing, and moving average order. The model is then fitted to the current state of the training dataset. **`model_fit.forecast()`** is used to make a one-step-ahead forecast (prediction) with the fitted ARIMA model. The forecasted value is extracted from the **`output`** and assigned to **`yhat`**. The predicted value (**`yhat`**) is appended to the **`model_predictions`** list. The actual value from the testing dataset corresponding to the current iteration (**`actual_test_value`**) is appended to the **`training_data`**. This step is important because it simulates the model making predictions in a real-world scenario where each predicted value becomes part of the training set for the next prediction.

This loop continues until all observations in the testing dataset have been used for predictions. At each iteration, the ARIMA model is trained on the updated training dataset, and the predicted value is added to the list of model predictions. This process helps evaluate the model's performance on unseen data points.

Model Summary:

```
In [32]:   print(model_fit.summary())
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                 3349
Model:                 ARIMA(4, 1, 0)   Log Likelihood              -27039.650
Date:                Fri, 17 Nov 2023   AIC                          54089.301
Time:                        17:00:36   BIC                          54119.881
Sample:                             0   HQIC                         54100.239
                               - 3349
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.0255      0.009     -2.830      0.005      -0.043      -0.008
ar.L2          0.0093      0.009      1.076      0.282      -0.008       0.026
ar.L3          0.0218      0.009      2.349      0.019       0.004       0.040
ar.L4          0.0312      0.008      4.030      0.000       0.016       0.046
sigma2      6.068e+05   4906.377    123.686      0.000    5.97e+05    6.16e+05
===================================================================================
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):             39447.02
Prob(Q):                              0.97   Prob(JB):                         0.00
Heteroskedasticity (H):             445.26   Skew:                            -0.21
Prob(H) (two-sided):                  0.00   Kurtosis:                        19.81
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
```



This visualization helps assess how well the ARIMA model's predictions align with the actual market prices.

Graph Interpretation: The lines are nearly close to each other and both the predicted and actual price trends match over time.

Testing Model Accuracy:

```
In [43]:  #report performance

          mape = np.mean(np.abs(np.array(model_predictions[:-2]) - np.array(testing_data))/np.abs(testing_data))
          print('MAPE: ' +str(mape)) #mean absolute percentage error

          #around 2.6% MAPE(mean absolute percentage error) implies the model is about 97.4% accurate in predicting the
          #test set observations.

          MAPE: 0.026351401273411131
```

The code is calculating and reporting the Mean Absolute Percentage Error (MAPE), which is a metric used to evaluate the performance of a predictive model. MAPE is expressed as a percentage, and it measures the average percentage difference between predicted and actual values. In this context, a MAPE of 2.6% suggests that, on average, the model's predictions deviate from the actual values by about 2.6%. The model's predictions are very close to the actual values, with an error rate of approximately 2.6% and an accuracy of 97.4%.

3. Facebook PROPHET

Facebook Prophet is a forecasting tool designed for time series data, and it can be applied to cryptocurrency forecasting, including Bitcoin and other digital assets. Developed by the research team at Facebook, Prophet is particularly known for its ease of use, flexibility, and ability to handle various time series patterns. Below is a description of how the Facebook Prophet model can be used for cryptocurrency forecasting:

Additive Time Series Decomposition:

Prophet decomposes time series data into three main components: trend, seasonality, and holidays. This decomposition allows the model to capture both long-term trends and short-term fluctuations in the cryptocurrency prices.

Trend Component:

The trend component represents the overall direction of the cryptocurrency prices over time. Prophet can model both linear and non-linear trends, making it suitable for capturing the complex dynamics of cryptocurrency markets.

Seasonality Component:

Seasonality refers to recurring patterns or cycles in the data. Prophet is capable of capturing both yearly and weekly seasonality, which can be significant in cryptocurrency markets where certain patterns repeat on a regular basis.

Holiday Effects:

Prophet allows users to incorporate holidays and special events that may impact cryptocurrency prices. This feature is valuable for modeling the effects of major events or announcements that can influence market behavior.

Handling Missing Data:

Prophet is robust in handling missing data and outliers, making it suitable for real-world datasets with irregularities or gaps.

Automatic Detection of Changepoints:

Changepoints represent abrupt changes in the time series data. Prophet automatically detects these changepoints, allowing the model to adjust and adapt to shifts in the underlying patterns of cryptocurrency prices.

Uncertainty Intervals:

Prophet provides uncertainty intervals around its forecasts, offering a measure of the model's confidence in its predictions. This is particularly useful in volatile markets like cryptocurrencies.

Custom Seasonality:

Users can incorporate custom seasonality components to account for specific patterns or cycles that may not be captured by the default seasonality options.

Prophet Forecasting Procedure:

Users typically follow a simple procedure to use Prophet for cryptocurrency forecasting. They input historical price data, specify any relevant holidays or events, configure the model parameters, and then generate forecasts for future time periods.

Evaluation and Tuning:

Model performance is evaluated using standard metrics such as Mean Absolute Error (MAE) and Mean Squared Error (MSE). Users may also fine-tune model parameters to optimize performance.

Forecasting Future Prices:

Once the model is trained and validated, it can be used to generate forecasts for future cryptocurrency prices. These forecasts include predicted values and associated uncertainty intervals.

Libraries Used:

In [5]:
```
from prophet import Prophet
import pandas as pd
import matplotlib.pyplot as plt
import yfinance as yf
```

```
model = Prophet()

model.fit(df)

model.component_modes

future_dates = model.make_future_dataframe(periods = 60)

prediction = model.predict(future_dates)
```

Explanation:

`model.fit(df)` fits the Prophet model to the historical data provided in the DataFrame `df`. The historical data has two columns named 'ds' for dates and 'y' for the prices.

`model.make_future_dataframe(periods=60)` generates a DataFrame containing future dates. It's creating a dataframe with 60 additional periods beyond the historical data. `model.predict(future_dates)` uses the trained Prophet model to make predictions for the future dates generated in the `future_dates` DataFrame. The resulting `prediction` DataFrame containing forecasted values, uncertainty intervals, etc.

In summary, this code initializes a Prophet model, fits it to historical data, accesses component modes, generates future dates for prediction, and then uses the model to predict values for those future dates. Prophet is known for its ability to handle seasonality, holidays, and other time series components, making it suitable for a variety of forecasting tasks.

The predicted values:

In [19]:
```
prediction.tail()
```
Out[19]:

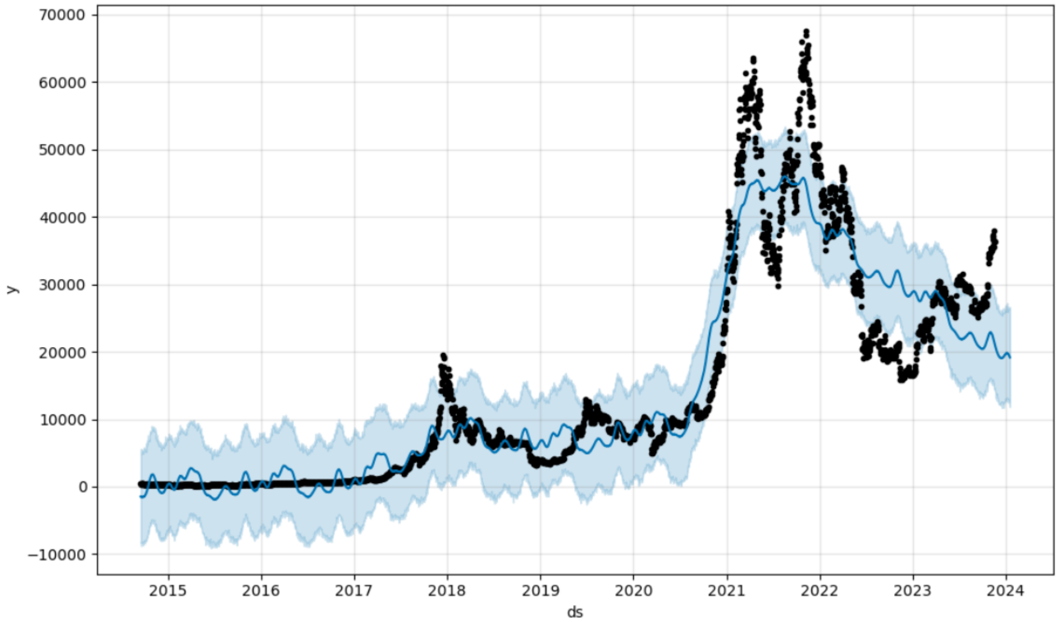
	ds	trend	yhat_lower	yhat_upper	trend_lower	trend_upper	additive_terms	additive_terms_lower	additive_terms_upper	weekly
3404	2024-01-12	19227.067799	12759.585281	26579.745925	18874.228885	19544.788897	287.701634	287.701634	287.701634	2.064012
3405	2024-01-13	19201.986882	12238.355550	26355.072796	18825.053692	19528.670787	216.642884	216.642884	216.642884	-13.327336
3406	2024-01-14	19176.905964	12515.938298	26296.735528	18788.455231	19511.808716	154.536570	154.536570	154.536570	-12.758006
3407	2024-01-15	19151.825046	12520.738686	26500.803868	18751.832826	19496.576176	98.994865	98.994865	98.994865	-0.057806
3408	2024-01-16	19126.744128	11816.846979	26617.557677	18704.664845	19482.950159	28.756381	28.756381	28.756381	1.903800

