BITCOIN PRICE PREDICTION

A PROJECT REPORT

for

DATA MINING TECHNIQUES (ITE2006)

in

B.Tech – Information Technology and Engineering

by

PRIYAL BHARDWAJ (18BIT0272)

Under the Guidance of

Dr. SENTHILKUMAR N C

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

June, 2021

DECLARATION BY THE CANDIDATE

I hereby declare that the project report entitled **"BITCOIN PRICE PREDICTION"** submitted by me to Vellore Institute of Technology University, Vellore in partial fulfillment of the requirement for the award of the course **Data Mining Techniques (ITE2006)** is a record of bonafide project work carried out by us under the guidance of **Dr. Senthilkumar N C.** I further declare that the work reported in this project has not been submitted and will not be submitted, either in part or in full, for the award of any other course.

Place : Vellore

Signature

Date : 2021-05-20

Priyal Bhardwaj

Digitally signed by Priyal Bhardwaj
Date: 2021.05.20 18:10:21 +05'30'

VIT®

Vellore Institute of Technology

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

School of Information Technology & Engineering [SITE]

CERTIFICATE

This is to certify that the project report entitled **"BITCOIN PRICE PREDICTION"** submitted by **Priyal Bhardwaj (18BIT0272)** to Vellore Institute of Technology University, Vellore in partial fulfillment of the requirement for the award of the course **Data Mining Techniques (ITE2006)** is a record of bonafide work carried out by them under my guidance.

Dr. Senthilkumar N C
GUIDE
Associate Professor, SITE

Bitcoin Price Prediction

Abstract

Data Mining is an advanced approach which refers to the extraction of previously unknown and useful information from large databases. Prediction in data mining refers to predicting the identity of one thing based purely on the description of another, related thing. 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. For the scope of this project, I have created a real time prediction model that will provide informative analysis of future market prices using state-of-the-art machine learning algorithms like ARIMA and LSTM.

Keywords – Prediction, Bitcoin, Machine Learning, ARIMA, LSTM, RMSE

I. INTRODUCTION

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, I 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. Thus, I analyzed the time series model prediction of bitcoin prices with greater efficiency using long short-term memory (LSTM) technique.

II. BACKGROUND

The existing systems use time series analysis which comprises methods for analyzing time series data in order to extract meaningful statistics and other characteristics of the data. Time series forecasting is the use of a model to predict future values based on previously observed values. ARIMA which stands for auto-regressive integrated moving average is a way of modelling time series data for forecasting (i.e., for predicting future points in the series). ARIMA models aim to describe the autocorrelations in the data. ARIMA models, as all forecasting methods, are essentially “backward looking”.

The long term forecast eventually goes to be straight line and poor at predicting series with turning points. Due to the difficulty of evaluating the exact nature of a Time Series (ARIMA) model, it is often very difficult to produce appropriate forecasts.

III. Literature Survey

Numerous prediction algorithms have been discussed and proposed in time forecasting domain for bitcoin price.

[1]. 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. 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 model performed a pattern aided regression algorithm and artificial neural networks to correctly predict the bitcoin value. After trying out various machine learning algorithms and artificial neural networks, they shortlisted their proposed models to Bayesian Regression and Random Forest trees algorithm.

[2]. 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 Ire experimented. In this research Theil-Sen Regression and Huber Regression Ire 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.

[3]. 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 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.

[4]. 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 prediction values of ARIMA model failed poorly when parameters such as RMSE and accuracy is considered. Learned Linear model provided unsuitability to forecast the time series of Bitcoin price.

[5]. Bitcoin's worth keeps varying or fluctuating like stocks in very unexpected ways. In this paper, the authors test out deep learning models to achieve better performance when compared to previous Machine Learning and Blockchain models. The authors took regression AI because of nonstop estimations of Bitcoin cost. 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.

[6]. 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) and Wavenets. 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 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 Ill in all the prediction gaps expect the D30.

[7]. 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. The authors decided to use Baidu Index to accurately quantify investors' attention to Bitcoin. The CNN part in the hybrid model was mainly responsible for input of data and feature extraction. After performing two successful convolutions they Ire able to extract the characteristic data as a one-dimensional vector array of length 50. All models Ire trained 30 times. It was seen through various performance metrics that the CNN-LSTM model had the best performance in both value prediction and direction prediction.

[8]. 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. 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. Next, they designed a mathematical function of stochastic layers in deep neural networks which would characterize the erratic behaviour of the financial system. In their proposed model they used a total of 23 feature with the window side of 7. 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.

[9]. 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 models that are generated are predicting price change for short-term and long-term from 2-days to 60 days. The conclusion drawn is the performance of MLP and RNN model for the prediction of change is price of Bitcoin is studied. 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.

[10]. 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. 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 attained experimental results portrayed that the OLS-SVM model could produce a superior predictive outcome for Blockchain financial products like Bitcoin.

[11]. 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 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 result by this study shows that linear regression outperforms the decision tree by high accuracy on price prediction.

[12]. 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. 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. The novel text representation method 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.

[13]. In this paper, authors compare various parameters that affect bitcoin price prediction based on Root Mean Square Error (RMSE) using various deep learning models. They used dataset from Poloniex which records the bitcoin price on an interval of 5 minutes. Convolutional Neural Network (CNN), Long Short Term Memory (LSTM) and Gated Recurrent Unit (GRU) deep learning models Ire compared. The Root Mean Square Error (RMSE) of LSTM model was found to be the least. So, it was concluded that LSTM is more capable of predicting long-term dependencies as compared to CNN and GRU.

[14]. In this paper, authors aim to create model prediction bitcoin stock market prediction using LSTM. There's a need automation tool for prediction to help investors decide for bitcoin or other cryptocurrency market investment. The dataset is collected from yahoo finance stock market based on the USD Exchange rate. The results indicated that when RMSE value of LSTM model was least the model predicted price accurately. It was concluded that the proposed model with time series techniques can build produce the results and the results can predict the price for the next days.

[15]. In this paper, authors compare various machine learning models for bitcoin price prediction. The price of Bitcoin varies over time and is governed by various factors, like the market it is being traded in, scarcity, supply and demand. Recurrent Neural Network (RNN), Long Short Term Memory (LSTM) and Linear Regression (LR) models Ire compared. The dataset used contains minute by minute prices of Bitcoin of over 5 years and contains almost

30,00,000 entries. It was concluded that RNN model with LSTM is better for forecasting and prediction of Bitcoin prices than LR model because of its capability to recognize longer term dependencies.

[16]. In this paper, authors investigate the predictive power of network sentiments and explore statistical and deep-learning methods to predict Bitcoin future price. Financial and sentiment features extracted from economic and crowd-sourced data respectively Ire analyzed. The authors try to show that sentiment is the most significant factor in predicting Bitcoin market stocks. Two models used for Bitcoin time-series predictions: Auto-Regressive Integrated Moving Average with exogenous input (ARIMAX) and the Recurrent Neural Network (RNN) Ire compared. It was concluded that both models achieved optimal results on new predictions, with a MSE value lower than 0.14%, due to the inclusion of the studied sentiment feature.

[17]. In this paper, authors try to recognize the pattern of change in the data time series in a certain period of time bitcoin price can be known several days ahead with a high degree of accuracy. The unstable price of Bitcoin can be anticipated by making a prediction or forecasting the price of Bitcoin in an upcoming time period. The Data used in this study was obtained from the site www.coingecko.com. They chose ARIMA method because it was able to predict large time series data and was able to produce good short-term predictions. It was concluded that the ARIMA (4,1,4) model is can be used as a predictive method of Bitcoin for one to seven days ahead.

[18]. In this paper, the authors try to establish correlation among Bitcoin price and Twitter and Google search patterns. Bitcoin being a decentralized and peer-to-peer cryptocurrency has attracted a large number of users on the Ib search and social media. And sentiments of the text used in emails, blogs, and social media posts affect human decision making and behaviour. They analysed models like Linear regression, polynomial regression, Recurrent Neural Network, and Long Short-Term Memory for the same. It was concluded that there is a relevant degree of correlation of Google Trends and Tweet volume data with the price of Bitcoin, and with no significant relation with the sentiments of tweets.

[19]. In this paper, authors compare the ARIMA model with the BPNN model in the prediction of Bitcoin price. The ARIMA model is mainly used to analyse and predict the statistical model of smooth time series data. The variable data from Yahoo Finance and Investing.com Ire statistically standardized. In this study, the free statistical programming software R was adopted to establish the BPNN. It was concluded that I should adopt the ARIMA model and BPNN for prediction, in order to increase investment return and establish comprehensive risk evasion strategies.

[20]. In this paper, authors aim at improving the prediction accuracy of cryptocurrencies' prices with a focus on Bitcoin using timeseries data charts. They use a more advanced architecture, ResNet, and implement stochastic gradient descent with restarts, and cyclical learning rate selection. The importance of using a visual representation of data in providing better prediction of those hidden patterns in bitcoin trends using deep learning with a focus on CNN is shown. After training on the correct classes, the model achieved the highest accuracy with, 78.60% on the Coinbase test set. They concluded that CNNs can be adopted to recognize subtle and undetectable patterns within images of timeseries data charts.

[21]. In this paper, authors introduce a real-time and adaptive cryptocurrency price prediction platform based on Twitter sentiments. It is based on a Spark-based architecture which handles the large volume of incoming data in a persistent and fault tolerant way. It combined an approach that supports sentiment analysis which can respond to large amounts of natural language processing queries in real time; and a predictive method grounded on online learning in which a model adapts its weights to cope with new prices and sentiments. The paper also describes the KryptoOracle platform implementation and experimental evaluation. They concluded that KryptoOracle is able to correctly predict the bitcoin price ahead of 1 minute time. The engine clearly learned from the errors it made and rewired itself to predict in real-time.

[22]. In this paper, the authors combined long short-term memory (LSTM), with singular spectrum analysis (SSA) to predict Bitcoin price. SSA was employed to decompose the original time series into independent signals in term of trend, market fluctuation and noise. Next, they introduced the smoothed series sequence into LSTM, to obtain prediction value. Empirical analysis shoId that the proposed hybrid SSA-LSTM model outperformed baseline single LSTM model, according to root mean square error (RMSE), mean absolute error (MAE) and mean absolute percentage error (MAPE). They concluded that the proposed hybrid model is better able to grasp pattern of Bitcoin price series since SSA can extract valid information from the original series and avoid overfitting.

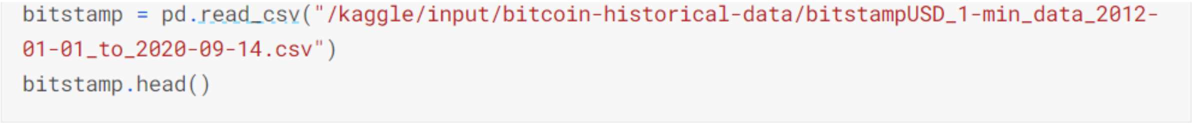
[23]. In this paper, authors propose various time scales and the associated price prediction technique for Bitcoin price forecasting. They propose a three LSTM-based neural network models are designed for differentiated time scales such as short-term (i.e., minute), mid-term (i.e., hour), and long-term (i.e., day) data sets in order to reflect Bitcoin price's considerable fluctuation. The dataset they selected, market data had samples with three-time intervals, ranging from December 1, 2014 (2014-12-01) to November 11, 2018 (2018-11-11). The prediction of the rise in price had high error, while the prediction of the down-falling point showed a reasonable performance. Finally, it was concluded that the proposed united model aggregated the LSTM networks with ensemble techniques and predicted the Bitcoin price with high accuracy while demonstrating best performance especially in risky periods.

[24]. One of the main goals in the Bitcoin analytics is price forecasting. In this paper, authors consider an approach for building regression predictive model for bitcoin price using expert correction by adding a correction term. They used the linear model for Bitcoin price which includes regression features based on Bitcoin currency statistics, mining processes, Google search trends and Wikipedia pages visits. The pattern of deviation of regression model prediction from real prices is simpler comparing to price time series. They concluded that with Bayesian inference, one can utilize the probabilistic approach using distributions with fat tails and take into account outliers in Bitcoin price time series.

