BITCOIN PRICE PREDICTION

A PROJECT REPORT

for

MACHINE LEARNING (ITE2011)

in

B.Tech. – Information Technology and Engineering

by

KUSHAGRA AGARWAL (18BIT0231)

PRIYAL BHARDWAJ (18BIT0272)

ROHAN JAIN (18BIT0429)

Under the Guidance of

Dr. DURAI RAJ VINCENT P.M

Associate Professor, SITE

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

School of Information Technology and Engineering

November, 2020

Abstract:

Bitcoin is the first digital decentralized cryptocurrency that has shown a significant increase in market capitalization in recent years. Bitcoin has recently received a lot of attention from the media and the public due to its recent price surge and crash. Correspondingly, many researchers have investigated various factors that affect the Bitcoin price and the patterns behind its fluctuations, in particular, using various machine learning methods.

In this project, we explored several algorithms of machine learning using supervised learning to develop a prediction model and provide informative analysis of future market prices, studied and compared various state-of-the-art machine learning methods such as Arima, XGBoost, Facebook Prophet and a long short-term memory (LSTM) model for Bitcoin price prediction. Due to the difficulty of evaluating the exact nature of a Time Series (ARIMA) model, it is often very difficult to produce appropriate forecasts. Then we continue to implement Recurrent Neural Networks (RNN) with long short-term memory cells (LSTM). Thus, we analysed the time series model prediction of bitcoin prices with greater efficiency using long short-term memory (LSTM) technique.

Experimental results showed that XGBoost-based prediction models performed worse than the other prediction models for Bitcoin price prediction and LSTM-based models performed the best for price ups and downs prediction (regression).

Description of the Project:

We have compared and contrasted 4 machine learning models to determine which performs best for bitcoin price prediction:

1. **ARIMA**

   It is an acronym that stands for Auto Regressive Integrated Moving Average. It is a class of model that captures a suite of different standard temporal structures in time series data. The acronym is descriptive, capturing the key aspects of the model itself. Briefly, they are:

   AR: Autoregression A model that uses the dependent relationship between an observation and some number of lagged observations.

   I: Integrated The use of differencing of raw observations (e.g. subtracting an observation from an observation at the previous time step) in order to make the time series stationary.

   MA: Moving Average A model that uses the dependency between an observation and a residual error from a moving average model applied to lagged observations.

   A standard notation is used of ARIMA(p,d,q) where the parameters are substituted with integer values to quickly indicate the specific ARIMA model being used.

   p: The number of lag observations included in the model, also called the lag order.

   d: The number of times that the raw observations are differenced, also called the degree of differencing.

   q: The size of the moving average window, also called the order of moving average.

2. **Facebook Prophet**

   Prophet is a procedure for forecasting time series data based on an additive model where non-linear trends are fit with yearly, weekly, and daily seasonality, plus holiday effects.

   It works best with time series that have strong seasonal effects and several seasons of historical data. Prophet is robust to missing data and shifts in the trend, and typically handles outliers well.

3. **XGBoost**

   It is an efficient implementation of gradient boosting for classification and regression problems.

   It is both fast and efficient, performing well, if not the best, on a wide range of predictive modelling tasks and is a favourite among data science competition winners, such as those on Kaggle.

   XGBoost can also be used for time series forecasting, although it requires that the time series dataset be transformed into a supervised learning problem first. It also requires the use of a specialized technique for evaluating the model called walk-forward validation, as evaluating the model using k-fold cross validation would result in optimistically biased results.

4. **LSTM**

   Long Short-Term Memory networks – usually just called "LSTMs" – are a special kind of RNN, capable of learning long-term dependencies.

   LSTMs are explicitly designed to avoid the long-term dependency problem. Remembering information for long periods of time is practically their default behaviour, not something they struggle to learn like RNNs.

   All recurrent neural networks have the form of a chain of repeating modules of neural network. In standard RNNs, this repeating module will have a very simple structure, such as a single tanh layer. Also, they do not suffer from problems like vanishing/exploding gradient descent.

Literature Survey:

Journal 1: Bitcoin Price Prediction using Machine Learning

Authors: Siddhi Velankar, Sakshi Valecha, Shreya Maji

Date: 2018

Link: https://ieeexplore.ieee.org/document/8323676

In this paper, authors attempt to predict the Bitcoin price accurately taking into consideration various parameters that affect the Bitcoin value. For the first phase of their investigation, their purpose was to understand and identify daily trends in the Bitcoin market while gaining insight into optimal features surrounding Bitcoin price. The data set consists of various features relating to the Bitcoin price and payment network over the course of five years,

recorded daily. For the second phase of their investigation, they tried to predict the sign of the daily price change with highest possible accuracy.

The first step of the paper is database collection. They have acquired bitcoin values from two different databases which are - Quandl and CoinmarketCap. After acquiring this time-series data recorded daily for five years at different time instances, they normalized and smoothened the data. To achieve this, they have implemented different normalization techniques like Log normalization, Z-score normalization, Boxcox normalization, Standard deviation normalization. The next step is to select parameters that will be fed to the machine learning algorithm. After feature selection, the sample inputs were fed to the model. After that the number of layers and the number of neurons per layer were tuned to get better efficiency. Hence, the model performed a pattern aided regression algorithm and artificial neural networks to correctly predict the bitcoin value. The accuracy was compared with different models after the final prediction.

After trying out various machine learning algorithms and artificial neural networks, they shortlisted their proposed models to Bayesian Regression and Random Forest trees algorithm. They used the above two algorithms and chose the best method to solve the Bitcoin Prediction problem. However, the result of the two algorithms was not mentioned in the research paper.

Journal 2: Predicting the Price of Bitcoin Using Machine Learning

Authors: Sean McNally, Jason Roche, Simon Caton

Date: 2018

Link: https://ieeexplore.ieee.org/document/8374483

In this paper, authors have tried to determine the accuracy of prediction of Bitcoin price in USD and compare parallelisation methods executed on multi-core and GPU environments. The price data is sourced from the Bitcoin Price Index. This paper follows the CRISP data mining methodology. The motivation for CRISP over the more traditional KDD revolves around the business setting of the prediction task.

The Bitcoin dataset used, ranges from August 2013 until July 2016. The data was also standardised to give it a mean of 0 and standard deviation of 1. To evaluate which features to include, Boruta was used. It is an ensemble method in which classification is performed by voting of multiple classifiers. The prediction task is achieved through the implementation of a Bayesian optimised recurrent neural network (RNN) and a Long Short-Term Memory (LSTM) network. In order to support a comparison of the deep learning methods to more traditional methods they built an ARIMA model, as they have been extensively used in price prediction problems.

The LSTM achieves the highest classification accuracy of 52% and a RMSE of 8%. The non-linear deep learning methods outperform the ARIMA forecast which performs poorly. Finally,

both deep learning models are benchmarked on both a GPU and a CPU with the training time on the GPU outperforming the CPU implementation by 67.7%.

Journal 3: Machine Learning Models Comparison for Bitcoin Price Prediction

Authors: Thearasak Phaladisailoed, Thanisa Numnonda

Date: 2018

Link: https://ieeexplore.ieee.org/document/8534911

In this paper, authors aim to discover the most efficient and highest accuracy model to predict Bitcoin prices from various machine learning algorithms. By using 1-minute interval trading data on the Bitcoin exchange website named bitstamp some different regression models with scikit-learn and Keras libraries were experimented.

They have used Bitcoin transaction data from the bitstamp website with publishing on the Kaggle website. However, in this research, 1-minute interval trading exchange data rate in USD from January 1, 2012 to January 8, 2018 is focused. The datasets are in CSV files. In the dataset, a total of 8 features were present of which 4 were selected using the scikit-learn library when the Weighted price was to be predict. Then, they have divided dataset into a training set and a test set with the ratio of 70:30. Min-Max scaler is used to adjust data into 0 to 1 by using min and max of the data. In this research Theil-Sen Regression and Huber Regression were selected to compare. For deep learning-based regression models, Keras library was used to create Long-Short term Memory (LSTM) and Gated Recurrent Unit (GRU) models.

From the results of all the implemented algorithms, GRU shows the best accuracy but takes calculated time more than Huber regression. The results show that deep learning-based regression models: GRU and LSTM give the better result than Theil-Sen regression and Huber regression. GRU give the best results of MSE at 0.00002 and R2 at 0.992. Whereas, the calculated time that Huber regression use is much less than LSTM and GRU.

Journal 4: Systematic Erudition of Bitcoin Price Prediction using Machine Learning Techniques

Authors: Prachi Vivek Rane, Sudhir N. Dh age

Date: 2019

Link: https://ieeexplore.ieee.org/document/8728424

In this paper, the authors conduct an in-depth research on growth of Bitcoin and also an orderly review is done on various machine learning algorithms used for predicting the prices. Comparative analysis visualizes to select optimal technique to forecast prices more precisely. The main motive of the paper is to adapt volatility of prices and forecast the prices of Bitcoin with precision. The various machine learning algorithms used are:

- Auto-Regressive Integrated Moving Average Model (ARIMA model): It is parametric model that predicts individual time series. This model forecast time series data with uncertainty within short term period. The prediction values of ARIMA model failed poorly when parameters such as RMSE and accuracy is considered.

- Regression Model: Linear Regression model estimates the coefficient of values. Prediction is done by generating a regression model from linear and multiple regression techniques. Learned Linear model provides unsuitability to forecast the time series of Bitcoin price.
- Binomial Generalized Linear Model (BGLM): It analyses linear and non-linear effects on a discrete or continuous dependent variable. The aim of this model is to provide estimation of model parameters
- Generalized Autoregressive Conditional Heteroskedasticity Model (GARCH): It estimates volatility of prices. It transforms the standard Ordinary Least Squares residuals into an endogenous process that allows its variance to vary across periods. It yields precise forecast of variances and covariances of returns through modelling time-varying conditional variances.
- Support Vector Machine Model (SVM): It is a predictive model for classification and regression problem. It works by creating a higher-dimensional model which assigns each new data provided to one category or another. Decision is obtained from the verge that which maximizes the functional and geometric margins between classes.