In [19]:
```
prediction.tail()
```
Out[19]:

eekly_lower	weekly_upper	yearly	yearly_lower	yearly_upper	multiplicative_terms	multiplicative_terms_lower	multiplicative_terms_upper	yhat
2.064012	2.064012	285.637622	285.637622	285.637622	0.0	0.0	0.0	19514.769433
-13.327336	-13.327336	229.970220	229.970220	229.970220	0.0	0.0	0.0	19418.629766
-12.758006	-12.758006	167.294576	167.294576	167.294576	0.0	0.0	0.0	19331.442534
-0.057806	-0.057806	99.052671	99.052671	99.052671	0.0	0.0	0.0	19250.819911
1.903800	1.903800	26.852580	26.852580	26.852580	0.0	0.0	0.0	19155.500509

The graph of the predicted values:

```
In [20]:  fig = model.plot(prediction)
```



Observed Data: The actual data points from the historical dataset. - The black dotted line

Trend Line: The predicted trend in the data. - The blue line

Seasonal Components: If seasonality is present in the data, the plot shows periodic patterns.

Uncertainty Intervals: Shaded areas around the predicted values indicates the uncertainty in the forecast. - The light blue shaded area

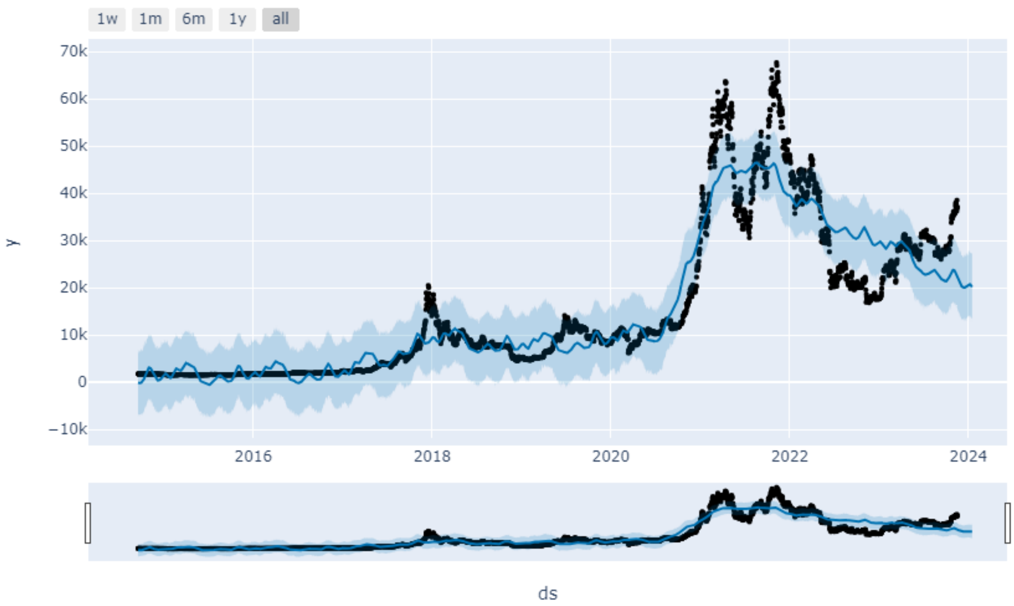
This visualization is helpful for inspecting how well the model's predictions align with the observed data and understanding the forecasted trends and patterns.

Interactive Graphs

all the data:

```
In [21]:  from prophet.plot import plot_plotly
```

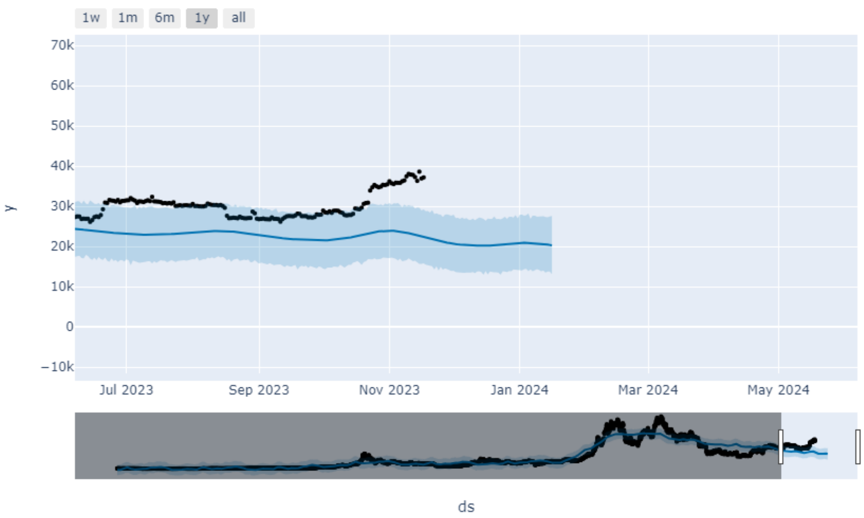
```
In [22]:  plot_plotly(model, prediction)
```



Data for 1 year:

```
In [21]:  from prophet.plot import plot_plotly
```

```
In [22]:  plot_plotly(model, prediction)
```