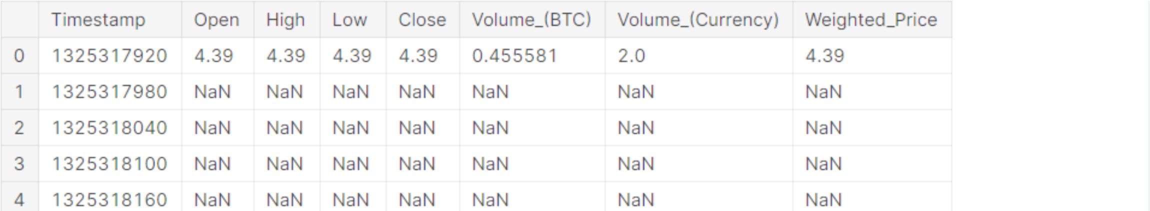
[25]. In this paper, authors attempted to answer two research questions. First, when comparing the Simple RNN, and GRU machine learning models, does one model predict the cryptocurrency price better than others? Second, will adding Google trends data improve the prediction power? They conducted experiments on three RNN models, with, and without Google Trend data, on three cryptocurrencies: Bitcoin, Ripple, and Litecoin. They collected daily trading data of three cryptocurrencies: Bitcoin (BTC), Litecoin (LTC), and Ripple (XRP), from coinmarketcap.com and Google trends data corresponding to each cryptocurrency was collected as Ill. They concluded that Google trends data is not a beneficial data input for RNN models when trying to predict the price of a cryptocurrency.

IV. DATASET DESCRIPTION & SAMPLE DATA

Dataset provided on Kaggle website: Bitcoin Historical Data. The data set consists bitcoin exchanges for the time period of Jan 2012 to December 2020, with minute-to-minute updates of OHLC (Open, High, Low, Close), volume in BTC and indicated currency, and weighted bitcoin price.

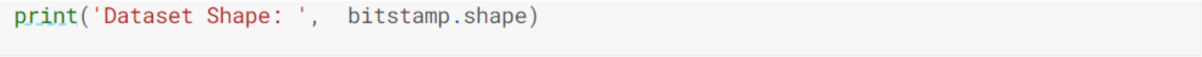


```
bitstamp = pd.read_csv("/kaggle/input/bitcoin-historical-data/bitstampUSD_1-min_data_2012-
01-01_to_2020-09-14.csv")
bitstamp.head()
```

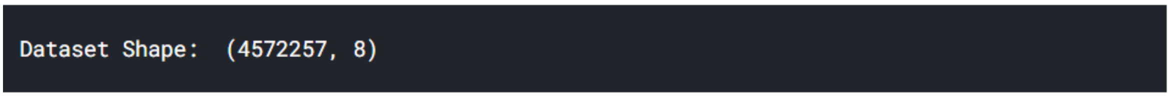


	Timestamp	Open	High	Low	Close	Volume_(BTC)	Volume_(Currency)	Weighted_Price
0	1325317920	4.39	4.39	4.39	4.39	0.455581	2.0	4.39
1	1325317980	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	1325318040	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	1325318100	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	1325318160	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Importing the required libraries. Importing the dataset (stored locally). After importing we check the shape of the dataset

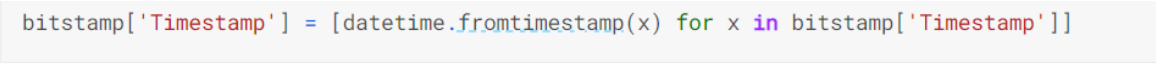


```
print('Dataset Shape: ',  bitstamp.shape)
```

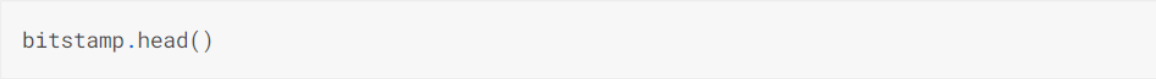


```
Dataset Shape:  (4572257, 8)
```

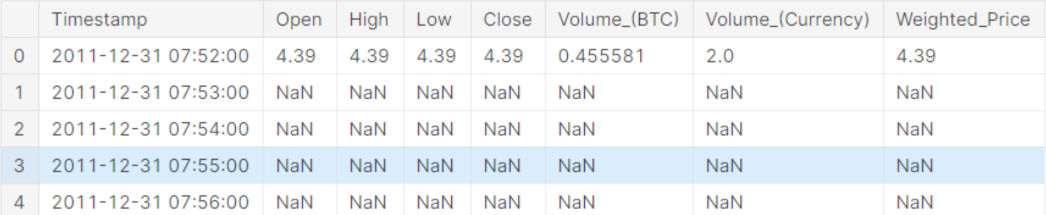
Function to convert timestamp into the required form and then viewing the first 5 rows of the dataframe.



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



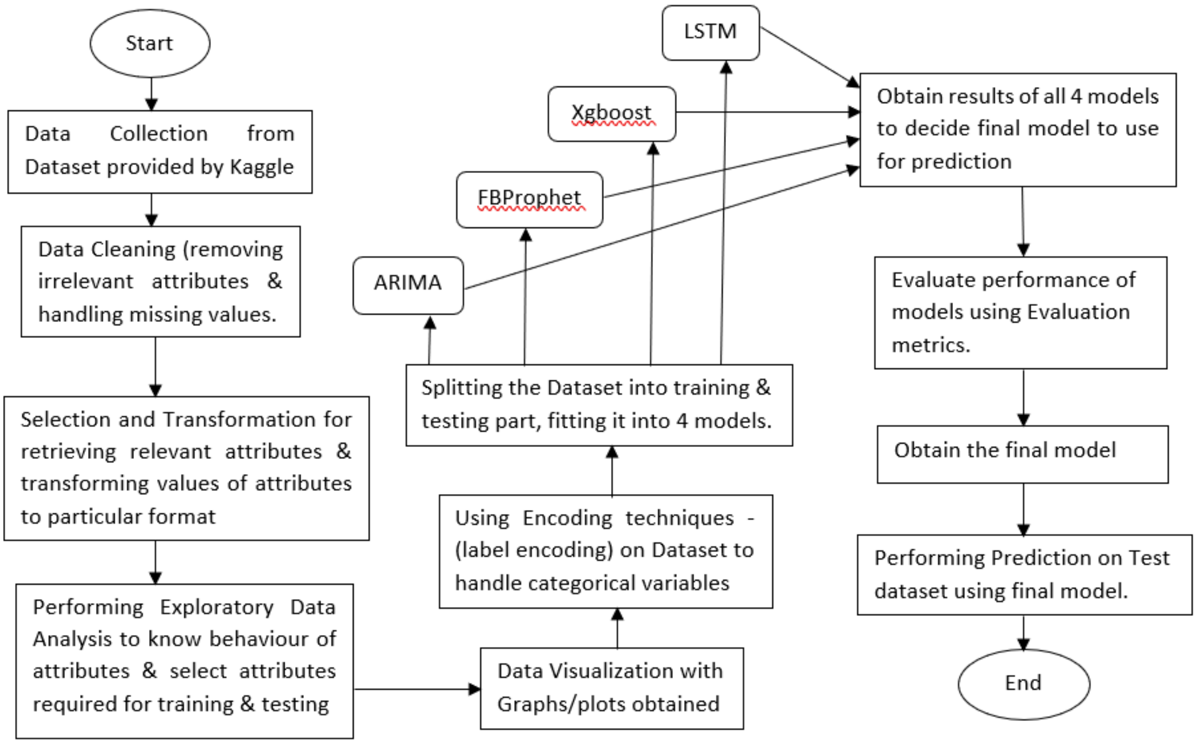
```
bitstamp.head()
```



	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

V. PROPOSED ALGORITHM WITH FLOWCHART

Flowchart:


Start
Data Collection from Dataset provided by Kaggle
Data Cleaning (removing irrelevant attributes & handling missing values.
Selection and Transformation for retrieving relevant attributes & transforming values of attributes to particular format
Performing Exploratory Data Analysis to know behaviour of attributes & select attributes required for training & testing
Data Visualization with Graphs/plots obtained
Using Encoding techniques - (label encoding) on Dataset to handle categorical variables
Splitting the Dataset into training & testing part, fitting it into 4 models.
ARIMA
FBProphet
Xgboost
LSTM
Obtain results of all 4 models to decide final model to use for prediction
Evaluate performance of models using Evaluation metrics.
Obtain the final model
Performing Prediction on Test dataset using final model.
End

1. Arima

ARIMA, short for 'Auto Regressive Integrated Moving Average' is actually a class of models that 'explains' a given time series based on its own past values, that is, its own lags and the lagged forecast errors, so that equation can be used to forecast future values. Any 'non-seasonal' time series that exhibits patterns and is not a random white noise can be modelled with ARIMA models. An ARIMA model is characterized by 3 terms: p, d, q where, p is the order of the AR term, q is the order of the MA term, d is the number of differencing required to make the time series stationary.

```
■ ( AR   I   MA )
     |    |    |
     p   d    q
■
```

- Where
  - p = order of auto-regression (rare>2)
  - d= order of integration (differencing)
  - q= order of moving average (rare>2)

Initialize

- Enter training and forecasting data to ARIMA model.
- Determine parameters p, d and q.

- Determine the presence or absence of a constant term in model.
- Select the best structure based on RMSE index.

End

2. LSTM

Long short-term memory (LSTM) is an artificial recurrent neural network (RNN) architecture used in the field of deep learning. LSTM networks are well-suited to classifying, processing and making predictions based on time series data, since there can be lags of unknown duration between important events in a time series.

```
Input: Multivariable time series [x(0).....x(N-1)] (training set and testing set)
Output: Predicted values y, trained network weights w'
Parameters:
Kernel size: k
Training iterations: M
Training batch size: B
Learning rate:
1: Load training set and testing set from {x(0).....x(N-1)}
2: Randomly initiative weight w
3: Begin Training
4. for {1, M} do
5. Forward passing as equations
6: Calculate loss as equation
7. Calculate gradients of weights
8. Backpropagation and update w
9. End Training
10. Begin Testing calculate predicted values y with testing
11. return predicted values y, w'
```

3. XGBoost

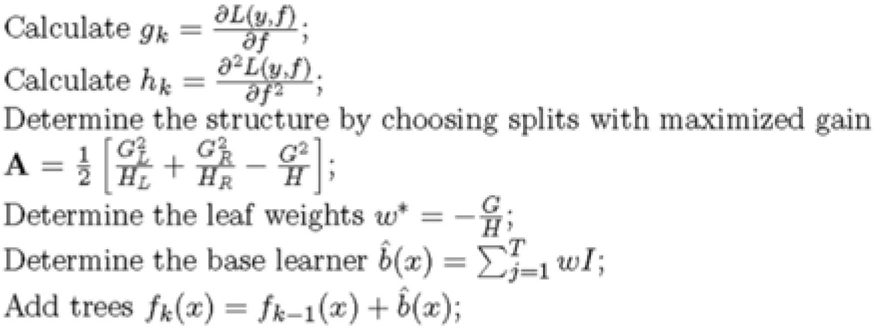
Extreme Gradient Boosting or XGBoost is one of the most popular machine learning models in current times. XGBoost is quite similar at the core to the original gradient boosting algorithm but features many additive features that significantly improve its performance such as built in support for regularization, parallel processing as well as giving additional hyperparameters to tune such as tree pruning, sub sampling and number of decision trees.

Algorithm

Data : Dataset and hyperparameters.