Journal 5: Stochastic Neural Networks for Cryptocurrency Price Prediction

Authors: Patel Jay, Vasu Kalariya, Pushpendra Parmar, Sudeep Tanwar, Neeraj Kumar, Mamoun Alazab

Date: 27 April 2020

Link: https://ieeexplore.ieee.org/document/9079491

In this paper, the author discusses the failures and issues with usage of traditional currency and also introduces Blockchain for determining or predicting the price of cryptocurrency. Bitcoin is the most famous form of cryptocurrency. The value of the cryptocurrency is determined by the consistency and security of the platform on which it is deployed. Double-spending and transaction data manipulation are some of the problems faced when using traditional currency. The value of Cryptocurrencies is primarily affected by a large number of factors like social sentiment, legislature, past price trends, and trade volumes. A significant amount of research work to anticipate Cryptocurrency prices using machine learning techniques has already been explored in the last few years.

The paper describes a technique to predict the prices of prevalent cryptocurrencies, i.e., Bitcoin, Ethereum, and Litecoin which is designed using a stochastic neural network process. Stochastic neural networks are a type of artificial neural networks built by introducing random variations of data into the neural network, either by giving the network's neurons stochastic transfer functions, or by giving them stochastic weights. Next, they designed a mathematical function of stochastic layers in deep neural networks which would characterize the erratic behavior of the financial system. It is important to determine market's reaction to information regarding cryptocurrency so that improvements can be made in existing models for prediction prices of cryptocurrencies.

In their proposed model they used a total of 23 feature with the window side of 7. To predict the price of a cryptocurrency, they took three essential data sources, the first being market statistics. Market statistics include the day low/high and volume. Secondly, they used blockchain network information that included hash rate, transaction count, transaction fee, etc. Thirdly, they used social sentiment information like google trends and tweet volume.

The results show that the proposed hypothesis was not only valid but effective in decrypting market volatility. Almost all of the stochastic versions of the neural net models outperformed the deterministic versions.

In the future, they plan to explore territories in the cross-disciplinary field of stochastic processes and neural networks that can be exploited in cryptocurrency markets.

Journal 6: An Optimal Least Square Support Vector Machine Based Earnings Prediction of Blockchain Financial Products

Authors: M. Sivaram, E. Laxmi Lydia, Irina V. Pustokhina, Denis Alexandrovich Pustokhin, Mohamed Elhoseny, Gyanendra Prasad Joshi, K. Shankar

Date: 29 June 2020

Link: https://ieeexplore.ieee.org/document/9127981

In this paper, it is discussed how many developers use Machine Learning to compute and quantify the analysis of Blockchain financial products which includes Bitcoin. Blockchain is basically a data structure which records the transactions in a sequential manner, facilitated as a distributed set of records. With the application of Neural Network (NN) in forecasting the economic data, several authors have identified the potentials of optimal learning for nonlinear function. Besides, it employs NN in detection and is suitable for financial data like stock markets. Since the Support Vector Machine (SVM) method is accurate, it is capable of minimizing the over-fitting issue and many developers follow quantitative analysis.

In their research, they present a new return rate prediction model for Blockchain financial products based on Optimal Least Square Support Vector Machine (OLS-SVM) model. In OLS-SVM model, the parameters in LS-SVM are optimized using hybridization of Grey Wolf Optimization (GWO) algorithm with Differential Evolution (DE), called optimal GWO (OGWO) algorithm. A validation of the OLS-SVM model is done on Ethereum (ETH) return rate which is chosen as the target. The experimental analysis was performed to verify the predictive results on the time series. The application of DE in GWO algorithm eliminates the local optima problem, enhances the diversity of the population, and achieves proper balance between exploration as well as exploitation. Least Squares SVM (LS-SVM) is the extended model of standard SVM that converts a Quadratic Programming (QP) issue into linear equations. Both rapid solution speed and robust real-time function can be attained.

The authors selected ETH return rate as the target to examine the effectiveness of the OLS-SVM model. Through extensive experimentation, they observed that both ANN and GA-SVM

models attained significantly lower performance over OLS-SVM model. The attained experimental results portrayed that the OLS-SVM model could produce a superior predictive outcome for Blockchain financial products like Bitcoin.

In the future, they plan to improve the performance of the OLS-SVM model by using deep learning concepts.

Journal 7: Bitcoin price forecasting method based on CNN-LSTM hybrid neural network model

Authors: Yan Li, Wei Dai

Date: 24 August 2020

Link: https://ieeexplore.ieee.org/document/9175181

In this paper, the authors work on a Bitcoin price predicting method based on the popular convolutional neural network (CNN) and long short-term memory (LSTM) neural network. They aim to create a hybrid neural network, CNN-LSTM. CNN is used quite widely for feature extraction and LSTM is an artificial recurrent neural network used majorly in deep learning. In terms of forecasting, machine learning has made breakthrough progress in recent years.

The authors decided to use Baidu Index to accurately quantify investors' attention to Bitcoin. The dataset they used came from the Baidu website (https:// index.baidu.com/#/). The forecast period used by them was 3 days, the characteristic parameter data of the previous 3 days was used to predict the closing price of Bitcoin on the fourth day. Each set of samples in the constructed data set had a 51- dimensional feature that contained all the features of the previous 3 trading days. The first 588 samples were used as the training set for model training, and the last 20 samples were applied in the test set to verify the feasibility of the model. The CNN part in the hybrid model was mainly responsible for input of data and feature extraction. After performing two successful convolutions they were able to extract the characteristic data as a one-dimensional vector array of length 50.

All models were trained 30 times. Moreover, the predictions of the single structure neural network models were compared with the results of the proposed CNN-LSTM hybrid model. It was seen through various performance metrics that the CNN-LSTM model had the best performance in both value prediction and direction prediction.

For future works, the authors suggested incorporating the popularity of Bitcoin on other worldwide social media like Instagram and Twitter while analysing macroeconomic variables.

Journal 8: Bitcoin price prediction using Deep Learning Algorithm

Authors: Muhammad Rizwan, Sanam Narejo, Moazzam Javed

Date: 15 December 2019

Link: https://ieeexplore.ieee.org/document/9024772

Bitcoin's worth keeps varying or fluctuating like stocks in very unexpected ways. This is one main reason why a price predicting model for cryptocurrencies like Bitcoin is required. In this

paper, the authors test out deep learning models to achieve better performance when compared to previous Machine Learning and Blockchain models. A Bayesian recurrent hierarchical (RNN) neural network and a long-term memory (LSTM) network can accomplish this function. In contrast to the profound training systems, the common ARIMA method for the prediction of time series. This model is not much efficient as deep learning model can be performed. The deep learning methods were predicted to outperform the poorly performing ARIMA prediction.

For the dataset, external information was considered and obtained from international economic indicators. The authors took regression AI because of nonstop estimations of Bitcoin cost. With the scikit-learn library, the best two regression models. Recurrent Neural Network (RNN), Long Short-Term Memory (LSTM). For profound learning-based relapse models, Keras library was utilized to make LSTM and GRU models. The RNN and LSTM network quality were tested on validation information utilizing overfitting prevention measures. Drop-out was applied in two layers and, unless its validity failure in five epochs changes, they automatically stop template learning. LSTM was implemented to address the disappearance problem. The number of predictions was close to the actual value. RNN is ideal for sequence or time serial data, but it has a long-term dependence data problem that causes discontinuity. When predictions were done on a set or around 100 dates, the number of predictions close to the actual value was around 84 and they were as close as 90-100% to the actual values.

The proposed model had been compared with the existing literature including RNN, LSTM and ARIMA. Accordingly, it was observed that the proposed model outperformed the existing methods in terms of both accuracy and error rate.

Journal 9: Prediction of Bitcoin Price Change using Neural Networks

Authors: Rahmat Albariqi, Edi Winarko

Date: 2020

Link: https://sci-hub.st/https://ieeexplore.ieee.org/abstract/document/9079257/

In this research paper a baseline neural network models are used to predict the long term and short-term price of BITCOIN. The baseline models that are used for research are Multilayer Perceptron (MLP) and the Recurrent Neural Network (RNN). The dataset used is obtained from Bitcoin's Blockchain. The data used id from August 2010 to October 2017 with 2-days period. The total amount of data is 1300. The models that are generated are predicting price change for short-term and long-term from 2-days to 60 days.

The dataset is obtained from blockchain.info. There are two columns in the dataset, the first column is a timestamp and the second column is the feature value. For the time period considered i.e. from 02 August 2010 to 30 October 2017 the training data file is created using 14 csv files which has 1324 rows and 15 columns. 1 column is the time stamp while the other 14 are the features. There are 35 possible Blockchain features that can be used for the input of neural network, but for this paper only 14 of them are used. Block Size, cost per transaction,

Difficulty, Hash Rate, Market Capitalization, Median Confirmation Time, Miners Revenue, Number of Orphaned Blocks, Number of transaction, Number of transaction per Block, Unique Address, Bitcoins in Circulation, Trade Volume and transaction fees are used as an input for the neural network.

The conclusion drawn is the performance of MLP and RNN model for the prediction of change is price of Bitcoin is studied. Just like the Stock Market Price Prediction i.e. long term price prediction have higher accuracy it is observed that the cryptocurrency follows the same long term price prediction have higher accuracy. The long term price prediction achieves higher accuracy result than the shorter one both in MLP and RNN. The result of the price prediction for long term varies from 60-80%. The performance in this research paper is Multilayer Perceptron with time widow of 3 and 200 epochs has an accuracy of 81.3%, Precision of 81% and recall of 94.7%.

The future research will be combining the multiple features into a single feature to make the neural networks learn faster and better.

Journal 10: Comparative study of Bitcoin price prediction using WaveNets, Recurrent Neural Networks and other Machine Learning Methods

Authors: Leonardo Felizardo, Roberth Oliveira, Emilio Del-Moral-Hernandez, Fabio Cozman

Date: 2019

Link: https://sci-hub.st/https://ieeexplore.ieee.org/abstract/document/8963009/

In this research Paper a comparison is drawn between carious algorithm to predict the change in price of bitcoin. The different methodologies used to estimate the future price are ARIMA (Autoregressive Integrated Moving Average), Random Forest (RF), SVM (Support Vector Machine), LSTM (Long Short Term Memory) and Wavenets.