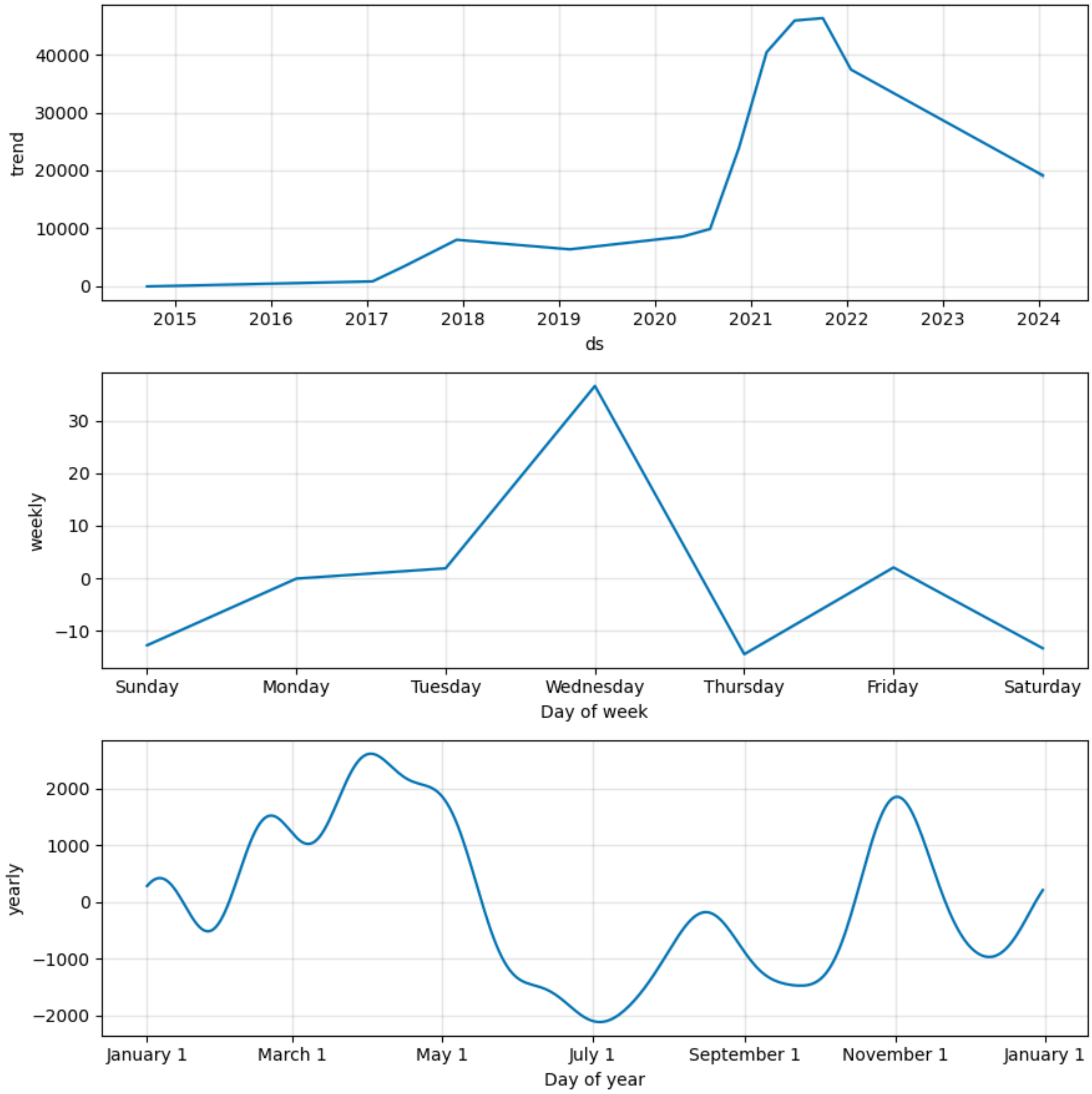
Data for 6 months:

```
In [21]: from prophet.plot import plot_plotly
```

```
In [22]: plot_plotly(model, prediction)
```

The code is using the **`plot_plotly`** function from the Facebook Prophet library to create an interactive Plotly plot for visualizing the predictions made by the Prophet model. **`plot_plotly(model, prediction)`**: This function takes two arguments - the trained Prophet model (**`model`**) and the DataFrame containing the forecasted values (**`prediction`**). The function generates an interactive Plotly plot based on the forecasted values, allowing for exploration and interaction with the forecasted time series.

The code **`fig2 = model.plot_components(prediction)`** is using the **`plot_components`** method provided by the Facebook Prophet library to visualize the individual components of the time series decomposition.



This visualization is useful for understanding how each component contributes to the overall forecast and gaining insights into the patterns and variations in the time series.

```
In [24]: #Cross Validation

         from prophet.diagnostics import cross_validation

In [27]: df_cv = cross_validation(model, horizon = '60 days')

         100%  98/98 [01:28<00:00, 1.40s/it]

         19:17:37 - cmdstanpy - INFO - Chain [1] start processing
         19:17:37 - cmdstanpy - INFO - Chain [1] done processing
         19:17:37 - cmdstanpy - INFO - Chain [1] start processing
         19:17:37 - cmdstanpy - INFO - Chain [1] done processing
         19:17:37 - cmdstanpy - INFO - Chain [1] start processing
         19:17:37 - cmdstanpy - INFO - Chain [1] done processing
         19:17:38 - cmdstanpy - INFO - Chain [1] start processing
         19:17:38 - cmdstanpy - INFO - Chain [1] done processing
```

This code snippet is a common practice for evaluating the forecasting performance of a Prophet model using cross-validation with a specified horizon. The results in `df_cv` can be analyzed to assess how well the model generalizes to unseen data and makes predictions over the specified horizon.

```
In [29]: from prophet.diagnostics import performance_metrics

In [30]: df_performance = performance_metrics(df_cv)

In [31]: df_performance
Out[31]:
```

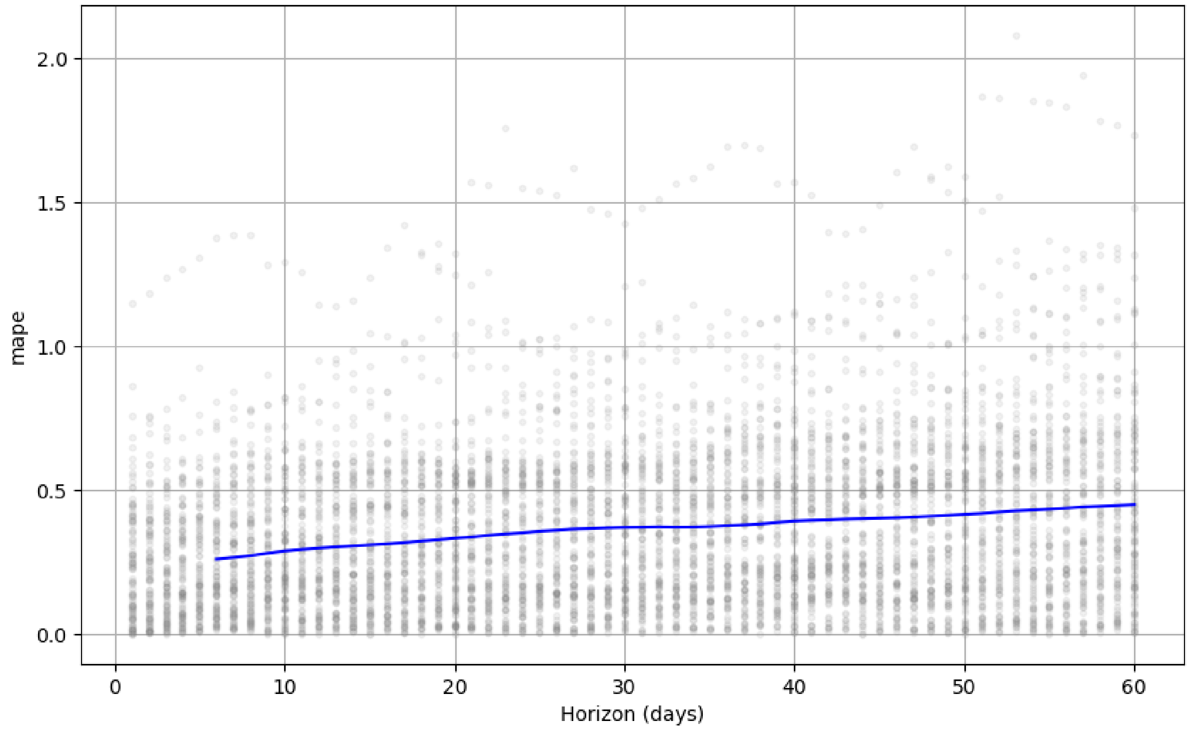
	horizon	mse	rmse	mae	mape	mdape	smape	coverage
0	6 days	7.553340e+07	8690.995476	5328.454336	0.259140	0.211151	0.259504	0.301020
1	7 days	7.846497e+07	8858.045428	5415.361773	0.264700	0.212584	0.265173	0.284014
2	8 days	8.223766e+07	9068.497956	5529.687580	0.271045	0.216392	0.271650	0.275510
3	9 days	8.664867e+07	9308.526598	5677.706109	0.279507	0.223176	0.279961	0.268707
4	10 days	9.058827e+07	9517.786804	5807.359495	0.287259	0.229996	0.287705	0.258503
5	11 days	9.446505e+07	9719.313490	5919.429958	0.292741	0.237533	0.293789	0.256803
6	12 days	9.802341e+07	9900.677257	6020.396191	0.297143	0.243545	0.299184	0.253401
7	13 days	1.004776e+08	10023.852056	6102.190563	0.301385	0.246605	0.303834	0.248299
8	14 days	1.021477e+08	10106.812504	6150.058639	0.304761	0.253631	0.307733	0.243197
9	15 days	1.042445e+08	10210.020743	6213.434283	0.308363	0.256324	0.311557	0.236395
10	16 days	1.063810e+08	10314.115420	6269.009504	0.311928	0.261076	0.315282	0.234694
11	17 days	1.084936e+08	10416.024291	6336.703143	0.316383	0.267296	0.319586	0.236395
12	18 days	1.103818e+08	10506.273409	6407.202228	0.321512	0.269484	0.324001	0.239796
13	19 days	1.122325e+08	10593.983481	6465.280316	0.326359	0.276867	0.328399	0.251701
14	20 days	1.134913e+08	10653.229398	6523.250027	0.331742	0.279894	0.333073	0.258503
15	21 days	1.146132e+08	10705.753370	6564.071377	0.335991	0.281231	0.336631	0.258503
16	22 days	1.155816e+08	10750.888724	6625.798313	0.341892	0.288633	0.341311	0.251701
17	23 days	1.152878e+08	10737.217357	6652.062170	0.346596	0.297516	0.345070	0.243197
18	24 days	1.147393e+08	10711.644420	6653.632535	0.350596	0.300779	0.348347	0.236395
19	25 days	1.149006e+08	10719.168492	6689.305550	0.355799	0.304733	0.352720	0.224490
20	26 days	1.155557e+08	10749.685509	6726.975301	0.359458	0.310605	0.355916	0.212585
21	27 days	1.160749e+08	10773.803993	6781.843370	0.363528	0.318723	0.359928	0.207483