Initialize f_0(x);

for k = 1,2,…,M do



```
    Calculate $g_k = \frac{\partial L(y,f)}{\partial f}$;
    Calculate $h_k = \frac{\partial^2 L(y,f)}{\partial f^2}$;
    Determine the structure by choosing splits with maximized gain
    $\mathbf{A} = \frac{1}{2}\left[\frac{G_L^2}{H_L} + \frac{G_R^2}{H_R} - \frac{G^2}{H}\right]$;
    Determine the leaf weights $w^* = -\frac{G}{H}$;
    Determine the base learner $\hat{b}(x) = \sum_{j=1}^{T} wI$;
    Add trees $f_k(x) = f_{k-1}(x) + \hat{b}(x)$;
```

End

Result: f(x) = $\sum_{k=0}^{M} f_k(x)$

4. FBProphet

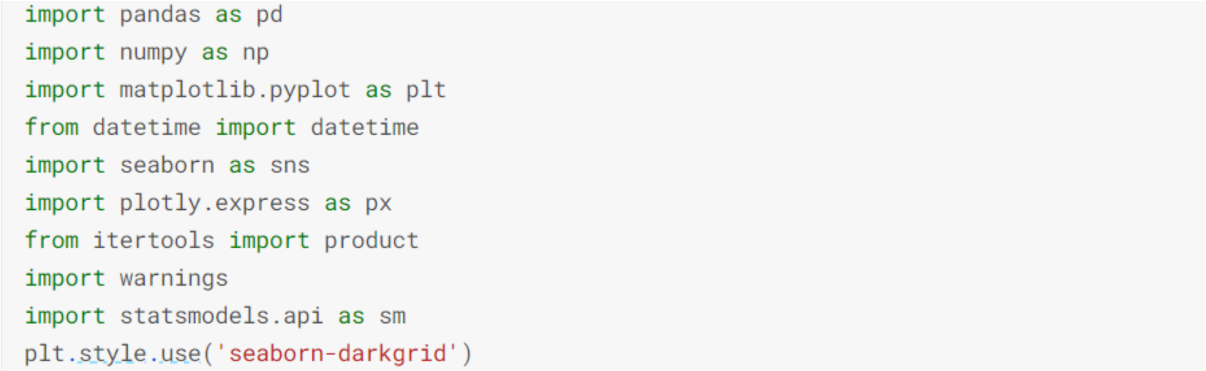
   FaceBook Prophet or FBProphet uses time as a regressor and tries to fit several linear and nonlinear function of time as components. By default, FBProphet will fit the data using a linear model but it can be changed to the nonlinear model (logistics growth) from its arguments.

   - Upload CSV file containing dataset.
   - Calculate basic statistics.
   - Review data.
   - Setup forecast.
   - Fit Prophet predict future.
   - View forecast (and update if needed).
   - Fit Prophet predict future again.
   - Visually inspect forecasts

VI. EXPERIMENTS RESULTS

Dataset taken from Kaggle and coding in python also done in Kaggle notebook: Click Here

In this section, experiments are executed to evaluate the performance of proposed techniques using Python. In my application, four machine learning algorithms are tested and compared.

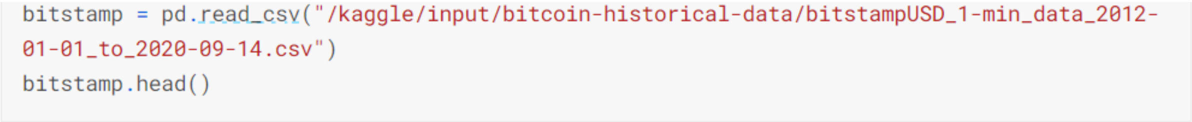


```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from datetime import datetime
import seaborn as sns
import plotly.express as px
from itertools import product
import warnings
import statsmodels.api as sm
plt.style.use('seaborn-darkgrid')
```

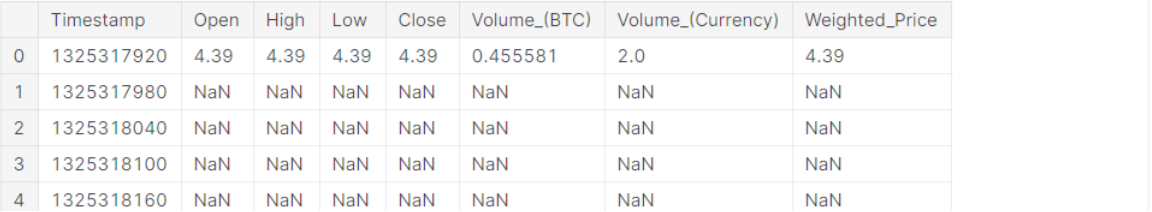
I have used the inbuilt Python libraries such as NUMPY, PANDAS(for linear algebra and scientific computing) and visualization libraries such as MATPLOTLIB, SEABORN.

Reading the dataset

Dataset provided on Kaggle website: Bitcoin Historical Data.

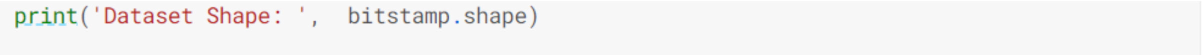


```
bitstamp = pd.read_csv("/kaggle/input/bitcoin-historical-data/bitstampUSD_1-min_data_2012-
01-01_to_2020-09-14.csv")
bitstamp.head()
```

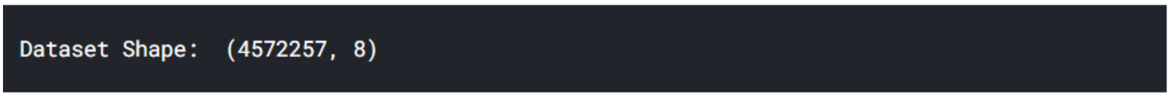


	Timestamp	Open	High	Low	Close	Volume_(BTC)	Volume_(Currency)	Weighted_Price
0	1325317920	4.39	4.39	4.39	4.39	0.455581	2.0	4.39
1	1325317980	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	1325318040	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	1325318100	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	1325318160	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Importing the required libraries. Importing the dataset (stored locally). After importing we check the shape of the dataset



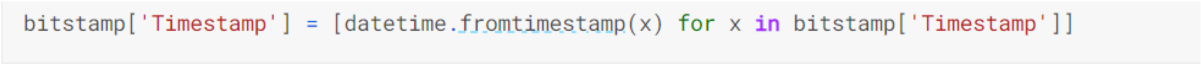
```
print('Dataset Shape: ',  bitstamp.shape)
```



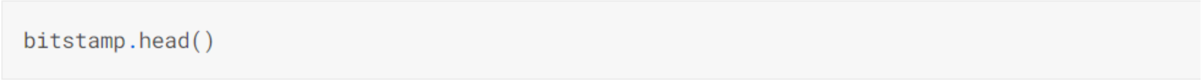
```
Dataset Shape:  (4572257, 8)
```

Function to convert timestamp into the required form

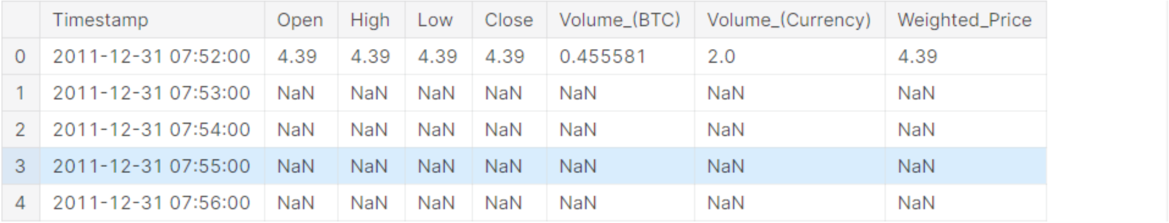
Viewing the first 5 rows of the dataframe.



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



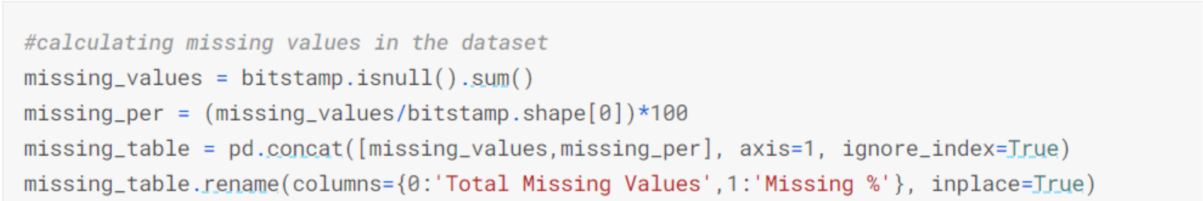
```python
bitstamp.head()
```



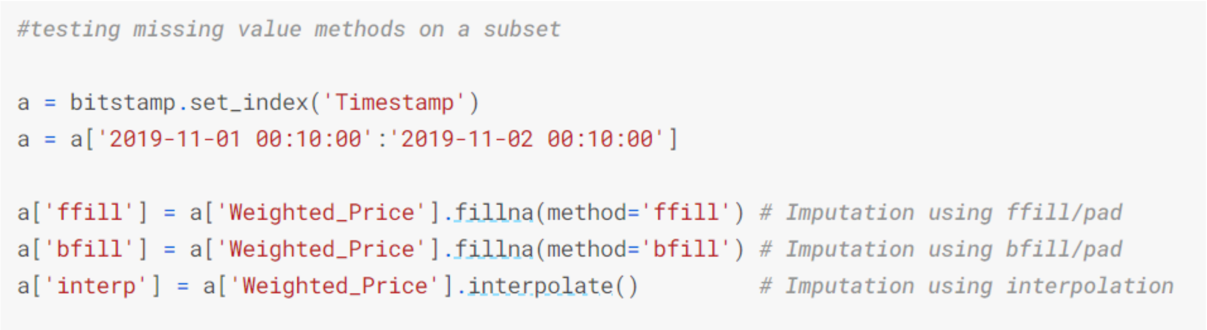
	Timestamp	Open	High	Low	Close	Volume_(BTC)	Volume_(Currency)	Weighted_Price
0	2011-12-31 07:52:00	4.39	4.39	4.39	4.39	0.455581	2.0	4.39
1	2011-12-31 07:53:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	2011-12-31 07:54:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	2011-12-31 07:55:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	2011-12-31 07:56:00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

Data Pre-processing

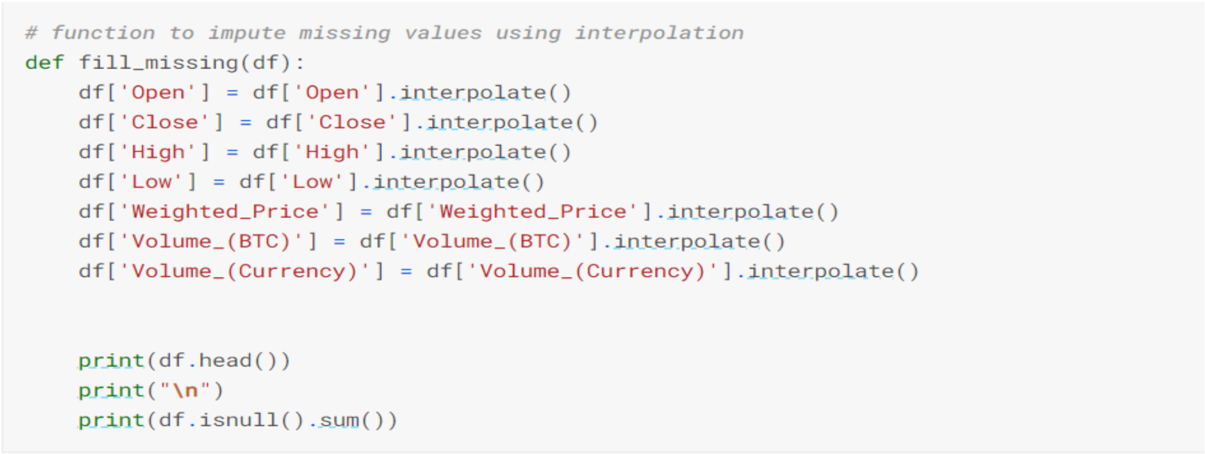
Data pre-processing is a data mining technique that involves transforming raw data into an understandable format. Real-world data is often incomplete, inconsistent, and/or lacking in certain behaviours or trends, and is likely to contain many errors.



```python
#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)
```