The data is extracted from a repository https://investing.com which is a base of financial time series. The data includes the bitcoin price, close, open, high and low, percentage change and the volume of transactions. This repository extracts data from Bitfinex using the API (Application Program Interface). The outliers are excluded from the data and it doesn't cause much effect due to the volume of the data. It has not considered the data from year 2017 and 2018 for analysis as is has many outliers and it undergoes great variance and random aspects. The data used is from the year 2012 – 2016. For better prediction results it has 4 prediction windows. D1(Prediction of price for the next day), D5(Prediction of price for 5 days in the future), D10(Prediction of price for 10 days in the future) and D30(Prediction of price for 30 days in the future).

For evaluate the system, error metrics between the models are compared and tested with the t-student test which is used to verify the null hypothesis. The error is evaluated using the error metrics. The metrics considered are: ME (Mean Error), MAE (Mean Absolute Error), RMSR (Root Mean Square Error), MPE (Mean Percentage Error) and MAPE (Mean Absolute Percentage Error). All the metrics are considered to compare all the results.

The conclusion drawn is all the methods perform worst which is indicated by the error metrics. The two models ARIMA and SVR tend to perform equally well in all the prediction gaps expect the D30. For long-term predictions all models tend perform equally badly, which indicates the random component gets more important and affect the accuracy.

Journal 11: Crypto-Currency price prediction using Decision Tree and Regression techniques

Authors: Karunya Rathan, Somarouthu Venkat Sai and Tubati Sai Manikanta

Date: 2019

Link: https://sci-hub.st/https://ieeexplore.ieee.org/abstract/document/8862585/

This research paper predicts the price of crypto currency. The crypto currency used in this research is Bitcoin. The aim of this research is to derive the accuracy of bitcoin prediction using different machine learning algorithms and compare the accuracy of those algorithms. The machine learning algorithm used are decision tree and regression model. The dataset used is the data of bitcoin from 2011-2019. It consists of open, high, low and close price details of the Bitcoin value. The dataset is obtained from quandl.com. The proposed work collected daa from quandl.com with name mentioned as BITSTAMPUSD. The data has the following features Time_stamp, high, low, open, close, volume_btc, volumne_currency, weighted_price.

The execution is performed in the following order Dataset is collected and the pre-processing is done. The pre-processing data is then split into Training set and testing set. The Decision tree and linear regression model is applied on the data. Then we train the model and give the test set to predict the values. Then we obtained the result for the price for the 5 days. The 80% data is considered as training data which trains the machine learning models (decision tree and the linear regression) and remaining 20% data is considered as test data which is used for result prediction.

The result is efficient for the bitcoin price prediction is the linear regression. The accuracy for the linear regression is 97.5% while the accuracy of the decision tree is 95.8%.
The conclusion from the research is that the price forecast for 5 days is done using the decision tree and linear regression models. The proposed learning method from the research suggest the best algorithms to choose and adopt for crypto currency prediction problem. The result by this study shows that linear regression outperforms the decision tree by high accuracy on price prediction.

Journal 12: Exploring the Influence of News Articles on Bitcoin Price with Machine Learning

Authors: Wenbing Yao, Ke Xu, Qi Li

Date: 2019

Link:

https://www.researchgate.net/publication/333995009_Exploring_the_Influence_of_News_Articles_on_Bitcoin_Price_with_Machine_Learning

In this research paper the aim to research the effects of news article on bitcoin prices. The extract features that are used from news articles are both commonly used text features extraction algorithms and SentiGraph. The paper focuses on the relationship between news article and the bitcoin price fluctuation, the experiment is performed on several machine learning algorithms using four text representation methods which includes a novel one also, the text representation method is a general framework.

The data of the news article is collected from 10 websites which concentrate on the area of cryptocurrency. All these articles are written in English. 108+ news articles are collected from March 2012 to February 2019. Each news article has title, release date and content. The news article without the exact release date are dropped. All the news articles cannot be used directly for prediction so they are transformed to machine friendly data formats using the text feature extraction methods. Also the original data has lot of noise i.e. non-English characters and pictures so they are removed as they are useless information. The price data of bitcoin is collected from coindesk.com. It has the close price detail of bitcoin of each hour.

The conclusion from this paper is a study is done on the influence of daily news on the price of bitcoin. A comparison is made on the performance of four features extraction methods by training four machine learning methods. The novel text representation method i.e. SentiGraph achieves the highest accuracy with 59%. This method's performance is better than other traditional method i.e. N-Gram, TF-IDF and Doc2vec. The results conclude that the news article have an impact on the price of bitcoin.

Complete Code and Complete Analysis:

https://www.kaggle.com/kushagra26/bitcoin-price-prediction#Model-Comparison

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from datetime import datetime
import seaborn as sns
import plotly.express as px
from itertools import product
import warnings
import statsmodels.api as sm
plt.style.use('seaborn-darkgrid')
```

Reading the dataset

```
bitstamp = pd.read_csv("/kaggle/input/bitcoin-historical-data/bitstampUSD_1-min_data_2012-01-01_to_2020-09-14.csv")
bitstamp.head()
```

	Timestamp	Open	High	Low	Close	Volume_(BTC)	Volume_(Currency)	Weighted_Price
0	1325317920	4.39	4.39	4.39	4.39	0.455581	2.0	4.39
1	1325317980	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	1325318040	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	1325318100	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	1325318160	NaN	NaN	NaN	NaN	NaN	NaN	NaN

```
print('Dataset Shape: ',  bitstamp.shape)
```

```
Dataset Shape:  (4572257, 8)
```

Function to convert timestamp into the required form

```
bitstamp['Timestamp'] = [datetime.fromtimestamp(x) for x in bitstamp['Timestamp']]
```

```
bitstamp.head()
```

	Timestamp	Open	High	Low	Close	Volume_(BTC)	Volume_(Currency)	Weighted_Price
0	2011-12-31 07:52:00	4.39	4.39	4.39	4.39	0.455581	2.0	4.39
1	2011-12-31 07:53:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	2011-12-31 07:54:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	2011-12-31 07:55:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	2011-12-31 07:56:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Data Preprocessing

```
#calculating missing values in the dataset
missing_values = bitstamp.isnull().sum()
missing_per = (missing_values/bitstamp.shape[0])*100
missing_table = pd.concat([missing_values,missing_per], axis=1, ignore_index=True)
missing_table.rename(columns={0:'Total Missing Values',1:'Missing %'}, inplace=True)
```

```
#testing missing value methods on a subset

a = bitstamp.set_index('Timestamp')
a = a['2019-11-01 00:10:00':'2019-11-02 00:10:00']

a['ffill'] = a['Weighted_Price'].fillna(method='ffill') # Imputation using ffill/pad
a['bfill'] = a['Weighted_Price'].fillna(method='bfill') # Imputation using bfill/pad
a['interp'] = a['Weighted_Price'].interpolate()         # Imputation using interpolation
```

```
# function to impute missing values using interpolation
def fill_missing(df):
    df['Open'] = df['Open'].interpolate()
    df['Close'] = df['Close'].interpolate()
    df['High'] = df['High'].interpolate()
    df['Low'] = df['Low'].interpolate()
    df['Weighted_Price'] = df['Weighted_Price'].interpolate()
    df['Volume_(BTC)'] = df['Volume_(BTC)'].interpolate()
    df['Volume_(Currency)'] = df['Volume_(Currency)'].interpolate()


    print(df.head())
    print("\n")
    print(df.isnull().sum())
```

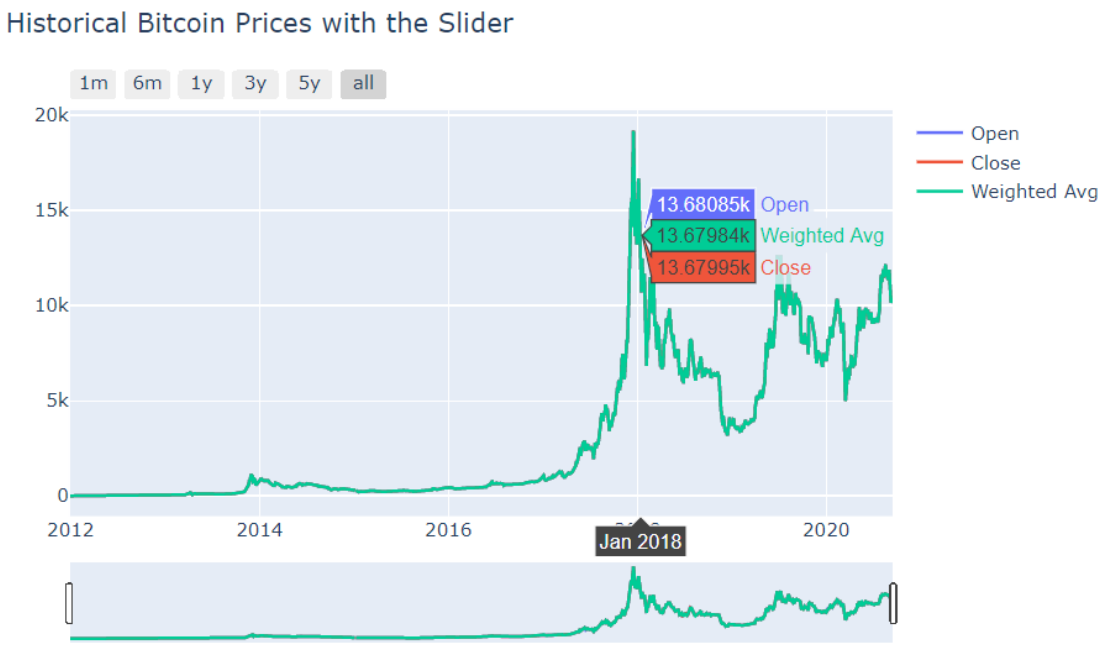
Data Visualisation