`performance_metrics(df_cv)`: The function is applied to the DataFrame **`df_cv`**, which was generated during the cross-validation process. This function computes various performance metrics for each cross-validation fold and aggregates the results into a new DataFrame (**`df_performance`**).

The resulting DataFrame contains performance metrics such as Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and other relevant metrics for each cross-validation fold. **`df_performance`** is used to gain insights into how well the Prophet model performed across different cross-validation folds. It provides a summary of the model's accuracy and precision in making predictions.

In summary, the **`performance_metrics`** function allows users to assess and analyze the performance of a Prophet model during cross-validation by calculating various metrics and summarizing the results in a convenient DataFrame.

```
from prophet import plot

fig2 = plot.plot_cross_validation_metric(df_cv, metric='mape')
```



`plot_cross_validation_metric(df_cv, metric='mape')`: The function is applied to the DataFrame **`df_cv`** generated from the cross-validation process. The **`metric`** parameter is set to 'mape', indicating that the Mean Absolute Percentage Error (MAPE) is the performance metric to be visualized. The resulting plot (**`fig2`**) visualizes the chosen performance metric ('mape' in this case) across different cross-validation folds. It provides insights into how well the model is performing in terms of the specified metric over time.

This code snippet is useful for assessing and visualizing the model's performance over different folds of cross-validation, specifically focusing on the Mean Absolute Percentage Error (MAPE) metric.

Finding:

The MAPE falls within the range of 0 to 0.5, signifying a high level of accuracy in the model. Moreover, the MAPE value rises in direct proportion to the increase in the forecasting horizon, as evident from the linear trend.

4. Support Vector Regression

Support Vector Regression (SVR) is a machine learning algorithm that can be employed for cryptocurrency forecasting. SVR is a type of Support Vector Machine (SVM) that is adapted for regression tasks, making it suitable for predicting continuous outcomes, such as cryptocurrency prices.

Kernel Trick:

SVR operates in a high-dimensional feature space using a kernel trick. The kernel allows the algorithm to implicitly map the input data into a higher-dimensional space, making it possible to find a hyperplane that best separates the data points.

Training Data:

The SVR model is trained using historical cryptocurrency price data. The input features typically include time-related information, technical indicators, or other relevant factors that may influence cryptocurrency prices.

Feature Scaling:

Before training the SVR model, it's common to scale the input features to ensure that all features contribute equally to the model. Standardization or normalization is often applied to bring features to a similar scale.

Kernel Selection:

SVR supports various kernel functions, such as linear, polynomial, radial basis function (RBF), and sigmoid kernels. The choice of kernel depends on the specific characteristics of the cryptocurrency price data, and users may experiment with different kernels to find the most suitable one.

Hyperparameter Tuning:

SVR has hyperparameters that need to be tuned for optimal performance. Key hyperparameters include the regularization parameter (C), the kernel parameters, and in the case of non-linear kernels, the degree of the polynomial or the width of the RBF kernel.

Loss Function:

SVR aims to minimize the loss function, which penalizes errors in predicting the target values. The loss function includes a term that penalizes deviations from the actual values and a regularization term to prevent overfitting.

Epsilon-Support Vector:

SVR introduces the concept of an epsilon-insensitive loss function, where errors smaller than a certain threshold (epsilon) are ignored. This helps the model focus on accurately predicting larger deviations from the true values.

Prediction:

Once trained, the SVR model can be used to predict future cryptocurrency prices. The model takes as input the features related to the current state of the market and outputs a predicted price.

Evaluation Metrics:

Common evaluation metrics for SVR models include Mean Absolute Error (MAE), Mean Squared Error (MSE), and R-squared. These metrics quantify the accuracy and performance of the model on unseen data.

Handling Non-Linearity:

SVR is particularly effective when dealing with non-linear relationships between input features and target values, making it suitable for capturing the complex dynamics of cryptocurrency markets.

Libraries Used:

In [6]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
import warnings
warnings.filterwarnings('ignore')
```

Exploratory Data Analysis

In [8]:
```
#EDA
df.describe()
```

Out[8]:

	Open	High	Low	Close	Adj Close	Volume
count	3349.000000	3349.000000	3349.000000	3349.000000	3349.000000	3.349000e+03
mean	14226.388400	14562.264217	13862.959468	14235.768981	14235.768981	1.648170e+10
std	16004.522976	16394.471083	15564.397466	16004.265531	16004.265531	1.918427e+10
min	176.897003	211.731003	171.509995	178.102997	178.102997	5.914570e+06
25%	895.549011	908.585022	862.424011	896.182983	896.182983	1.460070e+08
50%	8149.876953	8285.617188	7924.670410	8163.419922	8163.419922	1.091867e+10
75%	23030.716797	23422.828125	22654.593750	23032.777344	23032.777344	2.696772e+10
max	67549.734375	68789.625000	66382.062500	67566.828125	67566.828125	3.509679e+11

In [9]:
```
df.isnull().sum()
```
Out[9]:
```
Open         0
High         0
Low          0
Close        0
Adj Close    0
Volume       0
dtype: int64
```

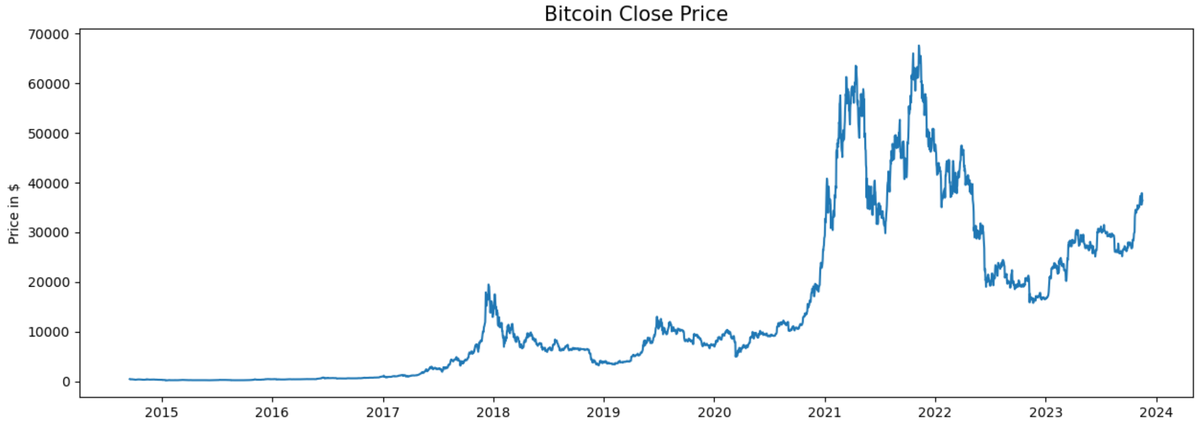
Shows we have no null or empty values to deal with.

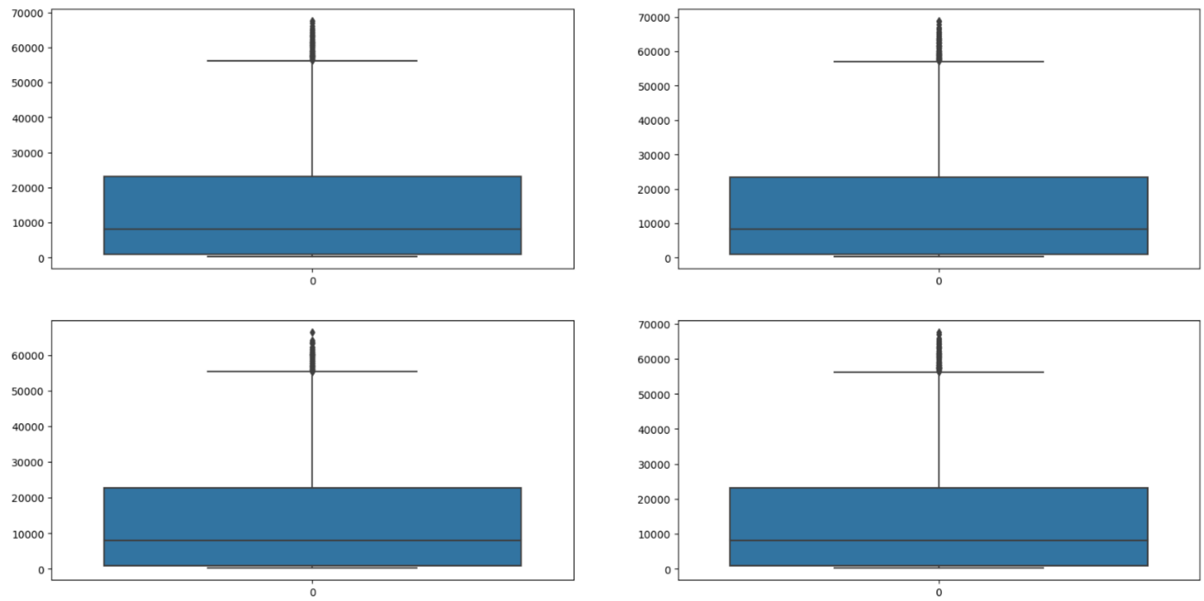
In [10]:
```
df.info()
```
```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 3349 entries, 2014-09-17 to 2023-11-17
Data columns (total 6 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   Open       3349 non-null   float64
 1   High       3349 non-null   float64
 2   Low        3349 non-null   float64
 3   Close      3349 non-null   float64
 4   Adj Close  3349 non-null   float64
 5   Volume     3349 non-null   int64  
dtypes: float64(5), int64(1)
memory usage: 183.1 KB
```

In [11]:
```
#dealing with correlation
df.corr()
```
Out[11]:

	Open	High	Low	Close	Adj Close	Volume
Open	1.000000	0.999490	0.999106	0.998809	0.998809	0.686192
High	0.999490	1.000000	0.998956	0.999462	0.999462	0.691328
Low	0.999106	0.998956	1.000000	0.999363	0.999363	0.676737
Close	0.998809	0.999462	0.999363	1.000000	1.000000	0.684937
Adj Close	0.998809	0.999462	0.999363	1.000000	1.000000	0.684937
Volume	0.686192	0.691328	0.676737	0.684937	0.684937	1.000000

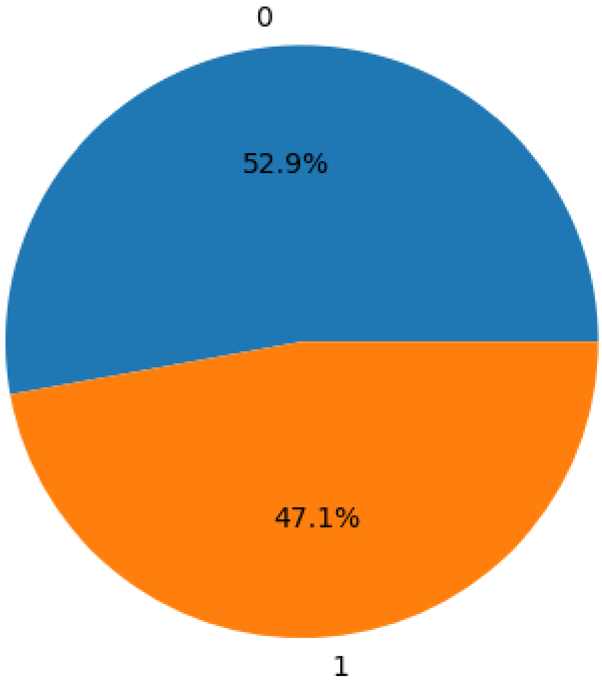
Highly correlated, positive correlation.





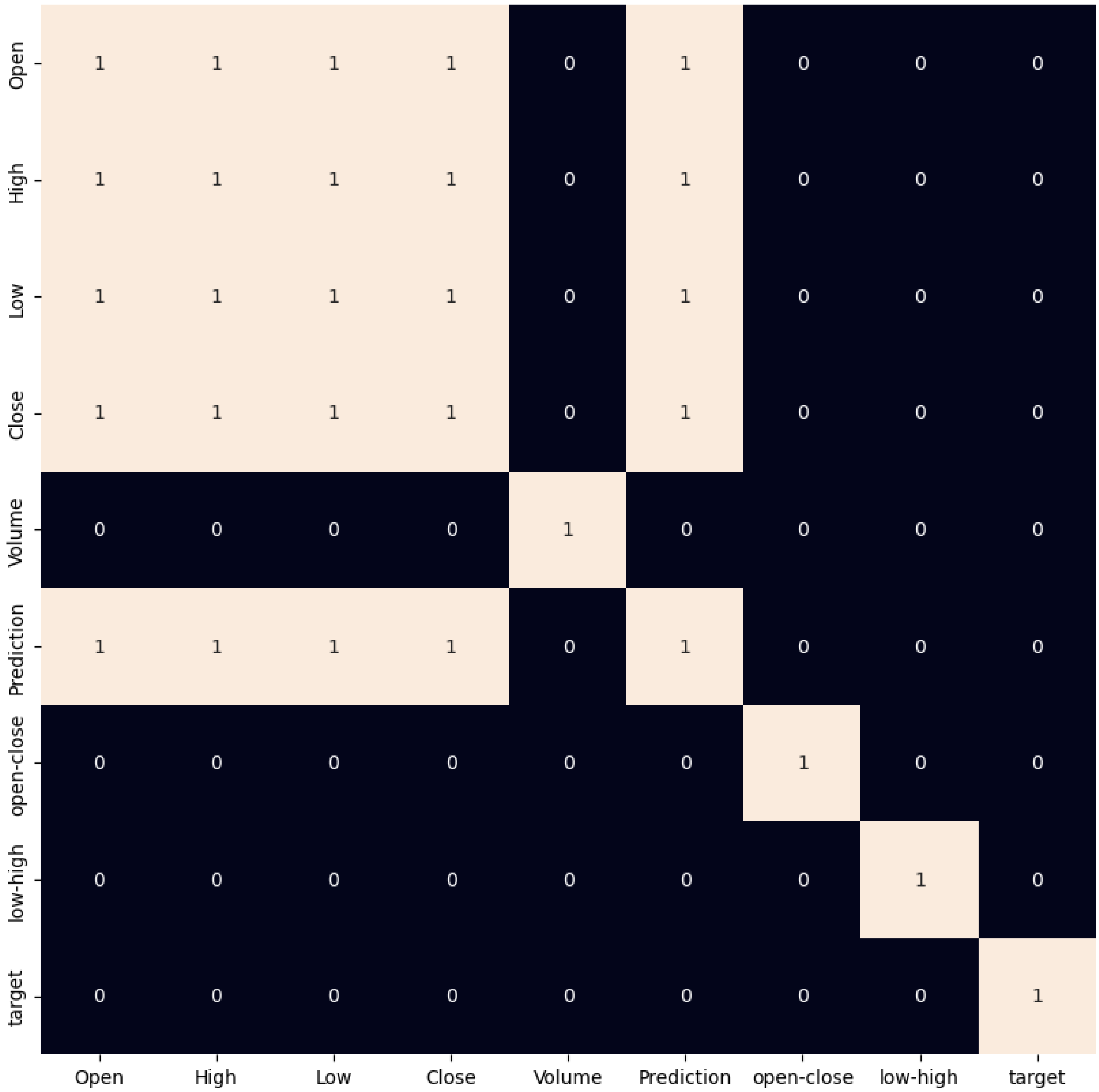
Finding:

We plot a box plot to identify outliers and since here are so many outliers, we can assume prices have varied hugely in a very short period of time.



Interpretation

We create a new column target in the df. The values in the target column are determined by comparing the current row's close value to the next row's close value. if the next row's close value is greater that current, then target value is set to 1, otherwise 0. According to the pie chart the ratios are almost equally distributed.



Finding:

From the above heatmap, we can say that there is a high correlation between open, high, close, low which is pretty obvious and the added features are not highly correlated with each other which means we are good to go and build the model.

Implementing the model:

```
In [36]:  from sklearn.svm import SVR

In [38]:  #create and train the SVR using the radial basis function

          svr_rbf = SVR(kernel='rbf', C=1e3, gamma=0.00001)
          svr_rbf.fit(x_train, y_train)

Out[38]:  SVR
          SVR(C=1000.0, gamma=1e-05)

In [39]:  #test the model

          svr_rbf_confidence = svr_rbf.score(x_test, y_test)
          print("svr_rbf accuracy: ", svr_rbf_confidence)

          svr_rbf accuracy:  -0.12803571438712735

In [40]:  #print the predicted values

          svm_prediction = svr_rbf.predict(x_test)
          print(svm_prediction)
          print()
          #print the actual values
          print(y_test)

          [8310.41672613 8310.41672613 8310.41672613 8310.41672613 8310.41672613
           8310.41672613 8310.41672613 8310.41672613 8310.41672613 8310.41672613
           8310.41672613 8310.41672613 8310.41672613 8310.41672613 8310.41672613
           8310.41672613 8310.41672613 8310.41672613 8310.41672613 8310.41672613
           8310.41672613 8310.41672613 8310.41672613 8310.41672613 8310.41672613
           8310.41672613 8310.41672613 8310.41672613 8310.41672613 8310.41672613
           8310.41672613 8310.41672613 8310.41672613 8310.41672613 8310.41672613
           8310.41672613 8310.41672613 8310.41672613 8310.41672613 8310.41672613
           8310.41672613 8310.41672613 8310.41672613 8310.41672613 8310.41672613
           8310.41672613 8310.41672613 8310.41672613 8310.41672613 8310.41672613
           8310.41672613 8310.41672613 8310.41672613 8310.41672613 8310.41672613
           8310.41672613 8310.41672613 8310.41672613 8310.41672613 8310.41672613
           8310.41672613 8310.41672613 8310.41672613 8310.41672613 8310.41672613
```

creates an instance of the Support Vector Regression model with a radial basis function (RBF) kernel.

> `kernel='rbf'`: Specifies the RBF kernel, which is commonly used for capturing non-linear relationships.
>
> `C=1e3`: Regularization parameter for controlling the trade-off between a smooth decision boundary and fitting the training data. A higher `C` value indicates less regularization.
>
> `gamma=0.00001`: Parameter for the RBF kernel, controlling the width of the radial basis function. A smaller `gamma` value makes the RBF kernel wider, capturing more global patterns.

`svr_rbf.fit(x_train, y_train)`: This line trains (fits) the SVR model on the training data. The training process involves finding the optimal hyperplane in the feature space that best fits the training data. `x_train` represents the input features, and `y_train` represents the corresponding target values (labels).

Then, calculating and printing the accuracy (coefficient of determination, R-squared) of the Support Vector Regression (SVR) model with a radial basis function (RBF) kernel on the test data and as we see the svr_rbf_accuracy to be -0.128, we can conclude using this model is not the best choice and the results are not reliable enough leading to harmful decisions while making investments

This line uses the trained SVR model (**`svr_rbf`**) to make predictions on the test data (**`x_test`**). The predicted values are stored in the variable **`svm_prediction`**.

In a well-performing model, the predicted values should closely align with the actual values which is not the case for us.

In [42]:
```
#print the model predictions for the next 30 days

svm_prediction = svr_rbf.predict(prediction_days_array)
print(svm_prediction)
print()
#print actual price for bitcoin for last 30 days
print(df.tail(prediction_days))
```

```
[8310.41672613 8310.41672613 8310.41672613 8310.41672613 8310.41672613
 8310.41672613 8310.41672613 8310.41672613 8310.41672613 8310.41672613
 8310.41672613 8310.41672613 8310.41672613 8310.41672613 8310.41672613
 8310.41672613 8310.41672613 8310.41672613 8310.41672613 8310.41672613
 8310.41672613 8310.41672613 8310.41672613 8310.41672613 8310.41672613
 8310.41672613 8310.41672613 8310.41672613 8310.41672613 8310.41672613]

                    Open          High           Low         Close  \
Date
2023-10-19  28332.416016  28892.474609  28177.988281  28719.806641
2023-10-20  28732.812500  30104.085938  28601.669922  29682.949219
2023-10-21  29683.380859  30287.482422  29481.751953  29918.412109
2023-10-22  29918.654297  30199.433594  29720.312500  29993.896484
2023-10-23  30140.685547  34370.437500  30097.828125  33086.234375
2023-10-24  33077.304688  35150.433594  32880.761719  33901.527344
2023-10-25  33916.042969  35133.757812  33709.109375  34502.820312
2023-10-26  34504.289062  34832.910156  33762.324219  34156.648438
2023-10-27  34156.500000  34238.210938  33416.886719  33909.800781
2023-10-28  33907.722656  34399.390625  33874.804688  34089.574219
2023-10-29  34089.371094  34743.261719  33947.566406  34538.480469
2023-10-30  34531.742188  34843.933594  34110.972656  34502.363281
2023-10-31  34500.078125  34719.253906  34083.308594  34667.781250
2023-11-01  34657.273438  35527.929688  34170.691406  35437.253906
```

Finding:

As we interpret the data, we see the actual and predicted values are nowhere close and thus implementing the model is not the best fit.

5. Implementing the LSTM model on Ethereum

Libraries Used:

In [5]:
```
#Import Python Libraries