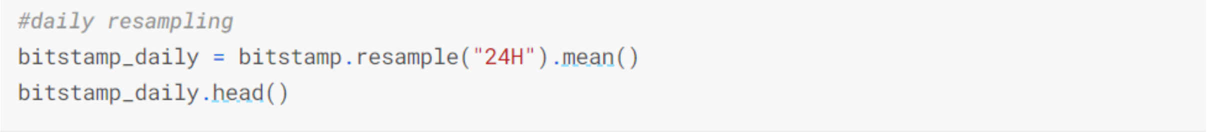
```python
#testing missing value methods on a subset

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
```



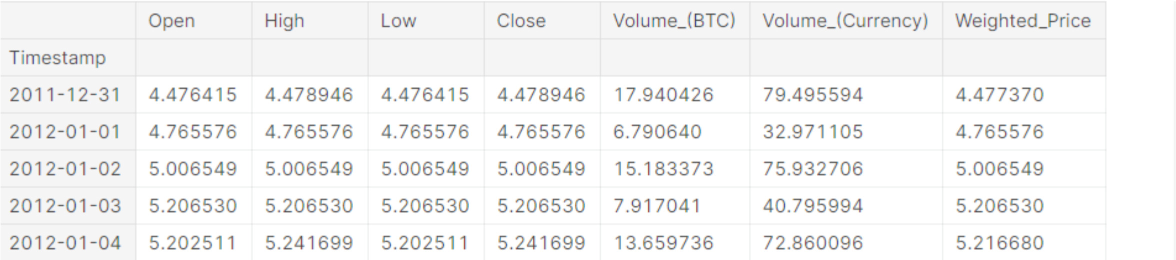
```python
# function to impute missing values using interpolation
def fill_missing(df):
    df['Open'] = df['Open'].interpolate()
    df['Close'] = df['Close'].interpolate()
    df['High'] = df['High'].interpolate()
    df['Low'] = df['Low'].interpolate()
    df['Weighted_Price'] = df['Weighted_Price'].interpolate()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

Data visualization helps identify areas that need attention, e.g outliers, which can later impact our machine learning model. It also helps us understand the factors that have more impacts on our results.



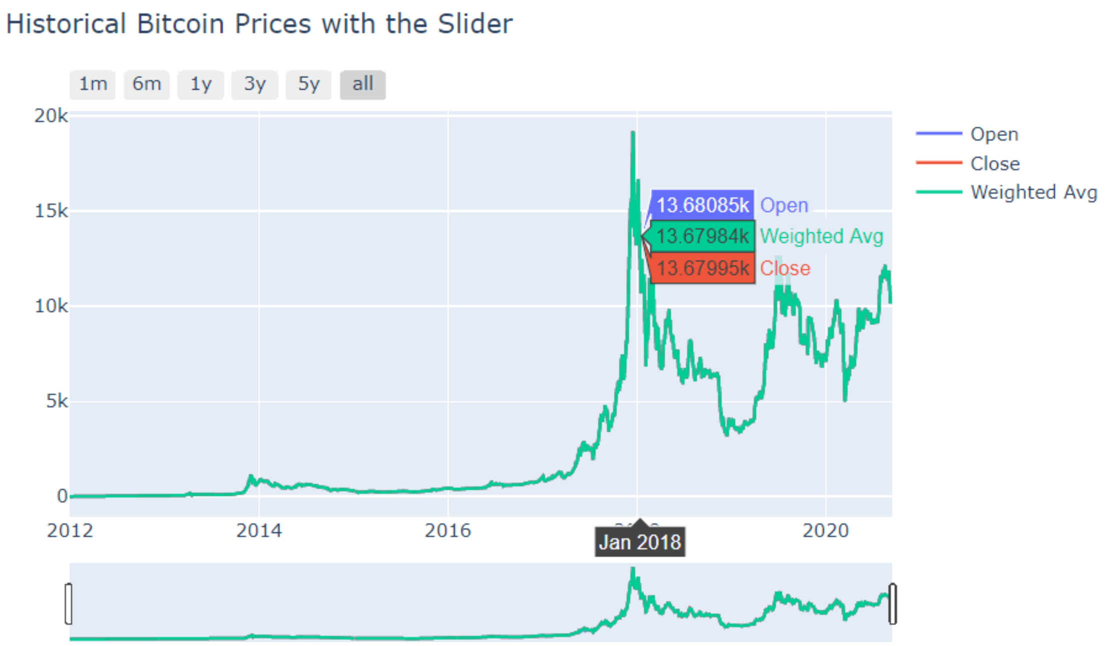
```
#daily resampling
bitstamp_daily = bitstamp.resample("24H").mean()
bitstamp_daily.head()
```



	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")
```

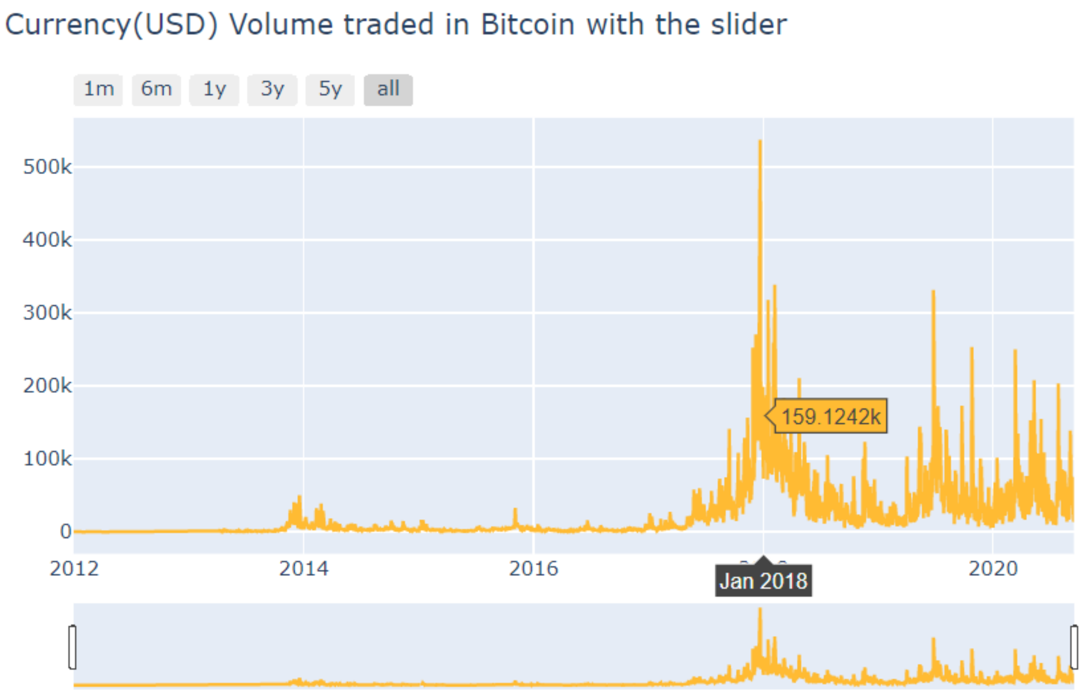
We use different graphs and plots to visualize complex data to ease the discovery of data patterns.



```
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',
),
```

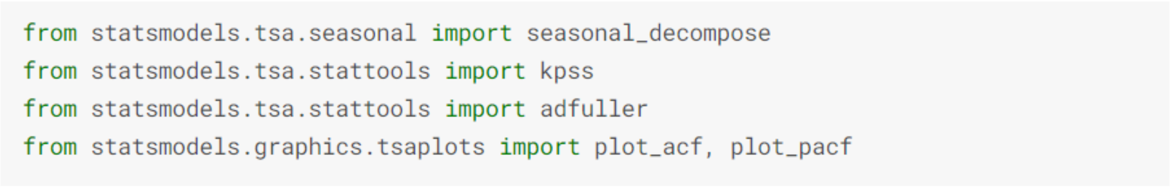
```
            stepmode='backward'),
        dict(count=6,
            label='6m',
            step='month',
            stepmode='backward'),
        dict(count=12,
            label='1y',
            step='month',
            stepmode='backward'),
        dict(count=36,
            label='3y',
            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")
```