```
#daily resampling
bitstamp_daily = bitstamp.resample("24H").mean()
bitstamp_daily.head()
```

	Open	High	Low	Close	Volume_(BTC)	Volume_(Currency)	Weighted_Price
Timestamp							
2011-12-31	4.476415	4.478946	4.476415	4.478946	17.940426	79.495594	4.477370
2012-01-01	4.765576	4.765576	4.765576	4.765576	6.790640	32.971105	4.765576
2012-01-02	5.006549	5.006549	5.006549	5.006549	15.183373	75.932706	5.006549
2012-01-03	5.206530	5.206530	5.206530	5.206530	7.917041	40.795994	5.206530
2012-01-04	5.202511	5.241699	5.202511	5.241699	13.659736	72.860096	5.216680

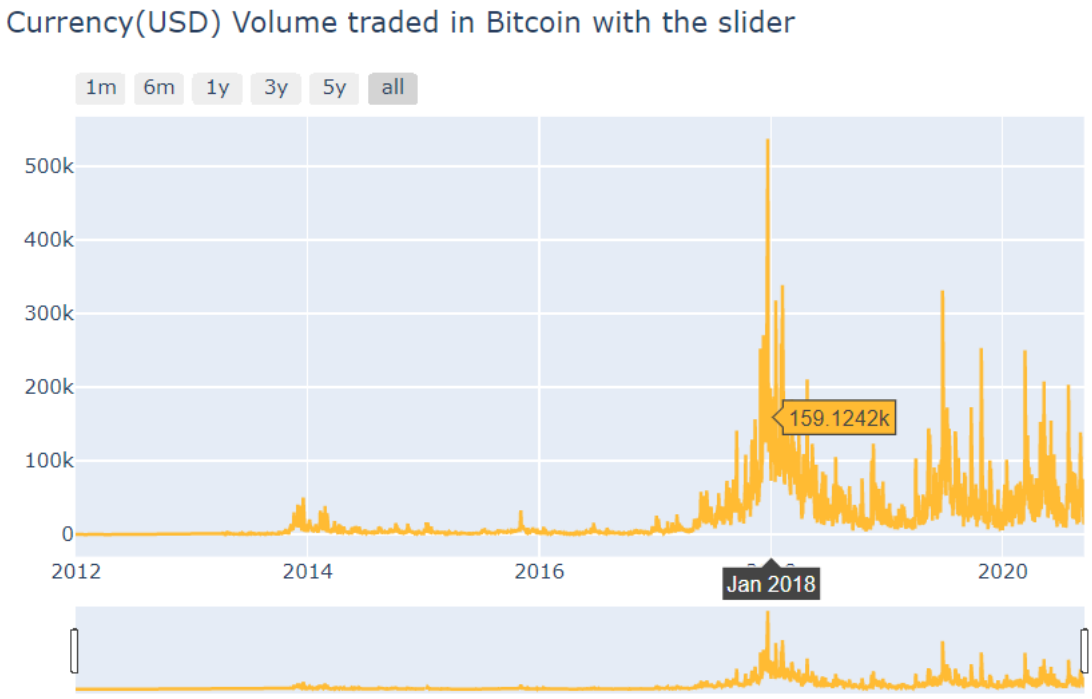
```
bitstamp_daily.reset_index(inplace=True)

trace1 = go.Scatter(
    x = bitstamp_daily['Timestamp'],
    y = bitstamp_daily['Open'].astype(float),
    mode = 'lines',
    name = 'Open'
)

trace2 = go.Scatter(
    x = bitstamp_daily['Timestamp'],
    y = bitstamp_daily['Close'].astype(float),
    mode = 'lines',
    name = 'Close'
)
trace3 = go.Scatter(
    x = bitstamp_daily['Timestamp'],
    y = bitstamp_daily['Weighted_Price'].astype(float),
    mode = 'lines',
    name = 'Weighted Avg'
)

layout = dict(
```

```
    title='Historical Bitcoin Prices with the Slider ',
    xaxis=dict(
        rangeselector=dict(
            buttons=list([
                dict(count=1,
                     label='1m',
                     step='month',
                     stepmode='backward'),
                dict(count=6,
                     label='6m',
                     step='month',
                     stepmode='backward'),
                dict(count=12,
                     label='1y',
                     step='month',
                     stepmode='backward'),
                dict(count=36,
                     label='3y',
                     step='month',
                     stepmode='backward'),
                dict(count=60,
                     label='5y',
                     step='month',
                     stepmode='backward'),
                dict(step='all')
            ])
        ),
        rangeslider=dict(
            visible = True
        ),
        type='date'
    )
)

data = [trace1,trace2,trace3]
fig = dict(data=data, layout=layout)
iplot(fig, filename = "Time Series with Rangeslider")
```




```
trace1 = go.Scatter(
    x = bitstamp_daily['Timestamp'],
    y = bitstamp_daily['Volume_(Currency)'].astype(float),
    mode = 'lines',
    name = 'Currency',
    marker = dict(
            color='#FFBB33')
)

layout = dict(
    title='Currency(USD) Volume traded in Bitcoin with the slider',
    xaxis=dict(
        rangeselector=dict(
            buttons=list([
                dict(count=1,
                     label='1m',
                     step='month',
                     stepmode='backward'),
                dict(count=6,
                     label='6m',
                     step='month',
                     stepmode='backward'),
                dict(count=12,
                     label='1y',
                     step='month',
                     stepmode='backward'),
                dict(count=36,
                     label='3y',
                     step='month',
                     stepmode='backward'),
                dict(count=60,
                     label='5y',
                     step='month',
                     stepmode='backward'),
                dict(step='all')
            ])
        ),
```

```
        rangeslider=dict(
            visible = True
        ),
        type='date'
    )
)

data = [trace1]
fig = dict(data=data, layout=layout)
iplot(fig, filename = "Time Series with Rangeslider")
```

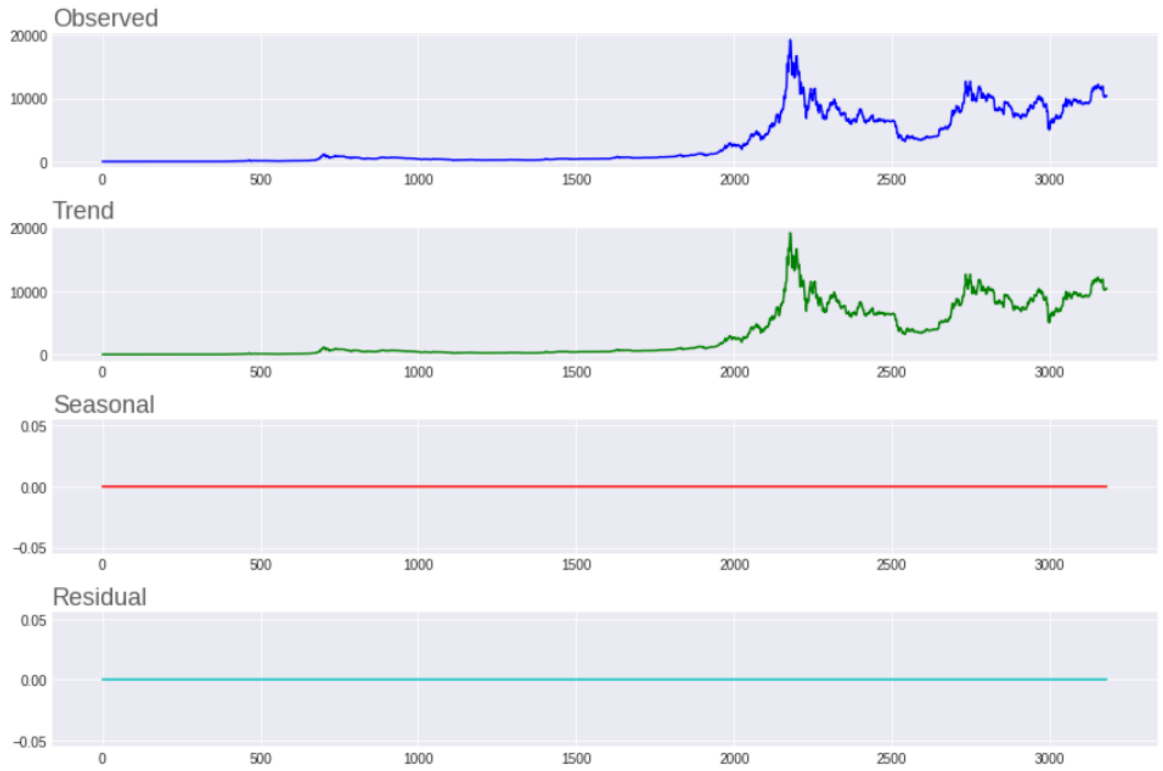


Time Series Decomposition & Statistical Tests

We can decompose a time series into trend, seasonal and remainder components. The seasonal_decompose in statsmodels is used to implements the decomposition.

```
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.stattools import kpss
from statsmodels.tsa.stattools import adfuller
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
```

```
decomposition = sm.tsa.seasonal_decompose(bitstamp_daily.Weighted_Price,period=1)

trend = decomposition.trend
seasonal = decomposition.seasonal
residual = decomposition.resid

ax, fig = plt.subplots(figsize=(12,8), sharex=True)

plt.subplot(411)
plt.plot(bitstamp_daily.Weighted_Price, label='Original',color='b')
plt.title("Observed",loc="left", alpha=0.75, fontsize=18)

plt.subplot(412)
plt.plot(trend, label='Trend',color='g')
plt.title("Trend",loc="left", alpha=0.75, fontsize=18)

plt.subplot(413)
plt.plot(seasonal,label='Seasonality',color='r')
plt.title("Seasonal",loc="left", alpha=0.75, fontsize=18)

plt.subplot(414)
plt.plot(residual, label='Residuals',color='c')
plt.title("Residual",loc="left", alpha=0.75, fontsize=18)
plt.tight_layout()
```



Rolling Windows

A rolling mean, or moving average, is a transformation method which helps average out noise from data. It works by simply splitting and aggregating the data into windows according to function, such as mean(), median(), count(), etc. For this dataset, we'll use a rolling mean for 3, 7 and 30 days.