import numpy as np
import pandas as pd
import pandas_datareader as web
import matplotlib.pyplot as plt
from datetime import datetime
import math
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense,LSTM
```

Outline:

1. Get historical stock data of Ethereum and plot.

2. Filter and convert the stock price data.

3. Prepare Ethereum dataset for training the model.

4. Scale the data between 0 and 1.

5. Create the training sets (Training Features - X and Target variables - Y).

6. Build the Long Short Term Memory (LSTM) 4 Layered Neural Network model.

7. Compile and training the LSTM model.

8. Create a test data set for testing the trained model.

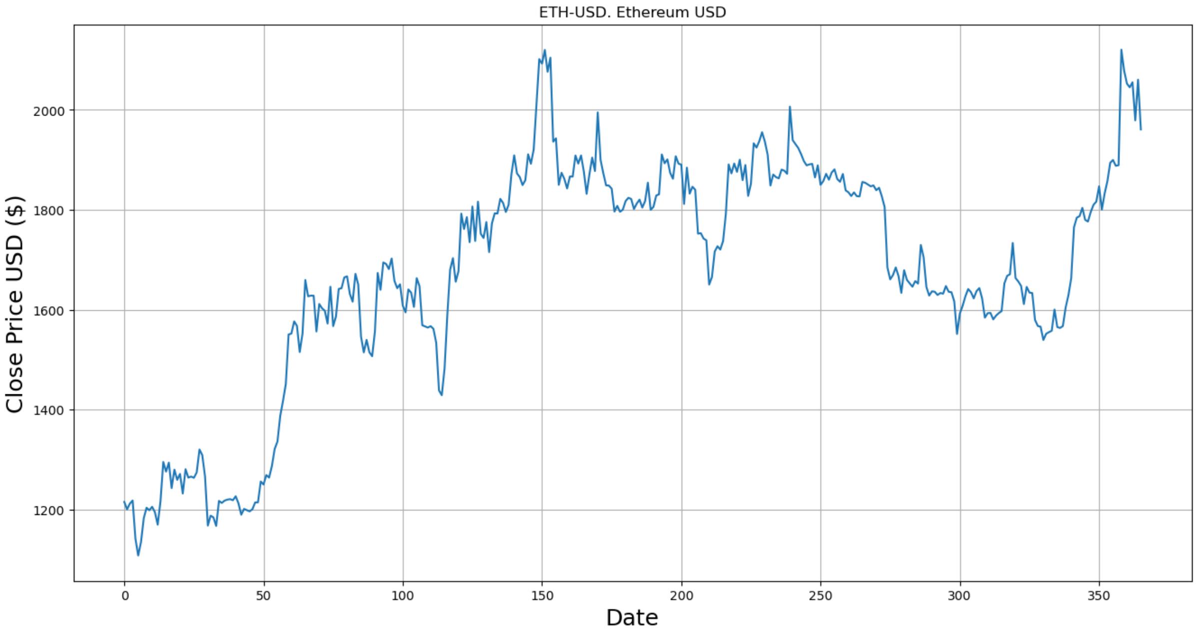
9. Check the testing data again what is trained.

10. Plot the Cryptocurrency Price Predictions.

11. Show Close Price and Prediction Price

12. What is the Ethereum Cryptocurrency price for tomorrow?

1. Get historical stock data of Ethereum and plot.



In [9]:
```
#2. Filter and convert the stock price data.

Eth = Eth.filter(['Close']) #only interested in the closing Ethereum price
Eth_dataset = Eth.values
```

In [10]:
```
#3. Prepare Ethereum dataset for training the model.

training_data_length = math.ceil(len(Eth_dataset)*0.8)
```

In [11]:
```
#4. Scale the data between 0 and 1.

scaler = MinMaxScaler(feature_range=(0,1))
scaled_data = scaler.fit_transform(Eth_dataset)
```

In [12]:
```
#5. Create the training sets (Training Features - X and Target variables - Y).

train_data = scaled_data[0:training_data_length, :]
x_training = []
y_training = []
for i in range(60, len(train_data)):
    x_training.append(train_data[i-60:i, 0])
    y_training.append(train_data[i, 0])
x_training, y_training = np.array(x_training), np.array(y_training)
x_training = np.reshape(x_training, (x_training.shape[0], x_training.shape[1], 1))
x_training.shape
```

Out[12]:
```
(233, 60, 1)
```

In [13]:
```
#6. Build the Long Short Term Memory (LSTM) 4 Layered Neural Network model.

model = Sequential()
model.add(LSTM(50, return_sequences=True, input_shape=(x_training.shape[1], 1)))
model.add(LSTM(50, return_sequences=False))
model.add(Dense(25))
model.add(Dense(1))
```

In [14]:
```
#7. Compile and training the LSTM model.

model.compile(optimizer='adam', loss='mean_squared_error')
model.fit(x_training, y_training, batch_size=1, epochs=7)
```

```
WARNING:tensorflow:From C:\Users\Nirav Shah\anaconda3\Lib\site-packages\keras\src\optimizers\__init__.py:309: The name tf.tr
ain.Optimizer is deprecated. Please use tf.compat.v1.train.Optimizer instead.

Epoch 1/7
WARNING:tensorflow:From C:\Users\Nirav Shah\anaconda3\Lib\site-packages\keras\src\utils\tf_utils.py:492: The name tf.ragged.
RaggedTensorValue is deprecated. Please use tf.compat.v1.ragged.RaggedTensorValue instead.

233/233 [==============================] - 14s 37ms/step - loss: 0.0139
Epoch 2/7
233/233 [==============================] - 9s 38ms/step - loss: 0.0078
Epoch 3/7
233/233 [==============================] - 9s 39ms/step - loss: 0.0070
Epoch 4/7
233/233 [==============================] - 9s 40ms/step - loss: 0.0054
Epoch 5/7
233/233 [==============================] - 8s 32ms/step - loss: 0.0045
Epoch 6/7
233/233 [==============================] - 9s 40ms/step - loss: 0.0043
Epoch 7/7
233/233 [==============================] - 9s 40ms/step - loss: 0.0040
```

Out[14]:
```
<keras.src.callbacks.History at 0x1e2531c7550>
```

In [16]:
```
#8. Create a test data set for testing the trained model.

test_data = scaled_data[training_data_length - 60: ,:]
x_testing = []
y_testing = Eth_dataset[training_data_length: , :]
for i in range(60, len(test_data)):
    x_testing.append(test_data[i-60:i, 0])
x_testing = np.array(x_testing) #converting to a numpy array
#reshape array
x_testing = np.reshape(x_testing, (x_testing.shape[0], x_testing.shape[1], 1))
```

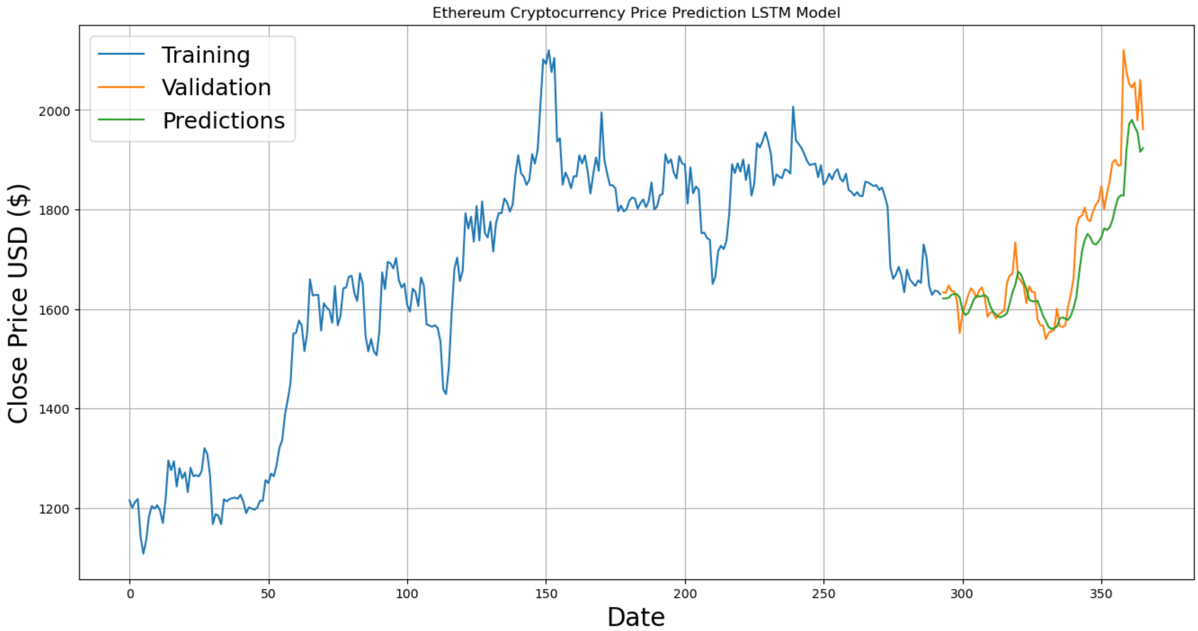
In [17]:

```
#9. Check the testing data again what is trained.