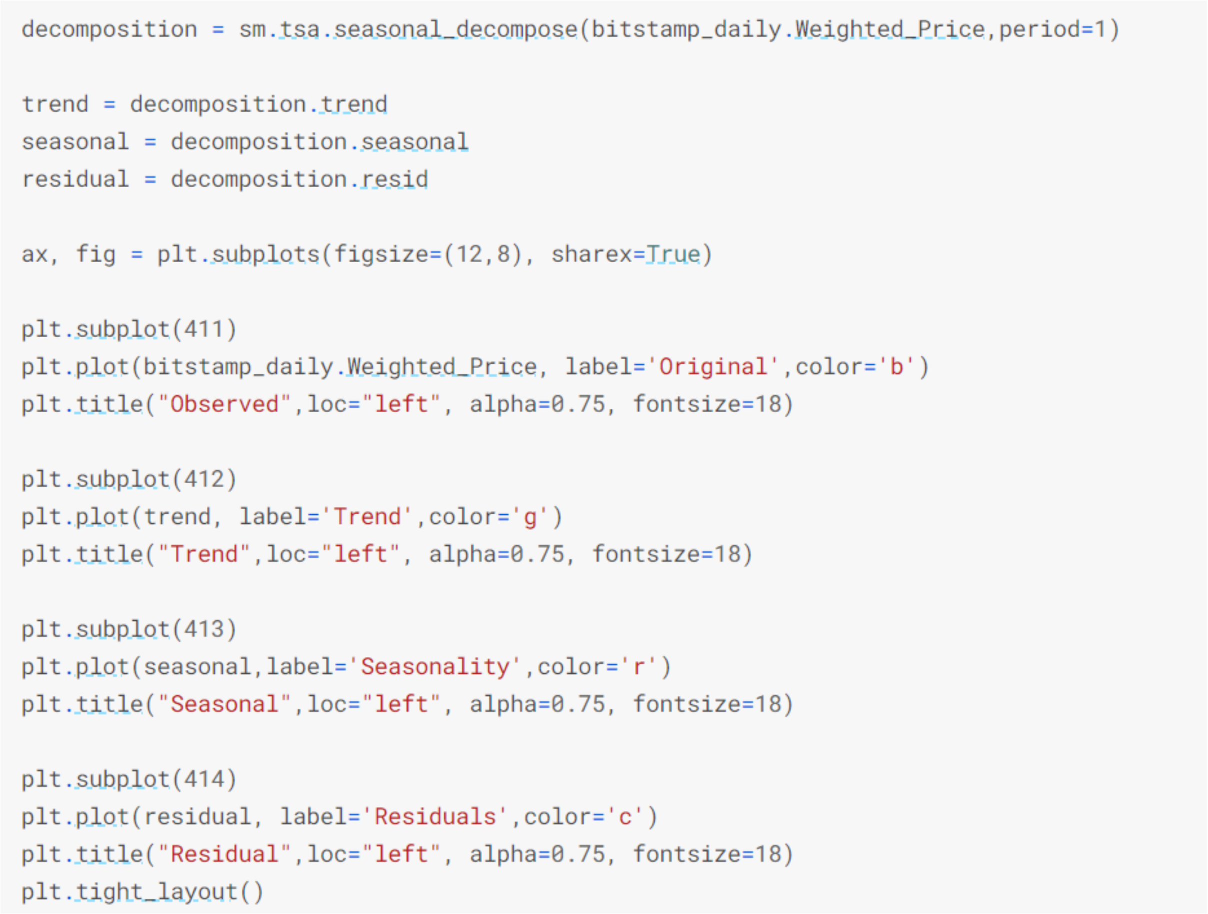


Time Series Decomposition & Statistical Tests

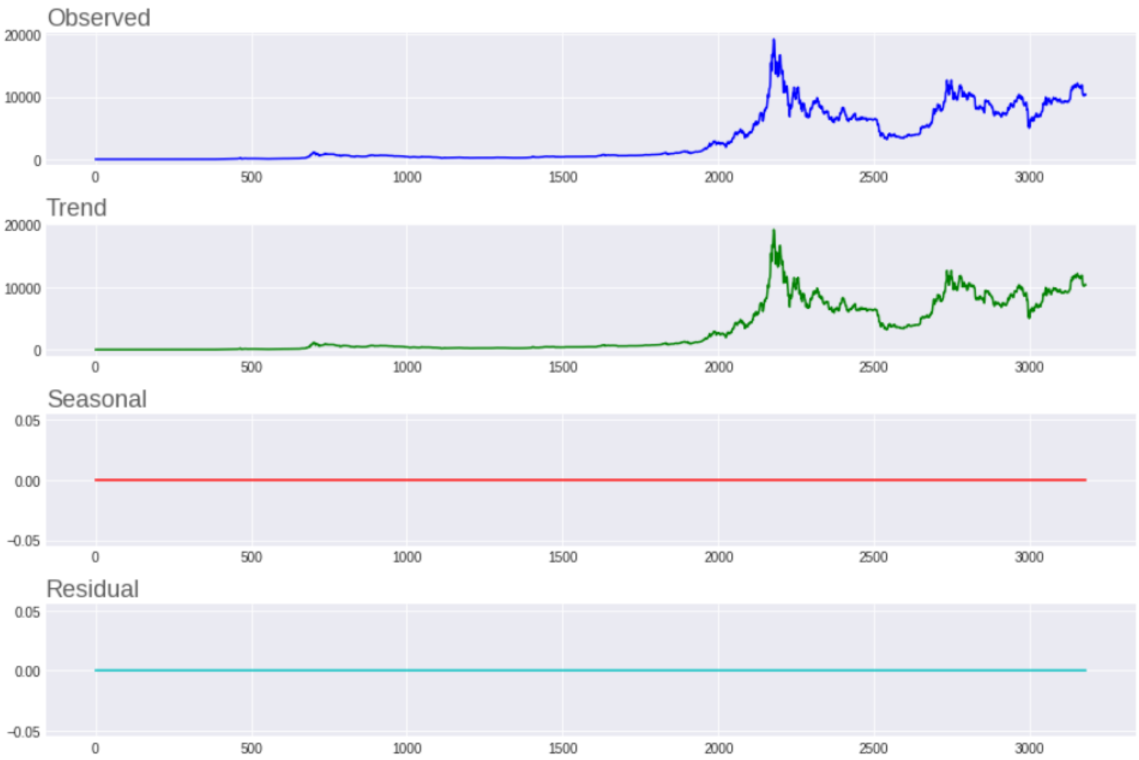
We can decompose a time series into trend, seasonal and remainder components. The seasonal_decompose in statsmodels is used to implements the decomposition.



```
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
```



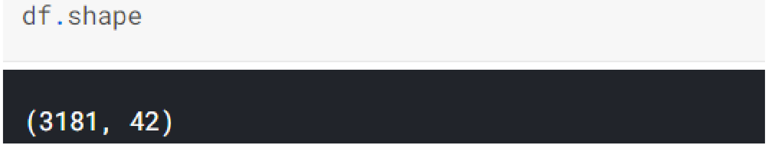
```
decomposition = sm.tsa.seasonal_decompose(bitstamp_daily.Weighted_Price,period=1)

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()
```

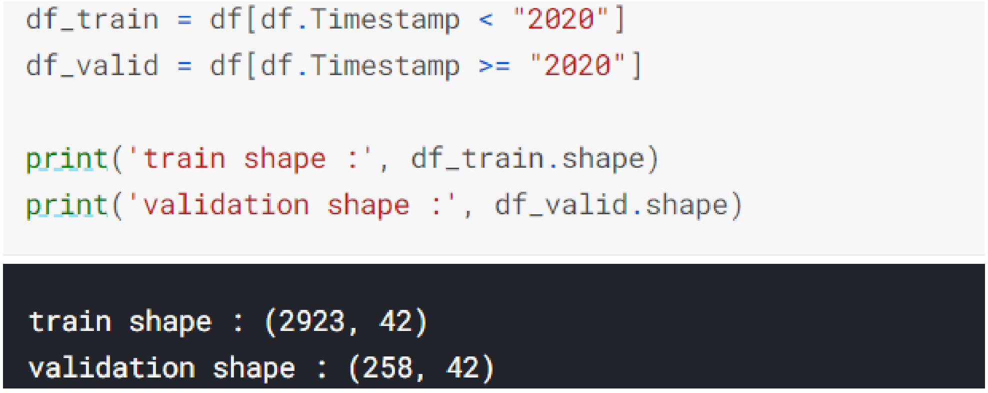


```
df.shape
```

```
(3181, 42)
```

Model Building:

Split data into training and test or validation data.

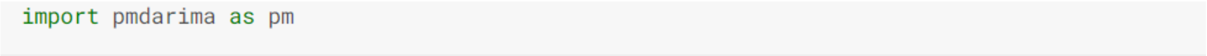


```
df_train = df[df.Timestamp < "2020"]
df_valid = df[df.Timestamp >= "2020"]

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

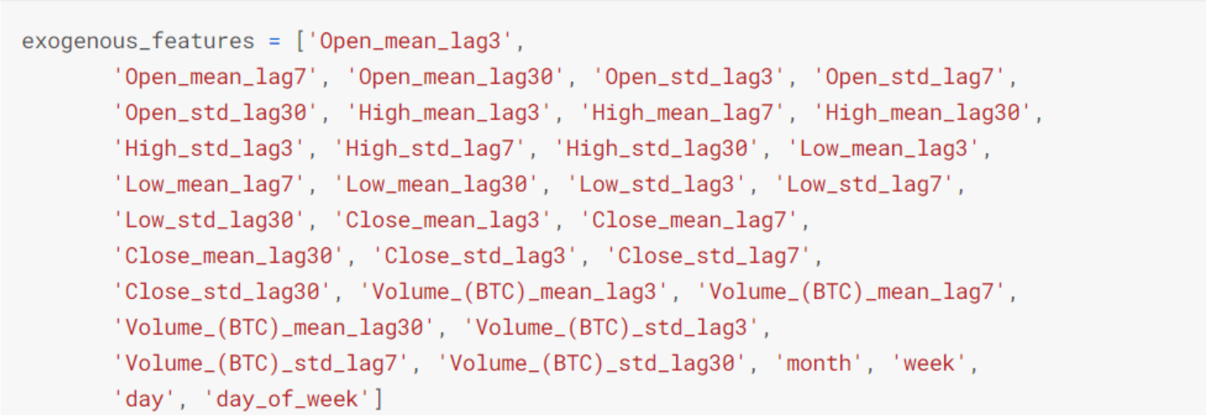
```
train shape : (2923, 42)
validation shape : (258, 42)
```

1) ARIMA Model

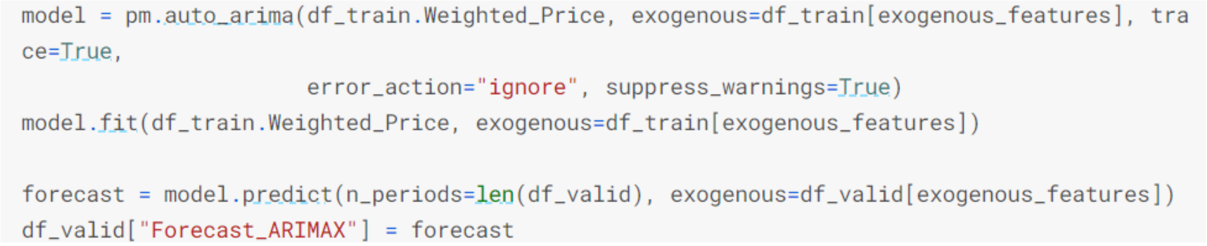
Pmdarima (originally pyramid-arima, for the anagram of 'py' + 'arima') is a statistical library designed to fill the void in Python's time series analysis capabilities.



```
import pmdarima as pm
```

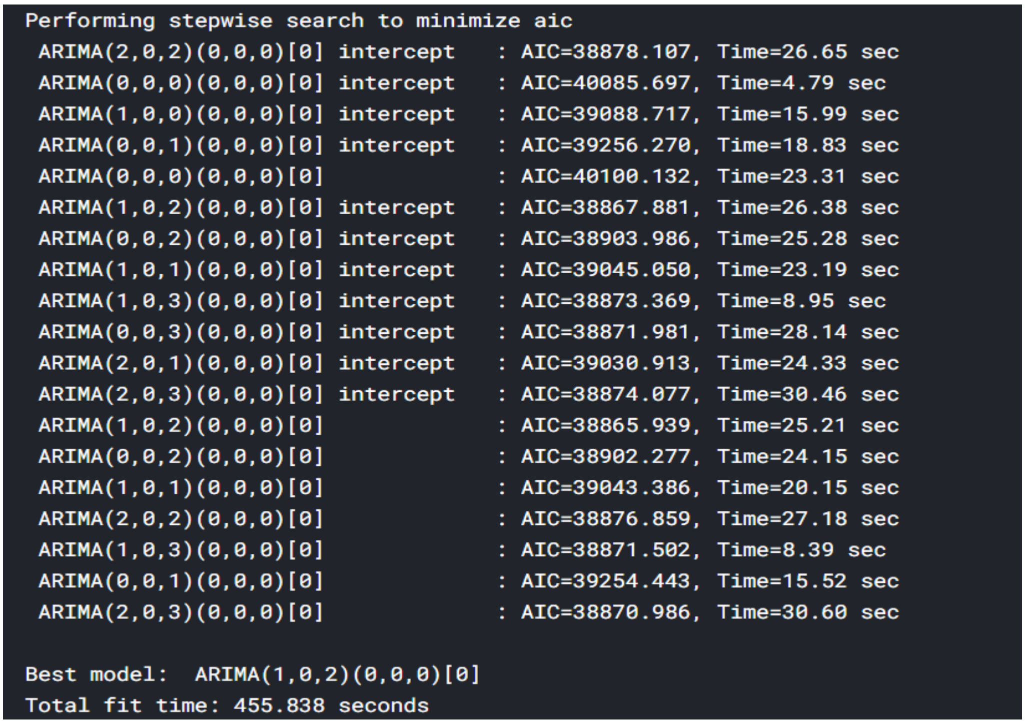


```
exogenous_features = ['Open_mean_lag3',
       'Open_mean_lag7', 'Open_mean_lag30', 'Open_std_lag3', 'Open_std_lag7',
       'Open_std_lag30', 'High_mean_lag3', 'High_mean_lag7', 'High_mean_lag30',
       'High_std_lag3', 'High_std_lag7', 'High_std_lag30', 'Low_mean_lag3',
       'Low_mean_lag7', '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']
```



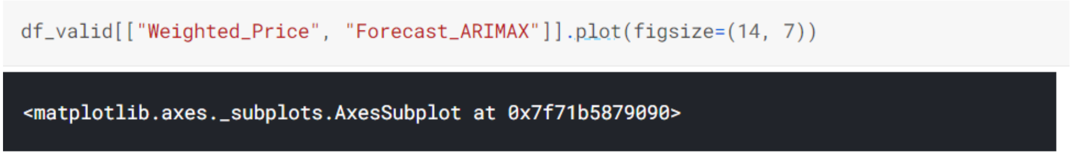
```
model = pm.auto_arima(df_train.Weighted_Price, exogenous=df_train[exogenous_features], tra
ce=True,
                      error_action="ignore", suppress_warnings=True)
model.fit(df_train.Weighted_Price, exogenous=df_train[exogenous_features])

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

Stepwise regression is a modification of the forward selection so that after each step in which a variable was added, all candidate variables in the model are checked to see if their significance has been reduced below the specified tolerance level. AIC estimates the relative amount of information lost by a given model: the less information a model loses, the higher the quality of that model.