```
df.reset_index(drop=False, inplace=True)

lag_features = ["Open", "High", "Low", "Close","Volume_(BTC)"]
window1 = 3
window2 = 7
window3 = 30

df_rolled_3d = df[lag_features].rolling(window=window1, min_periods=0)
df_rolled_7d = df[lag_features].rolling(window=window2, min_periods=0)
df_rolled_30d = df[lag_features].rolling(window=window3, min_periods=0)

df_mean_3d = df_rolled_3d.mean().shift(1).reset_index()
df_mean_7d = df_rolled_7d.mean().shift(1).reset_index()
df_mean_30d = df_rolled_30d.mean().shift(1).reset_index()

df_std_3d = df_rolled_3d.std().shift(1).reset_index()
df_std_7d = df_rolled_7d.std().shift(1).reset_index()
df_std_30d = df_rolled_30d.std().shift(1).reset_index()

for feature in lag_features:
    df[f"{feature}_mean_lag{window1}"] = df_mean_3d[feature]
    df[f"{feature}_mean_lag{window2}"] = df_mean_7d[feature]
    df[f"{feature}_mean_lag{window3}"] = df_mean_30d[feature]

    df[f"{feature}_std_lag{window1}"] = df_std_3d[feature]
    df[f"{feature}_std_lag{window2}"] = df_std_7d[feature]
    df[f"{feature}_std_lag{window3}"] = df_std_30d[feature]

df.fillna(df.mean(), inplace=True)

df.set_index("Timestamp", drop=False, inplace=True)

df["month"] = df.Timestamp.dt.month
df["week"] = df.Timestamp.dt.week
df["day"] = df.Timestamp.dt.day
df["day_of_week"] = df.Timestamp.dt.dayofweek
df.head()
```

```
df.shape
```

```
(3181, 42)
```

Model Building:

```
df_train = df[df.Timestamp < "2020"]
df_valid = df[df.Timestamp >= "2020"]

print('train shape :', df_train.shape)
print('validation shape :', df_valid.shape)
```

```
train shape : (2923, 42)
validation shape : (258, 42)
```

1) ARIMA Model

```
import pmdarima as pm
```

```
exogenous_features = ['Open_mean_lag3',
       'Open_mean_lag7', 'Open_mean_lag30', 'Open_std_lag3', 'Open_std_lag7',
       'Open_std_lag30', 'High_mean_lag3', 'High_mean_lag7', 'High_mean_lag30',
       'High_std_lag3', 'High_std_lag7', 'High_std_lag30', 'Low_mean_lag3',
       'Low_mean_lag7', 'Low_mean_lag30', 'Low_std_lag3', 'Low_std_lag7',
       'Low_std_lag30', 'Close_mean_lag3', 'Close_mean_lag7',
       'Close_mean_lag30', 'Close_std_lag3', 'Close_std_lag7',
       'Close_std_lag30', 'Volume_(BTC)_mean_lag3', 'Volume_(BTC)_mean_lag7',
       'Volume_(BTC)_mean_lag30', 'Volume_(BTC)_std_lag3',
       'Volume_(BTC)_std_lag7', 'Volume_(BTC)_std_lag30', 'month', 'week',
       'day', 'day_of_week']
```

```
model = pm.auto_arima(df_train.Weighted_Price, exogenous=df_train[exogenous_features], tra
ce=True,
                      error_action="ignore", suppress_warnings=True)
model.fit(df_train.Weighted_Price, exogenous=df_train[exogenous_features])

forecast = model.predict(n_periods=len(df_valid), exogenous=df_valid[exogenous_features])
df_valid["Forecast_ARIMAX"] = forecast
```

```
Performing stepwise search to minimize aic
 ARIMA(2,0,2)(0,0,0)[0] intercept   : AIC=38878.107, Time=26.65 sec
 ARIMA(0,0,0)(0,0,0)[0] intercept   : AIC=40085.697, Time=4.79 sec
 ARIMA(1,0,0)(0,0,0)[0] intercept   : AIC=39088.717, Time=15.99 sec
 ARIMA(0,0,1)(0,0,0)[0] intercept   : AIC=39256.270, Time=18.83 sec
 ARIMA(0,0,0)(0,0,0)[0]             : AIC=40100.132, Time=23.31 sec
 ARIMA(1,0,2)(0,0,0)[0] intercept   : AIC=38867.881, Time=26.38 sec
 ARIMA(0,0,2)(0,0,0)[0] intercept   : AIC=38903.986, Time=25.28 sec
 ARIMA(1,0,1)(0,0,0)[0] intercept   : AIC=39045.050, Time=23.19 sec
 ARIMA(1,0,3)(0,0,0)[0] intercept   : AIC=38873.369, Time=8.95 sec
 ARIMA(0,0,3)(0,0,0)[0] intercept   : AIC=38871.981, Time=28.14 sec
 ARIMA(2,0,1)(0,0,0)[0] intercept   : AIC=39030.913, Time=24.33 sec
 ARIMA(2,0,3)(0,0,0)[0] intercept   : AIC=38874.077, Time=30.46 sec
 ARIMA(1,0,2)(0,0,0)[0]             : AIC=38865.939, Time=25.21 sec
 ARIMA(0,0,2)(0,0,0)[0]             : AIC=38902.277, Time=24.15 sec
 ARIMA(1,0,1)(0,0,0)[0]             : AIC=39043.386, Time=20.15 sec
 ARIMA(2,0,2)(0,0,0)[0]             : AIC=38876.859, Time=27.18 sec
 ARIMA(1,0,3)(0,0,0)[0]             : AIC=38871.502, Time=8.39 sec
 ARIMA(0,0,1)(0,0,0)[0]             : AIC=39254.443, Time=15.52 sec
 ARIMA(2,0,3)(0,0,0)[0]             : AIC=38870.986, Time=30.60 sec

Best model:  ARIMA(1,0,2)(0,0,0)[0]
Total fit time: 455.838 seconds
```

```
df_valid[["Weighted_Price", "Forecast_ARIMAX"]].plot(figsize=(14, 7))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f71b5879090>
```

```
from sklearn.metrics import mean_squared_error, mean_absolute_error

print("RMSE of Auto ARIMAX:", np.sqrt(mean_squared_error(df_valid.Weighted_Price, df_vali
d.Forecast_ARIMAX)))
print("\nMAE of Auto ARIMAX:", mean_absolute_error(df_valid.Weighted_Price, df_valid.Forec
ast_ARIMAX))
```

```
RMSE of Auto ARIMAX: 322.1896817251843

MAE of Auto ARIMAX: 227.00036001821348
```

2) Facebook Prophet

```
from fbprophet import Prophet
```

```
# Resampling originial data to day level and forward fill the missing values
daily_data = bitstamp.resample("24H").mean()
fill_missing(daily_data)
```

```
# Renaming the column names accroding to Prophet's requirements
daily_data_fb = daily_data.reset_index()[['Timestamp','Weighted_Price']].rename({'Timestam
p':'ds','Weighted_Price':'y'}, axis=1)
daily_data_fb.head()
```

	ds	y
0	2011-12-31	4.477370
1	2012-01-01	4.765576
2	2012-01-02	5.006549
3	2012-01-03	5.206530
4	2012-01-04	5.216680

```
split_date = "2020-01-01"
train_filt = daily_data_fb['ds'] <= split_date
test_filt = daily_data_fb['ds'] > split_date

train_fb = daily_data_fb[train_filt]
test_fb = daily_data_fb[test_filt]
```

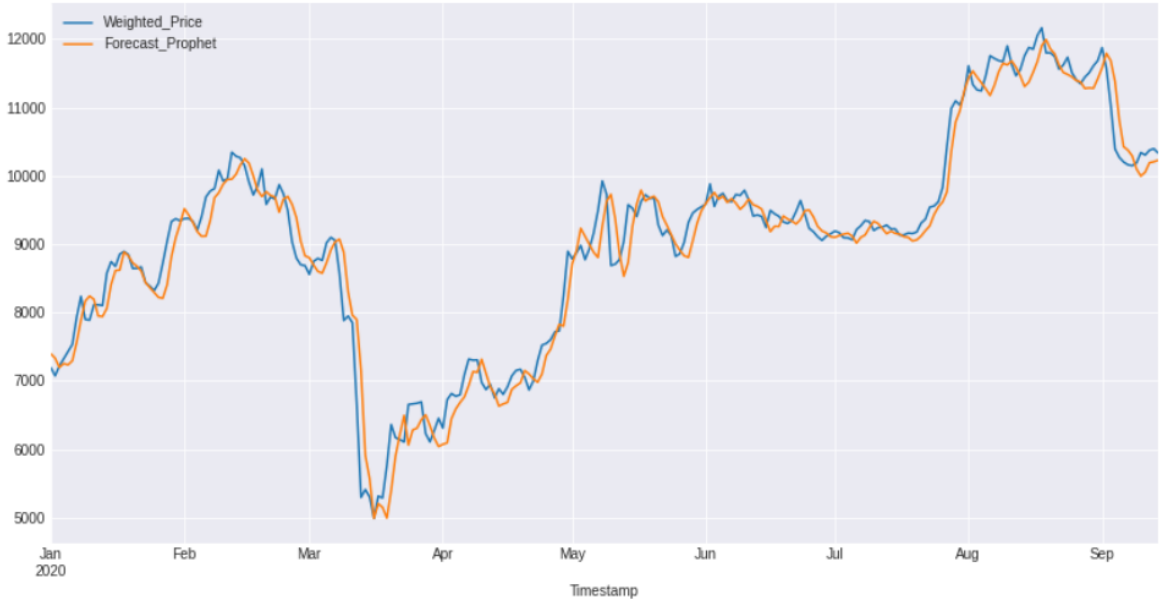
```
model_fbp = Prophet()
for feature in exogenous_features:
    model_fbp.add_regressor(feature)

model_fbp.fit(df_train[["Timestamp", "Weighted_Price"] + exogenous_features].rename(column
s={"Timestamp": "ds", "Weighted_Price": "y"}))

forecast = model_fbp.predict(df_valid[["Timestamp", "Weighted_Price"] + exogenous_feature
s].rename(columns={"Timestamp": "ds"}))
forecast.head()
```

	ds	trend	yhat_lower	yhat_upper	trend_lower	trend_upper	Close_mean_lag3	Close_mean
0	2020-01-01	2676.830342	7069.509097	7698.975493	2676.830342	2676.830342	6237.527006	6237.5270
1	2020-01-02	2676.897316	7050.597294	7656.684851	2676.897316	2676.897316	6171.967757	6171.9677
2	2020-01-03	2676.964289	6890.600285	7490.007098	2676.964289	2676.964289	6073.072798	6073.0727
3	2020-01-04	2677.031263	6964.556121	7552.208327	2677.031263	2677.031263	6081.350401	6081.3504
4	2020-01-05	2677.098237	6955.615874	7545.693521	2677.098237	2677.098237	6136.674697	6136.6746