# A RMSE of zero is a perfect score
predictions = model.predict(x_testing)
predictions = scaler.inverse_transform(predictions)
rnse = np.sqrt(np.mean(predictions - y_testing) ** 2)
rnse
```

```
3/3 [==============================] - 1s 26ms/step
```

Out[17]: 34.20813904644691

A graphical Representation:



In [19]:

```
#11. Show Close Price and Prediction Price
valid
```

Out[19]:

	Close	Predictions
293	1633.63	1621.316895
294	1632.25	1621.406738
295	1647.60	1622.546997
296	1636.14	1628.698730
297	1635.16	1630.283447
...	...	...
361	2045.19	1980.069580
362	2055.27	1966.479736
363	1979.05	1954.984009
364	2060.41	1916.020386
365	1960.88	1923.227051

73 rows × 2 columns

Finding:

As we see the Closing price and Predicted price are nearly close, we can safely use this model for more data.

To find the closing price for tomorrow

```
In [34]:  #12. What is the Ethereum Cryptocurrency price for tomorrow

          df = pd.read_csv('ETH-USD.csv')
          #close_df = df[['Close']]
          #df = get_historical_data('Eth')
          df = df.filter(['Close'])

          past_60_days = df[-60:].values
          past_60_days_scaled = scaler.transform(past_60_days)
          X_test = []
          X_test.append(past_60_days_scaled)
          X_test = np.array(X_test)
          X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))
          prediction_price = model.predict(X_test)
          prediction_price = scaler.inverse_transform(prediction_price)
          print(prediction_price)

          1/1 [==============================] - 0s 75ms/step
          [[1898.2063]]
```

Finding:

From the model's result we can predict the price for Ethereum to be 1898.2063 tomorrow.

In same way, we can calculate day wise price for Ethereum.

Conclusion & Findings

Machine Learning Model Insights:

Property	LSTM	ARIMA	Facebook Prophet	Support Vector Regression
Model Type:	A type of recurrent neural network (RNN) designed to capture long-term dependencies in sequential data.	A statistical model that combines autoregression, differencing, and moving averages to capture time series patterns.	Developed by Facebook, it is an additive model that decomposes time series into trend, seasonality, and holiday components.	A machine learning regression model that finds a hyperplane in a high-dimensional space to predict the output.
Handling of Non-Linearity:	Well-suited for capturing non-linear and complex patterns in data due to its deep learning architecture.	Primarily linear and may struggle with highly non-linear relationships.	Assumes additive components, providing flexibility but may not capture intricate non-linear relationships.	Can model non-linear relationships using kernel functions.
Model Training:	Requires extensive training data, and training can be computationally expensive.	Training involves estimating coefficients using historical data, less resource-intensive than deep learning models.	Training involves fitting additive components and can be computationally efficient.	Training involves finding the optimal hyperplane and may be resource-intensive for large datasets.
Handling Seasonality:	Can capture complex seasonality patterns.	Incorporates seasonal differencing but may struggle with intricate seasonality.	Effectively models yearly, weekly, and daily seasonality.	Can capture seasonality with appropriate feature engineering.
Forecasting Horizon:	Can be adapted for both short and long-term forecasting.	Typically used for short to medium-term forecasting.	Suited for short to medium-term forecasting.	Can be used for both short and long-term forecasting.
Ease of Use:	Requires expertise in deep learning and parameter tuning.	More straightforward to implement with fewer hyperparameters to tune.	Designed for ease of use, suitable for users without extensive time series forecasting experience.	Requires parameter tuning, but can be implemented with standard machine learning practices.

Limitations:

LSTM: Long Short-Term Memory (LSTM) models faced challenges in cryptocurrency price forecasting, being highly data-dependent and computationally intensive. The models struggled to capture meaningful patterns in dynamic markets and reacted slowly to sudden changes, limiting their effectiveness. Additionally, hyperparameter tuning complexities and difficulties in handling noisy data further impacted their accuracy and practicality for real-time decision-making in cryptocurrency markets.

ARIMA: ARIMA models faced challenges in cryptocurrency price forecasting due to assumptions of linearity and specific statistical distributions, unsuitable for the non-linear and non-Gaussian nature of cryptocurrency prices. The requirement for stationary data conflicted with dynamic cryptocurrency trends, impacting adaptability. Sudden market changes and sensitivity to outliers further hindered ARIMA's effectiveness in capturing the unpredictability of cryptocurrency markets.

Facebook PROPHET: Prophet faced challenges in cryptocurrency forecasting, requiring intricate tuning of hyperparameters and struggling with its additive component assumption in the non-linear cryptocurrency market. Sensitivity to outliers, limitations in incorporating external variables, and assumptions about strict seasonality patterns raised concerns about its adaptability and reliability beyond the training data.

Support Vector Regression: Support Vector Regression (SVR) faced challenges in cryptocurrency price forecasting, requiring intricate tuning of hyperparameters. Its resource-intensive training, especially with large datasets, raised concerns for real-time applications. SVR's "black-box" nature hindered interpretability, and susceptibility to overfitting and difficulties in long-term forecasting emphasized the need for effective feature engineering for optimal results.

Conclusion:

Our comprehensive exploration of machine learning models for Cryptocurrency Price Forecasting underscores the superior performance of LSTM in predicting price trends. The graphical representation clearly demonstrates LSTM's ability to anticipate price movements ahead of actual market shifts. ARIMA showcases a commendable accuracy of 97.4%, solidifying its reliability in forecasting. While Facebook PROPHET provides interactive visualizations, its interpretability is not as easy considering the nature of the data but the MAPE's consistent accuracy within the range of 0 to 0.5 reinforces the robustness of our model. Conversely, Support Vector Regression proves unsuitable, with a negative accuracy of -0.128 and notable discrepancies between predicted and actual values, rendering it an unreliable choice for investment predictions. Therefore, among the various models considered, LSTM stands out as the most suitable choice.

Future Work

While the present study focused on basic machine learning model implementation, there are several avenues for future research and enhancements that can further enrich our understanding and support the development of more sophisticated strategies for project's findings, offering a more comprehensive and practical framework for cryptocurrency price forecasting and investment decision support.

- **Real-Time Prediction:** Implement a real-time prediction framework to assess the models' performance in predicting cryptocurrency prices as new data becomes available. This would provide practical insights for real-world investment decision-making.

- **Cross-Cryptocurrency Comparison:** Extend the comparative analysis to multiple cryptocurrencies to assess how well the models generalize across different digital assets. This could provide insights into the models' versatility and suitability for a broader range of cryptocurrency investments.

- **Incorporation of Additional Features:** Integrate additional relevant features, such as sentiment analysis from social media or macroeconomic indicators, to enrich the dataset. This could potentially enhance the models' ability to capture external factors influencing cryptocurrency prices.

References

Datasets

1. https://finance.yahoo.com/quote/BTC-USD/profile?p=BTC-USD

2. https://finance.yahoo.com/quote/ETH-USD/profile?p=ETH-USD

LSTM

3. https://www.mdpi.com/2504-3110/7/2/203

4. https://ieeexplore.ieee.org/document/10141048

ARIMA

5. https://www.analyticsvidhya.com/blog/2021/12/cryptocurrency-price-prediction-using-arima-model/

6. https://ieeexplore.ieee.org/document/9702842

Facebook PROPHET

7. https://medium.com/geekculture/bitcoin-price-prediction-using-facebook-prophet-b1b11c8dde2f

8. https://benthamscience.com/chapter/15670

Support Vector Regression

9. https://www.ijeast.com/papers/226-229,%20Tesma611,IJEAST.pdf

10. https://ieeexplore.ieee.org/document/10091183

About machine learning in cryptocurrency

11. https://www.sciencedirect.com/science/article/abs/pii/S0275531922001854#:~:text=The%20use%20of%20machine%20learning,topics%20in%20the%20cryptocurrency%20field.

12. https://jfin-swufe.springeropen.com/articles/10.1186/s40854-020-00217-x