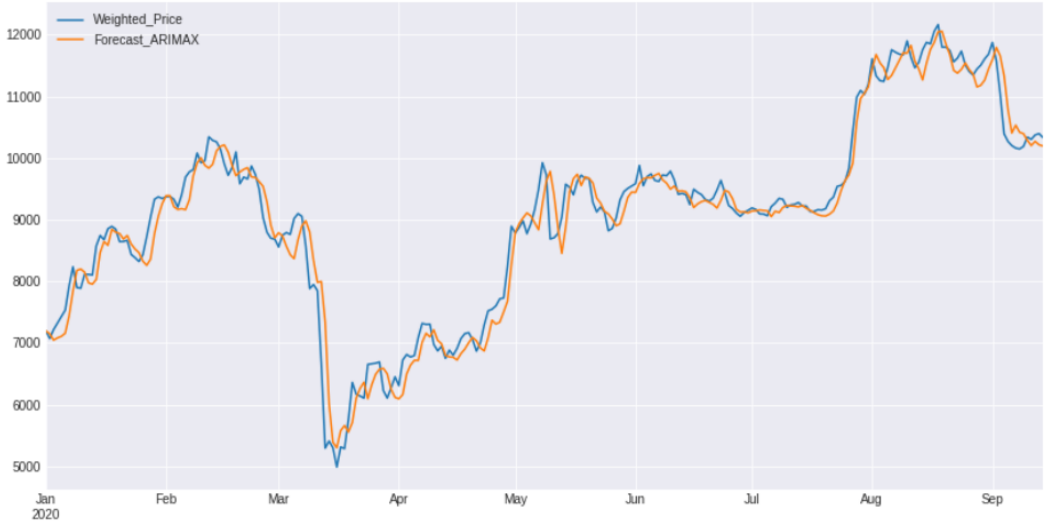
```
Performing stepwise search to minimize aic
 ARIMA(2,0,2)(0,0,0)[0] intercept   : AIC=38878.107, Time=26.65 sec
 ARIMA(0,0,0)(0,0,0)[0] intercept   : AIC=40085.697, Time=4.79 sec
 ARIMA(1,0,0)(0,0,0)[0] intercept   : AIC=39088.717, Time=15.99 sec
 ARIMA(0,0,1)(0,0,0)[0] intercept   : AIC=39256.270, 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
```

Use matplotlib library to visualise weighted price and forecast ARIMAX.

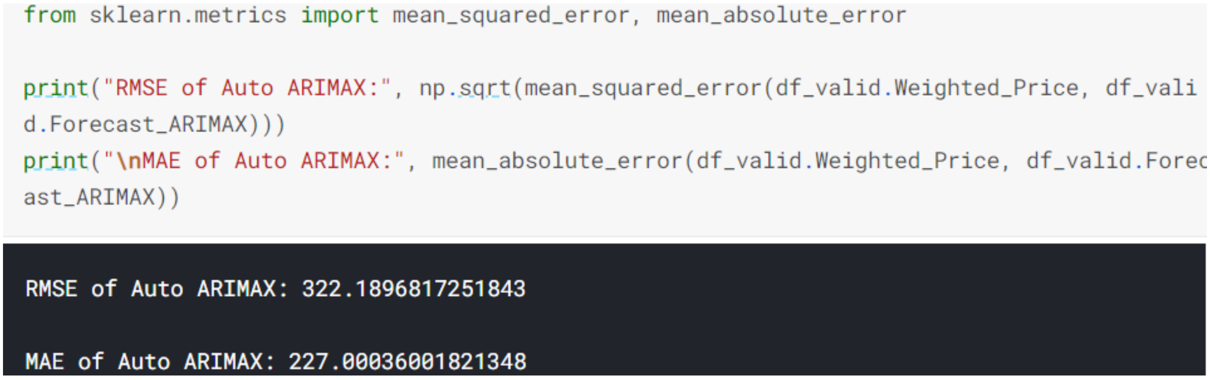


```
df_valid[["Weighted_Price", "Forecast_ARIMAX"]].plot(figsize=(14, 7))
```

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



Calculate RMSE and MAE. The Mean Absolute Error (MAE) is a very good KPI to measure forecast accuracy. The Root Mean Squared Error (RMSE) is defined as the square root of the average squared error.

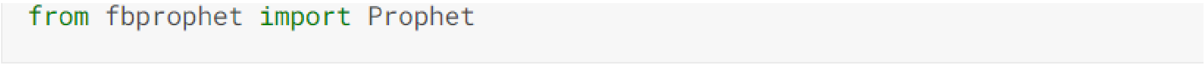


```
from sklearn.metrics import mean_squared_error, mean_absolute_error

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))
```

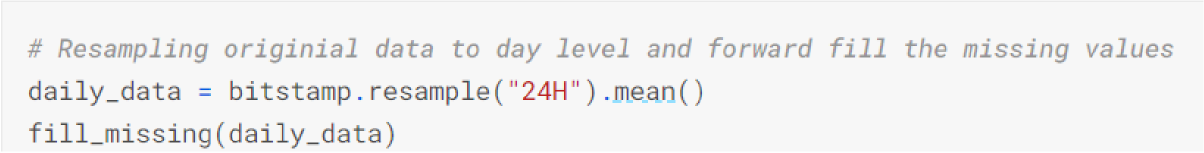
```
RMSE of Auto ARIMAX: 322.1896817251843

MAE of Auto ARIMAX: 227.00036001821348
```

2) Facebook Prophet

Prophet, or “Facebook Prophet,” is an open-source library for univariate (one variable) time series forecasting developed by Facebook. Import the library. Resample original data and fill the missing values.

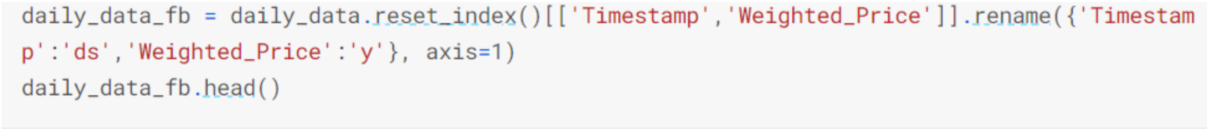


```
from fbprophet import Prophet
```



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

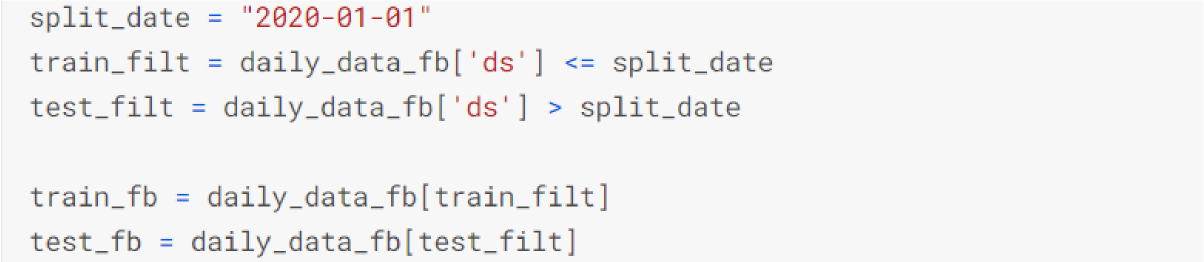
According to the requirements of the model we rename the columns of the dataset.



```
daily_data_fb = daily_data.reset_index()[['Timestamp','Weighted_Price']].rename({'Timestam
p':'ds','Weighted_Price':'y'}, axis=1)
daily_data_fb.head()
```

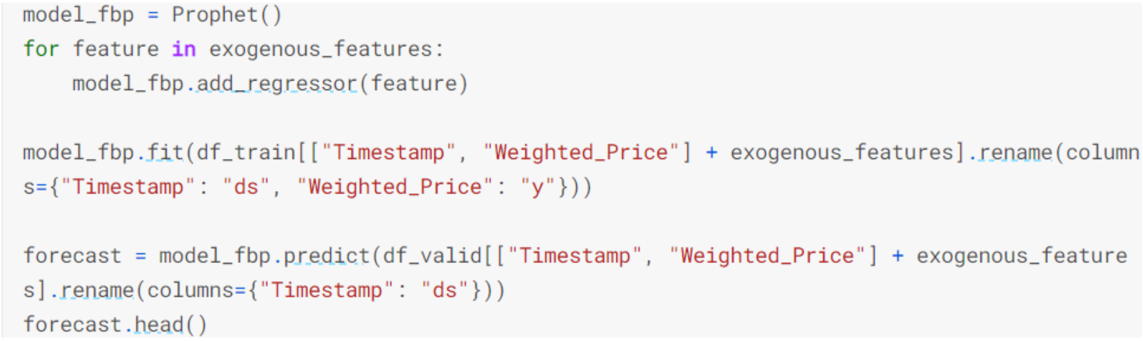
	ds	y
0	2011-12-31	4.477370
1	2012-01-01	4.765576
2	2012-01-02	5.006549
3	2012-01-03	5.206530
4	2012-01-04	5.216680

Split the data into training and testing data.



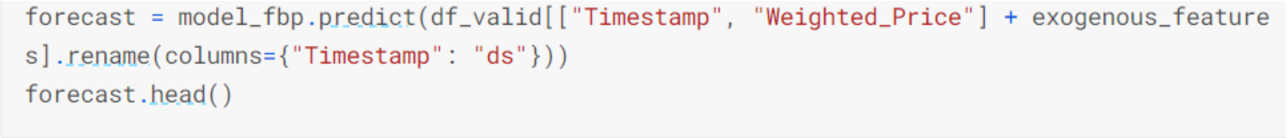
```
split_date = "2020-01-01"
train_filt = daily_data_fb['ds'] <= split_date
test_filt = daily_data_fb['ds'] > split_date

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

We fit the model by instantiating a new Prophet object. Any settings to the forecasting procedure are passed into the constructor. Then we call its fit method and pass in the historical dataframe.

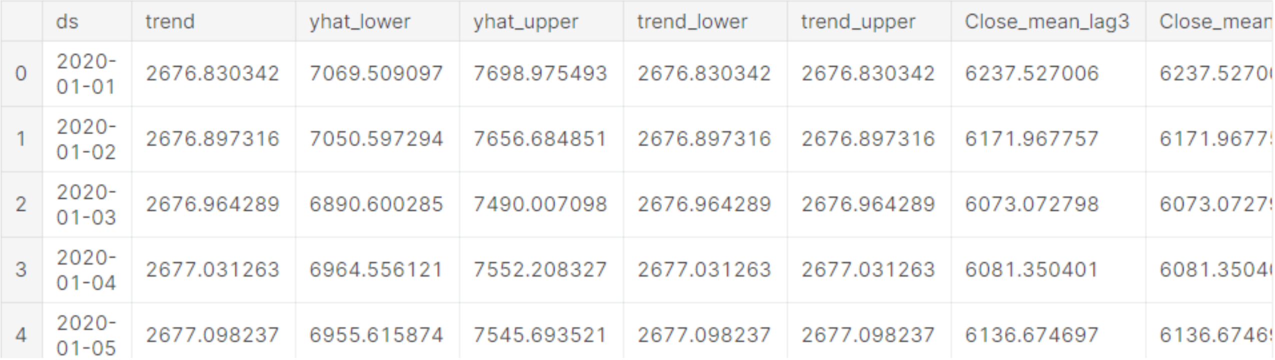


```
model_fbp = Prophet()
for feature in exogenous_features:
    model_fbp.add_regressor(feature)