```
# Plot Our Predictions
fig1 = model_fbp.plot(forecast)
```

```
df_valid[["Weighted_Price", "Forecast_Prophet"]].plot(figsize=(14, 7))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f7110e74fd0>
```




```
test_mae = mean_absolute_error(df_valid['Weighted_Price'], df_valid['Forecast_Prophet'])
test_rmse = np.sqrt(mean_squared_error(df_valid['Weighted_Price'], df_valid['Forecast_Prophet']))

print(f"Prophet's MAE : {test_mae}")
print(f"Prophet's RMSE : {test_rmse}")
```

```
Prophet's MAE : 233.0184261175854
Prophet's RMSE : 327.348083280574
```

3) XGBoost Model

```
from sklearn import ensemble
from sklearn import metrics
from sklearn.model_selection import RandomizedSearchCV
import xgboost as xgb
from xgboost import plot_importance, plot_tree
from sklearn.metrics import mean_squared_error, mean_absolute_error
plt.style.use('fivethirtyeight')

from datetime import datetime
```

```
#Train Test Split
X_train, y_train = df_train[exogenous_features], df_train.Weighted_Price
X_test, y_test = df_valid[exogenous_features], df_valid.Weighted_Price
```

```
reg = xgb.XGBRegressor()
```

We have tuned our model using Hyperparameter tuning.

```
params={
 "learning_rate"    : [0.10,0.20,0.30],
 "max_depth"        : [1, 3, 4, 5, 6, 7],
 "n_estimators"     : [int(x) for x in np.linspace(start=100, stop=1000, num=10)],
 "min_child_weight" : [int(x) for x in np.arange(3, 10, 1)],
 "gamma"            : [0.0, 0.2 , 0.4, 0.6],
 "subsample"        : [0.5, 0.6, 0.7, 0.8, 0.9, 1],
 "colsample_bytree" : [0.5, 0.7, 0.9, 1],
 "colsample_bylevel": [0.5, 0.7, 0.9, 1],
}
```

```
model  = RandomizedSearchCV(
                reg,
                param_distributions=params,
                n_iter=20,
                n_jobs=-1,
                cv=5,
                verbose=3,
                )
```

```
print(f"Model Best Parameters : {model.best_params_}")
```

```
Model Best Parameters : {'subsample': 0.9, 'n_estimators': 1000, 'min_child_weight': 4,
'max_depth': 1, 'learning_rate': 0.2, 'gamma': 0.6, 'colsample_bytree': 0.9, 'colsample
_bylevel': 0.7}
```

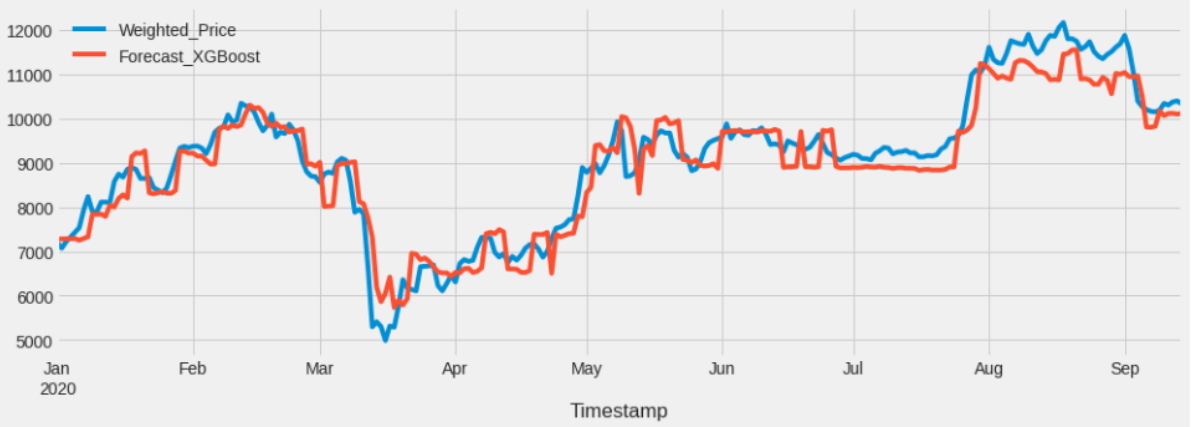
```
model.best_estimator_
```

```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=0.7,
             colsample_bynode=1, colsample_bytree=0.9, gamma=0.6, gpu_id=-1,
             importance_type='gain', interaction_constraints='',
             learning_rate=0.2, max_delta_step=0, max_depth=1,
             min_child_weight=4, missing=nan, monotone_constraints='()',
             n_estimators=1000, n_jobs=0, num_parallel_tree=1, random_state=0,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=0.9,
             tree_method='exact', validate_parameters=1, verbosity=None)
```

```
df_valid['Forecast_XGBoost'] = model.predict(X_test)

overall_data = pd.concat([df_train, df_valid], sort=False)
```

```
df_valid[['Weighted_Price','Forecast_XGBoost']].plot(figsize=(15, 5))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f710c344090>
```




```
train_mae = mean_absolute_error(df_train['Weighted_Price'], df_train['Predicted_Weighted_Price'])
train_rmse = np.sqrt(mean_squared_error(df_train['Weighted_Price'], df_train['Predicted_Weighted_Price']))

print(f"train MAE : {train_mae}")
print(f"train RMSE : {train_rmse}")
```

```
train MAE : 98.33401369443165
train RMSE : 191.17002897491432
```

```
test_mae = mean_absolute_error(df_valid['Weighted_Price'], df_valid['Forecast_XGBoost'])
test_rmse = np.sqrt(mean_squared_error(df_valid['Weighted_Price'], df_valid['Forecast_XGBoost']))

print(f"test MAE : {test_mae}")
print(f"test RMSE : {test_rmse}")
```

```
test MAE : 386.8104183740535
test RMSE : 482.2268967243938
```

4) LSTM

```
price_series = bitstamp_daily.reset_index().Weighted_Price.values
price_series
```

```
array([4.47737025e+00, 4.76557639e+00, 5.00654859e+00, ...,
       1.03731430e+04, 1.03957982e+04, 1.03324294e+04])
```

```
price_series.shape
```

```
(3181,)
```

```
# Feature Scaling
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler(feature_range = (0, 1))
price_series_scaled = scaler.fit_transform(price_series.reshape(-1,1))
```

```
train_data, test_data = price_series_scaled[0:2923], price_series_scaled[2923:]
```

```
train_data.shape, test_data.shape
```

```
((2923, 1), (258, 1))
```

```
def windowed_dataset(series, time_step):
    dataX, dataY = [], []
    for i in range(len(series)- time_step-1):
        a = series[i : (i+time_step), 0]
        dataX.append(a)
        dataY.append(series[i+ time_step, 0])

    return np.array(dataX), np.array(dataY)
```

```
X_train, y_train = windowed_dataset(train_data, time_step=100)
X_test, y_test = windowed_dataset(test_data, time_step=100)
```

```
X_train.shape, y_train.shape, X_test.shape, y_test.shape
```

```
((2822, 100), (2822,), (157, 100), (157,))
```

```
#reshape inputs to be [samples, timesteps, features] which is requred for LSTM

X_train = X_train.reshape(X_train.shape[0], X_train.shape[1], 1)
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], 1)

print(X_train.shape)
print(X_test.shape)
```

```
(2822, 100, 1)
(157, 100, 1)
```

```
#Create LSTM Model
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import LSTM
from tensorflow.keras.layers import Dropout
```

```
# Initialising the LSTM
regressor = Sequential()

# Adding the first LSTM layer and some Dropout regularisation
regressor.add(LSTM(units = 50, return_sequences = True, input_shape = (X_train.shape[1],
1)))
regressor.add(Dropout(0.5))

# Adding a second LSTM layer and some Dropout regularisation
regressor.add(LSTM(units = 50, return_sequences = True))
regressor.add(Dropout(0.5))

# Adding a third LSTM layer and some Dropout regularisation
regressor.add(LSTM(units = 50, return_sequences = True,))
regressor.add(Dropout(0.5))

# Adding a fourth LSTM layer and some Dropout regularisation
regressor.add(LSTM(units = 50))
regressor.add(Dropout(0.5))

# Adding the output layer
regressor.add(Dense(units = 1))

# Compiling the model
regressor.compile(optimizer = 'adam', loss = 'mean_squared_error')
```

```
regressor.summary()
```

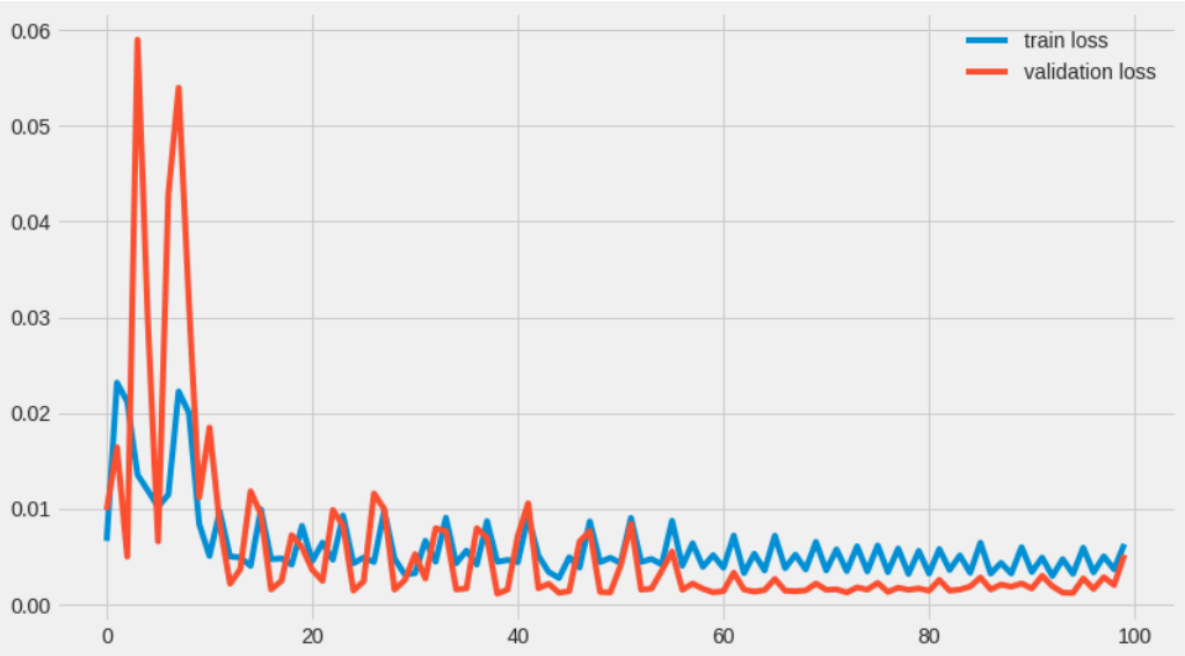
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 100, 50)           10400
_________________________________________________________________
dropout (Dropout)            (None, 100, 50)           0
_________________________________________________________________
lstm_1 (LSTM)                (None, 100, 50)           20200
_________________________________________________________________
dropout_1 (Dropout)          (None, 100, 50)           0
_________________________________________________________________
lstm_2 (LSTM)                (None, 100, 50)           20200
_________________________________________________________________
dropout_2 (Dropout)          (None, 100, 50)           0
_________________________________________________________________
lstm_3 (LSTM)                (None, 50)                20200
_________________________________________________________________
dropout_3 (Dropout)          (None, 50)                0
_________________________________________________________________
dense (Dense)                (None, 1)                 51
=================================================================
Total params: 71,051
Trainable params: 71,051
Non-trainable params: 0
```

```
# Fitting the LSTM to the Training set
history = regressor.fit(X_train, y_train, validation_split=0.2, epochs = 100, batch_size =
32, verbose=1, shuffle=False)
```

```
Epoch 1/100
71/71 [==============================] - 3s 41ms/step - loss: 0.0066 - val_loss: 0.0099
Epoch 2/100
71/71 [==============================] - 2s 23ms/step - loss: 0.0232 - val_loss: 0.0164
Epoch 3/100
71/71 [==============================] - 2s 24ms/step - loss: 0.0212 - val_loss: 0.0050
Epoch 4/100
71/71 [==============================] - 2s 21ms/step - loss: 0.0136 - val_loss: 0.0590
Epoch 5/100
71/71 [==============================] - 2s 22ms/step - loss: 0.0119 - val_loss: 0.0297
Epoch 6/100
71/71 [==============================] - 2s 21ms/step - loss: 0.0102 - val_loss: 0.0066
Epoch 7/100
71/71 [==============================] - 2s 21ms/step - loss: 0.0115 - val_loss: 0.0428
Epoch 8/100
71/71 [==============================] - 2s 21ms/step - loss: 0.0223 - val_loss: 0.0540
Epoch 9/100
71/71 [==============================] - 2s 24ms/step - loss: 0.0200 - val_loss: 0.0315
```

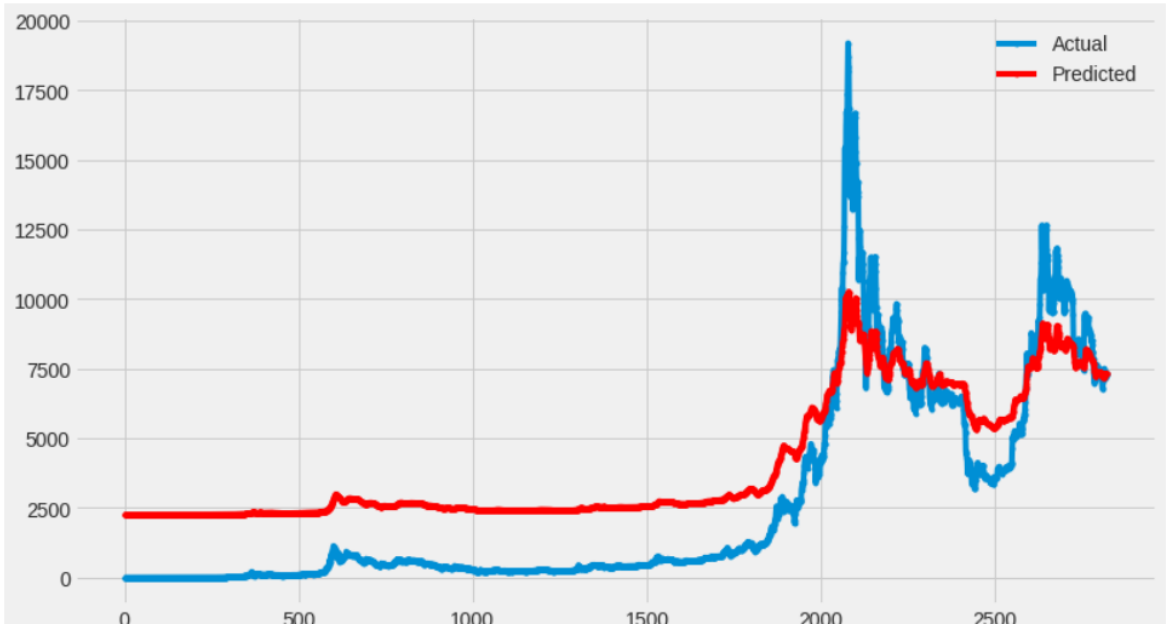
```
plt.figure(figsize=(12,7))
plt.plot(history.history["loss"], label= "train loss")
plt.plot(history.history["val_loss"], label= "validation loss")
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7f70885e4ed0>
```




```
plt.figure(figsize=(12,7))
plt.plot(y_train_inv.flatten(), marker='.', label="Actual")
plt.plot(train_predict_inv.flatten(), 'r', marker='.', label="Predicted")
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7f708862af10>
```



```
train_RMSE = np.sqrt(mean_squared_error(y_train, train_predict))
train_MAE = np.sqrt(mean_absolute_error(y_train, train_predict))
LSTM_RMSE = np.sqrt(mean_squared_error(y_test, test_predict))
LSTM_MAE = np.sqrt(mean_absolute_error(y_test, test_predict))


print(f"Train RMSE: {train_RMSE}")
print(f"Train MAE: {train_MAE}")

print(f"Test RMSE: {LSTM_RMSE}")
print(f"Test MAE: {LSTM_MAE}")
```

```
Train RMSE: 0.10646585356714015
Train MAE: 0.31343995505940286
Test RMSE: 0.09016122848323625
Test MAE: 0.2819190834604146
```

Outcome of project:

We have executed 4 different algorithms – **ARIMA model, Facebook Prophet, XGBoost model, and LSTM model** – on our Bitcoin Price Prediction Dataset. From the literature survey, we found out that **LSTM worked best** with time series datasets. So, we applied that model on our Bitcoin dataset as well and found the similar results as mentioned in the research papers. We have compared our models on the basis of error score. We have used 2 metrics for comparisons, Mean Absolute Error **(MAE)** and Root Mean Square Error **(RMSE)**. For the error metrics, lower the error value, better is the accuracy of model.

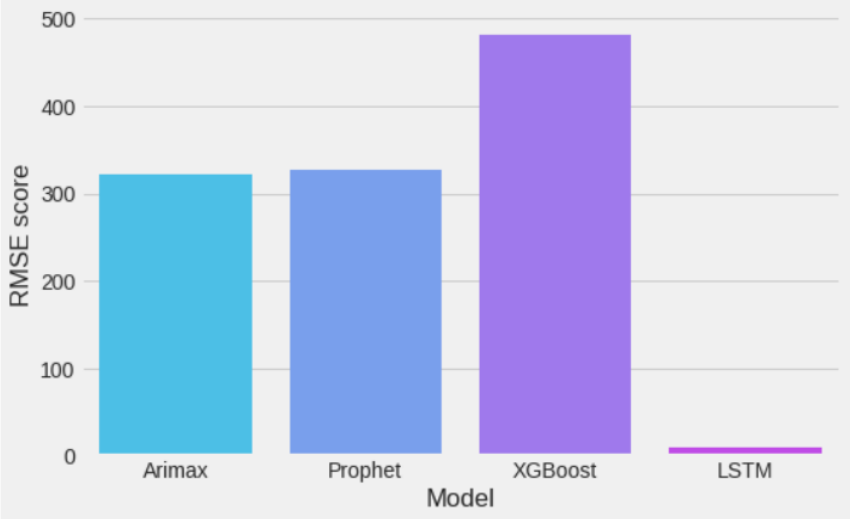
LSTM gave lowest MAE and RMSE values and that too by a large scale when compared to other three models.

```
print("ARIMAX RMSE :", arimax_rmse)
print("FB Prophet RMSE :", fbp_rmse)
print("XGBoost RMSE :", xgb_rmse)
print("LSTM RMSE :", LSTM_RMSE)

print("\nARIMAX MAE :", arimax_mae)
print("FB Prophet MAE :", fbp_mae)
print("XGBoost MAE :", xgb_mae)
print("LSTM MAE :", LSTM_MAE)
```