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()
```

The predict method will assign each row in future a predicted value which it names yhat. If you pass in historical dates, it will provide an in-sample fit. The forecast object here is a new dataframe that includes a column yhat with the forecast, as well as columns for components and uncertainty intervals.

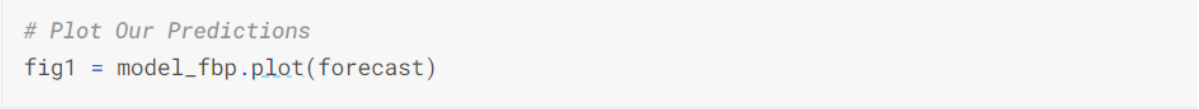


```
forecast = model_fbp.predict(df_valid[["Timestamp", "Weighted_Price"] + exogenous_feature
s].rename(columns={"Timestamp": "ds"}))
forecast.head()
```

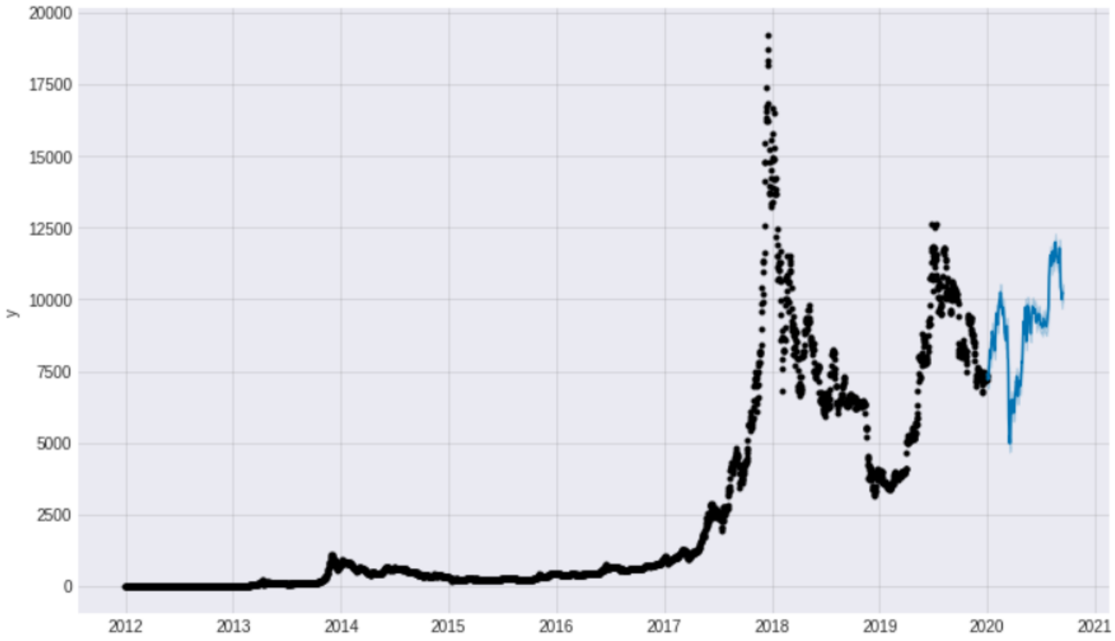


	ds	trend	yhat_lower	yhat_upper	trend_lower	trend_upper	Close_mean_lag3	Close_mean
0	2020-01-01	2676.830342	7069.509097	7698.975493	2676.830342	2676.830342	6237.527006	6237.5270
1	2020-01-02	2676.897316	7050.597294	7656.684851	2676.897316	2676.897316	6171.967757	6171.9677
2	2020-01-03	2676.964289	6890.600285	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

Next, we plot the forecast by calling the Prophet.plot method and passing in our forecast dataframe.



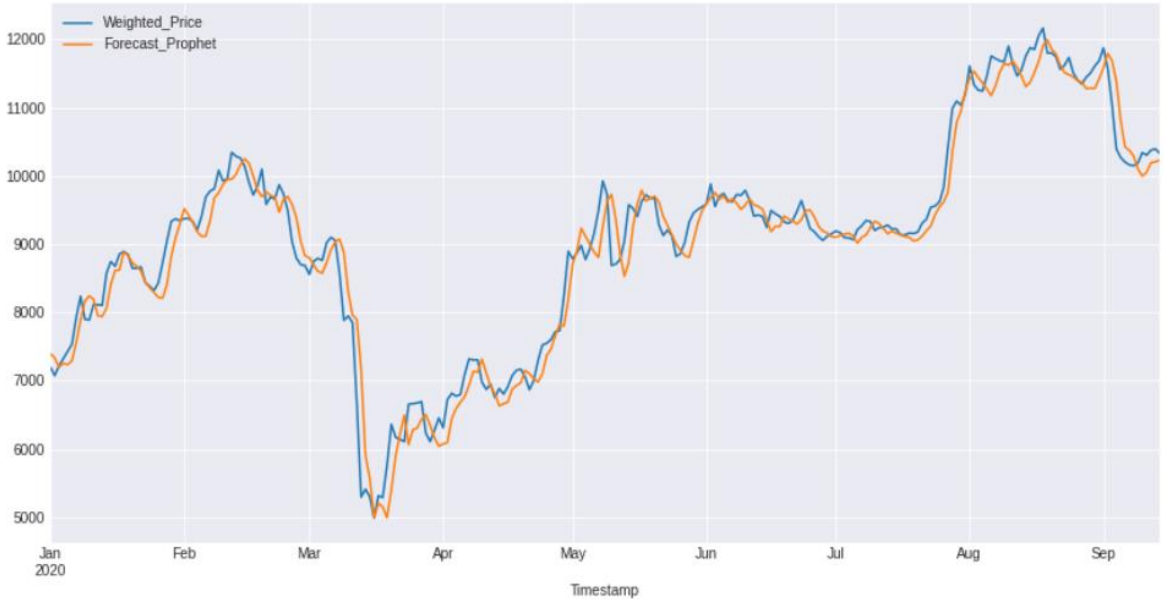
```
# Plot Our Predictions
fig1 = model_fbp.plot(forecast)
```



Next, we use matplotlib library again to plot the weighted price with the results or forecasts from FBProphet model.

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

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



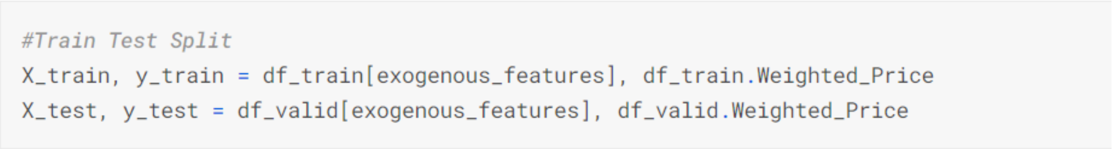
Again, we calculate MAE and RMSE values for FBProphet.

```
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_Prop
het']))

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

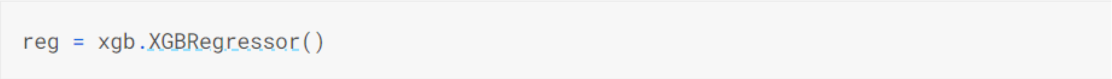
```
Prophet's MAE : 233.0184261175854
Prophet's RMSE : 327.348083280574
```

3) XGBoost

```
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
```

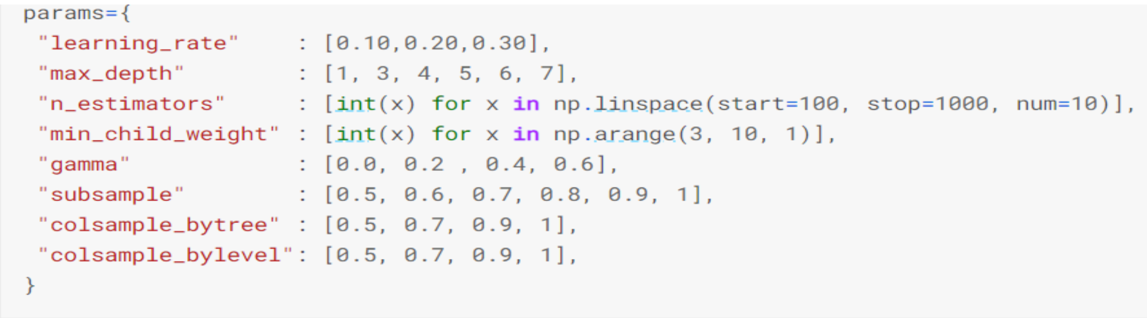


```
#Train Test Split
X_train, y_train = df_train[exogenous_features], df_train.Weighted_Price
X_test, y_test = df_valid[exogenous_features], df_valid.Weighted_Price
```

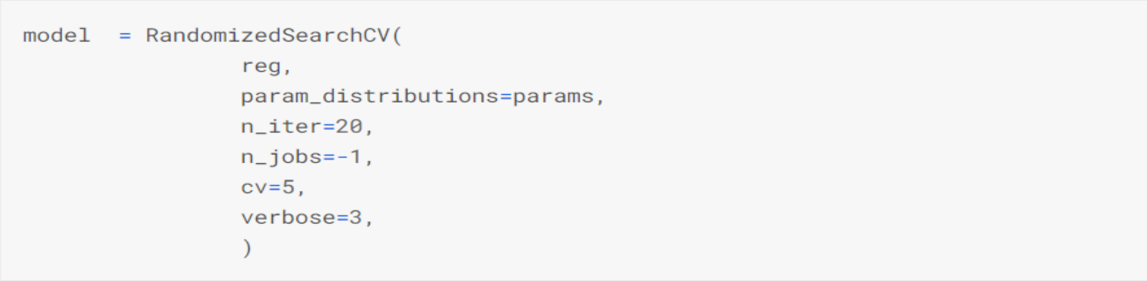


```
reg = xgb.XGBRegressor()
```

We have tuned our model using Hyperparameter tuning.

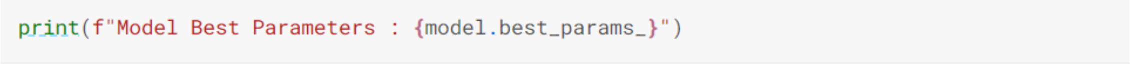


```
params={
 "learning_rate"    : [0.10,0.20,0.30],
 "max_depth"        : [1, 3, 4, 5, 6, 7],
 "n_estimators"     : [int(x) for x in np.linspace(start=100, stop=1000, num=10)],
 "min_child_weight" : [int(x) for x in np.arange(3, 10, 1)],
 "gamma"            : [0.0, 0.2 , 0.4, 0.6],
 "subsample"    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

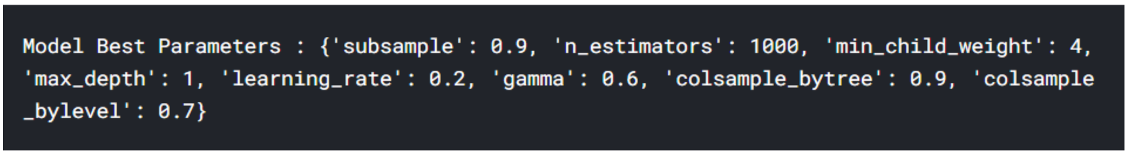