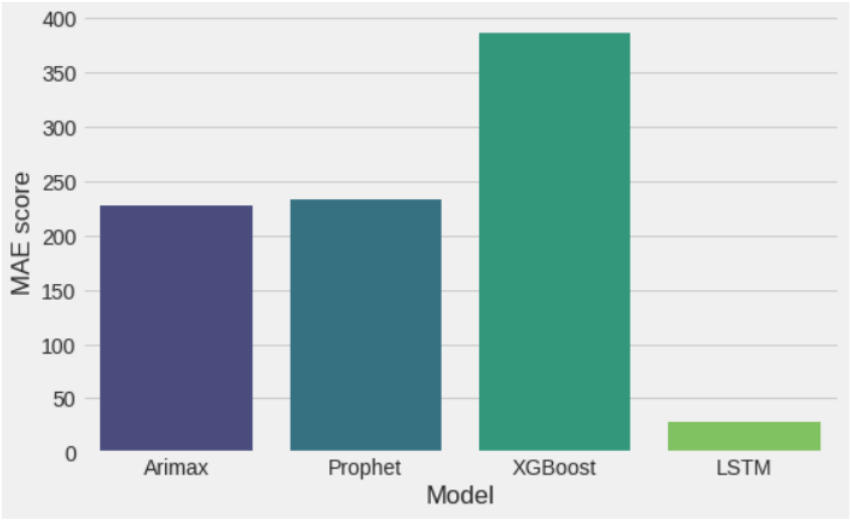
```
ARIMAX RMSE : 322.1896817251843
FB Prophet RMSE : 327.348083280574
XGBoost RMSE : 482.2268967243938
LSTM RMSE : 0.09016122848323625

ARIMAX MAE : 227.00036001821348
FB Prophet MAE : 233.0184261175854
XGBoost MAE : 386.8104183740535
LSTM MAE : 0.2819190834604146
```

As we can see from the above scores, error values of LSTM model are less than 1 whereas the error metrics of the other three models are in hundreds.

```
plt.figure(figsize = (8,5))
X = ['Arimax','Prophet','XGBoost','LSTM']
Y = [arimax_rmse,fbp_rmse,xgb_rmse,LSTM_RMSE*100]
ax = sns.barplot(x=X,y=Y,palette='cool')
ax.set(xlabel ='Model',ylabel ='RMSE score')
plt.show()
print('*Note: We have multiplied Rmse score of lstm model by 100 so that it can be visualised')
```




```
*Note: We have multiplied Rmse score of lstm model by 100 so that it can be visualised
```




```
*Note: We have multiplied MAE score of lstm model by 100 so that it can be visualised
```

XGBoost proved to be the worst model for the time series analysis. As an ensemble classifier, we thought it might perform better on the dataset however, due to shifting mean and median

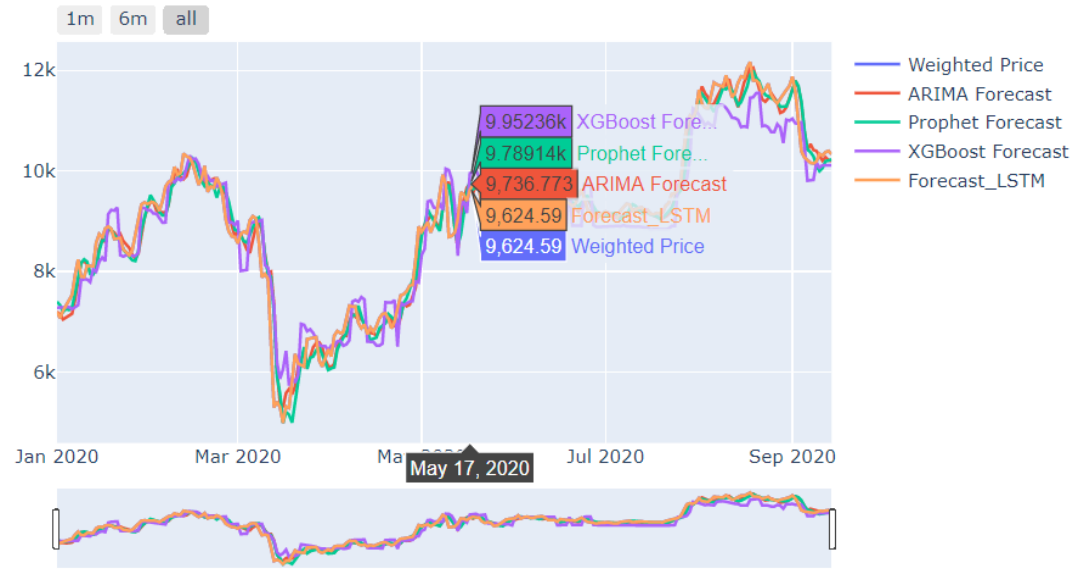
in the time series data, it wasn't able to predict data efficiently. Thus, we have also used Machine learning models which work better on time series analysis for comparison. Arima and Prophet are developed for time series analysis and as we can observe from RMSE graph, they have performed more efficiently when compared to XGBoost model but weren't as efficient as LSTM model which outperformed every other model.

We have also plotted an overall graph where all the models are plotted with the actual values and we can visualise the accuracy of each model.

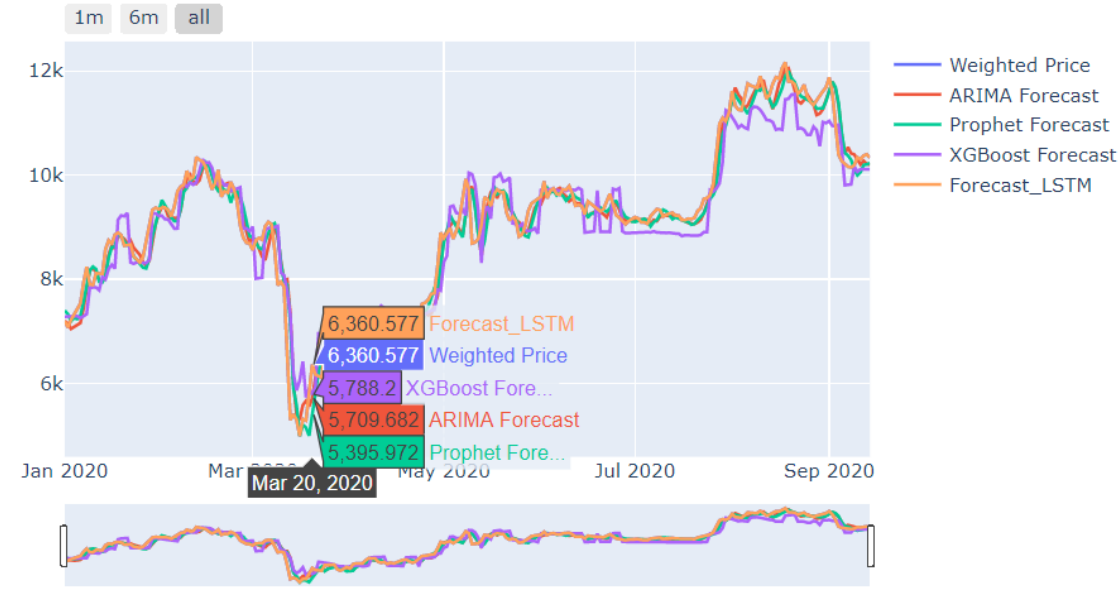
```
trace1 = go.Scatter(
    x = df_valid['Timestamp'],
    y = df_valid['Weighted_Price'],
    mode = 'lines',
    name = 'Weighted Price'
)

trace2 = go.Scatter(
    x = df_valid['Timestamp'],
    y = df_valid['Forecast_ARIMAX'],
    mode = 'lines',
    name = 'ARIMA Forecast'
)
trace3 = go.Scatter(
    x = df_valid['Timestamp'],
    y = df_valid['Forecast_Prophet'],
    mode = 'lines',
    name = 'Prophet Forecast'
)
trace4 = go.Scatter(
    x = df_valid['Timestamp'],
    y = df_valid['Forecast_XGBoost'],
    mode = 'lines',
    name = 'XGBoost Forecast'
)
trace5 = go.Scatter(
    x = df_valid['Timestamp'],
    y = df_valid['Weighted_Price'],
    mode = 'lines',
    name = 'Forecast_LSTM'
)


layout = dict(
    title='Model Comparison ',
    xaxis=dict(
        rangeselector=dict(
            buttons=list([
                dict(count=1,
                     label='1m',
                     step='month',
                     stepmode='backward'),
                dict(count=6,
```

```
                    label='6m',
                    step='month',
                    stepmode='backward'),
                dict(step='all')
            ])
        ),
        rangeslider=dict(
            visible = True
        ),
        type='date'
    )
)

data = [trace1,trace2,trace3,trace4,trace5]
fig = dict(data=data, layout=layout)
iplot(fig, filename = "Time Series with Rangeslider")
```

Model Comparison



Model Comparison



Conclusion and Future Work:

Bitcoin system is unique from any other asset on the financial market and thereby creates new possibilities. Our project can help as an exploratory beginning for several techniques, descriptive analysis of Bitcoin prices. As the Bitcoin network is huge, small as well as large transactions are carried out in this network which results in volatility of prices which needs to be maintained. The results show survey of techniques that used and also the technique which suits best for the prediction. Machine learning models which are created for time series analysis such as ARIMA and Prophet gave decent predictions which were close to actual value of Bitcoin. However, Deep Learning model LSTM was found to be the best amongst as all the models with a very low error metrics score. Overall, with our prediction's investors can have a fair idea about when the price will go up or when will it go down so that they can invest accordingly.

The analysis conducted in this work can be further extended by more research on upcoming advance methods. Hence more thorough picture of forecasting prices can be obtained. We can train the model on a larger and more accurate dataset to increase prediction accuracy. We can also try to compare our models with the other Deep Learning models since in our case, deep learning model gave the best result so we can try other models such as GRU for comparison. Having a well-rounded approach towards prediction is important thus further study is required to find other promising features. The dataset can be broken up onto sequential patterns and a linear regression model to be used on the patterns of data to predict the results, or use K-means clustering to group the data points. These grouped data points can be then used with a deep learning model.

****