```
model  = RandomizedSearchCV(
                reg,
                param_distributions=params,
                n_iter=20,
                n_jobs=-1,
                cv=5,
                verbose=3,
                )
```

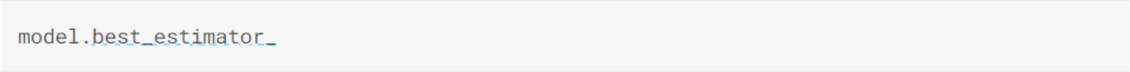
For the regression problem, I will use the XGBRegressor class of the xgboost package and define it with its default parameters.



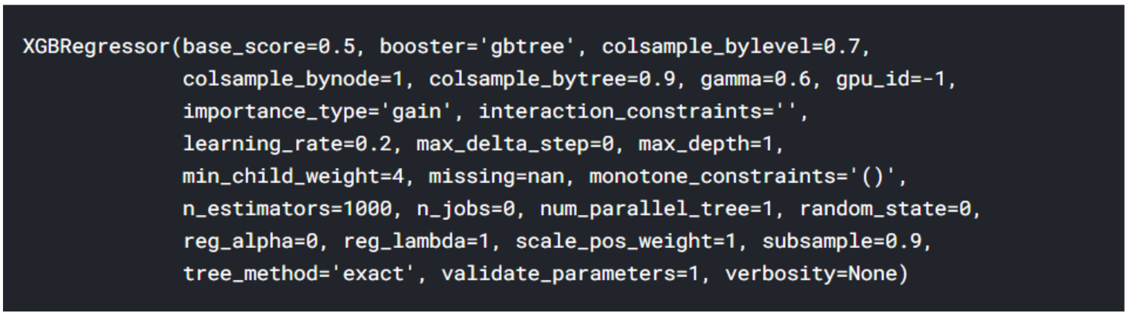
```
print(f"Model Best Parameters : {model.best_params_}")
```



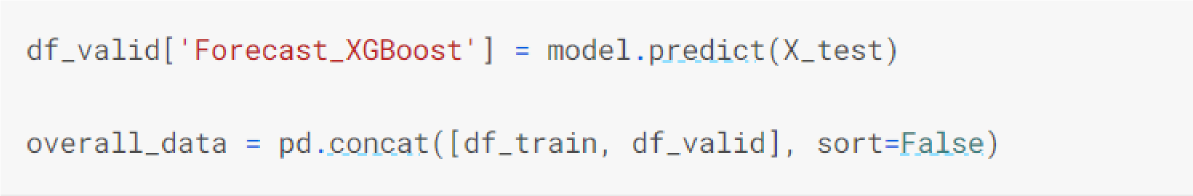
```
Model Best Parameters : {'subsample': 0.9, 'n_estimators': 1000, 'min_child_weight': 4,
'max_depth': 1, 'learning_rate': 0.2, 'gamma': 0.6, 'colsample_bytree': 0.9, 'colsample
_bylevel': 0.7}
```



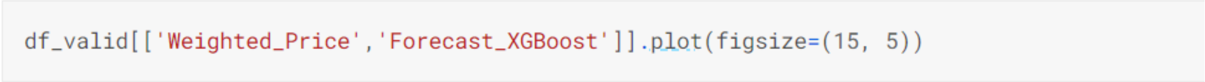
```
model.best_estimator_
```



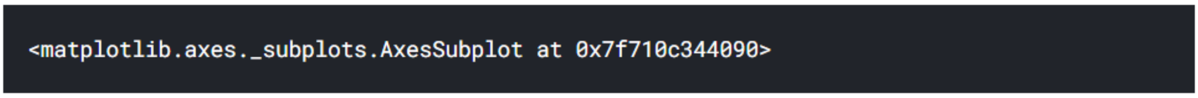
```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=0.7,
             colsample_bynode=1, colsample_bytree=0.9, gamma=0.6, gpu_id=-1,
             importance_type='gain', interaction_constraints='',
             learning_rate=0.2, max_delta_step=0, max_depth=1,
             min_child_weight=4, 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)
```



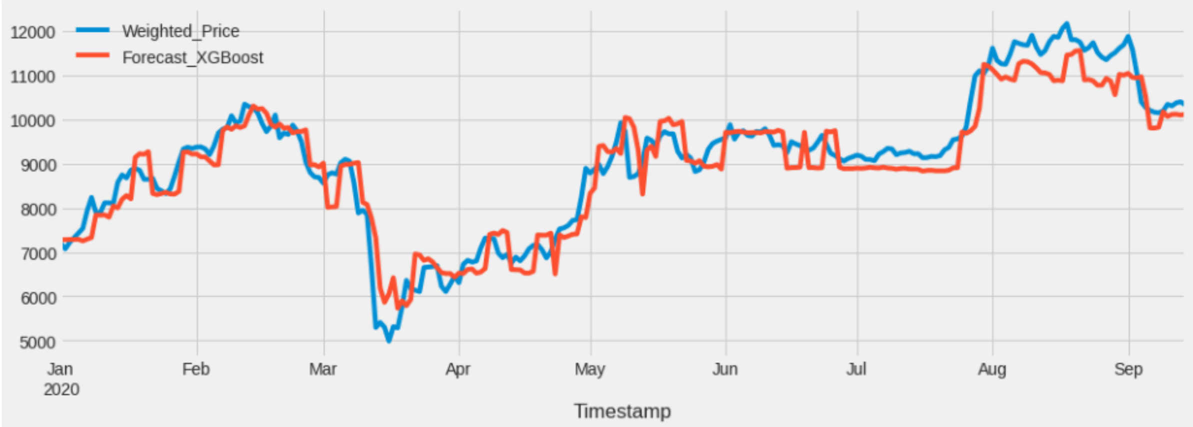
```
df_valid['Forecast_XGBoost'] = model.predict(X_test)

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

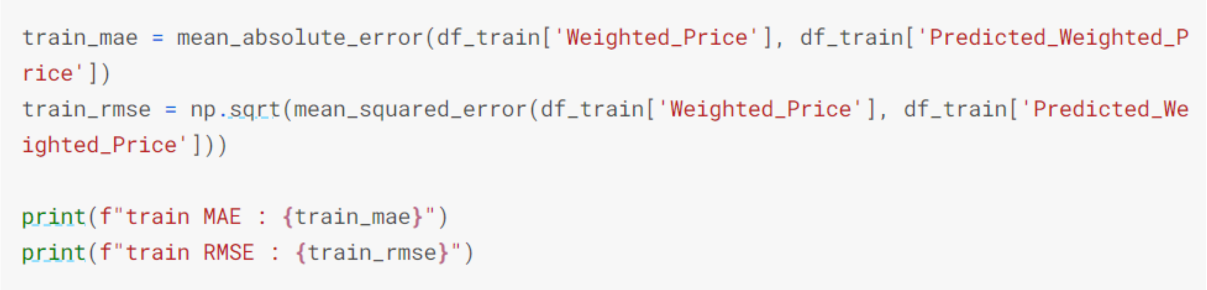


```
df_valid[['Weighted_Price','Forecast_XGBoost']].plot(figsize=(15, 5))
```

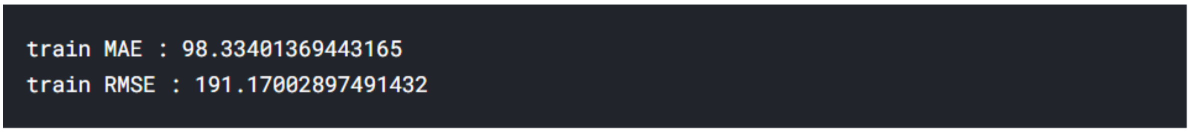


```
<matplotlib.axes._subplots.AxesSubplot at 0x7f710c344090>
```

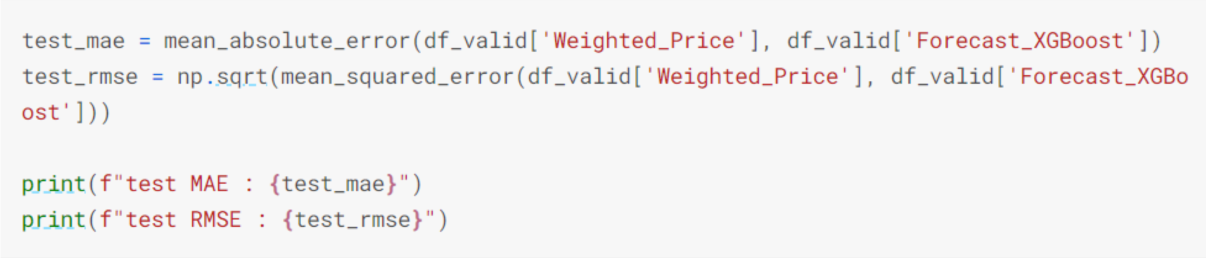




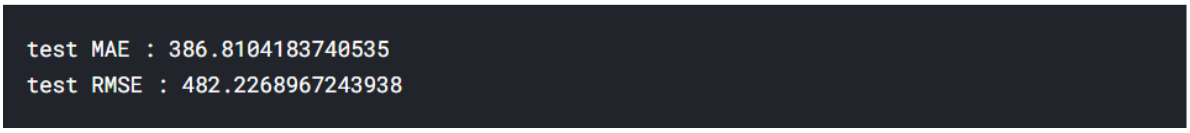
```
train_mae = mean_absolute_error(df_train['Weighted_Price'], df_train['Predicted_Weighted_P
rice'])
train_rmse = np.sqrt(mean_squared_error(df_train['Weighted_Price'], df_train['Predicted_We
ighted_Price']))

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_XGBo
ost']))

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

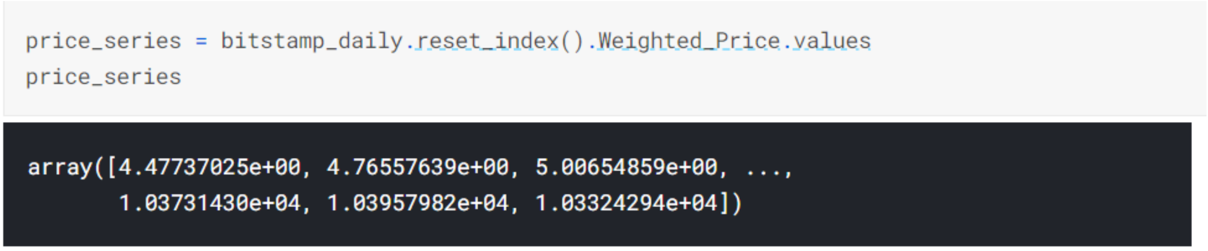


```
test MAE : 386.8104183740535
test RMSE : 482.2268967243938
```

4) LSTM

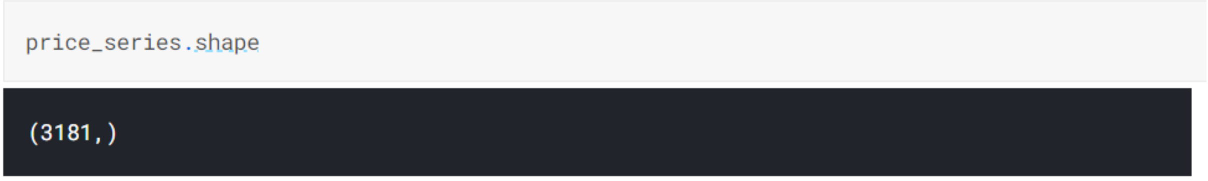
LSTMs are very powerful in sequence prediction problems because they are able to store past information. This is important in our case because the previous price of bitcoin is crucial in predicting its future price.

Sklearn’s MinMaxScaler transforms features by scaling each feature to a given range. This estimator scales and translates each feature individually such that it is in the given range on the training set, e.g. between zero and one.



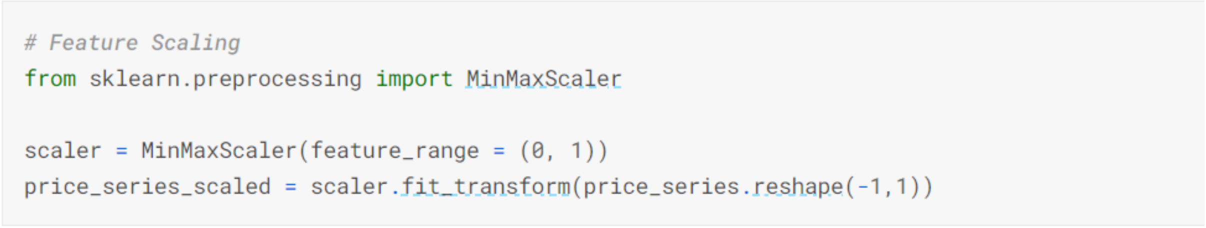
```
price_series = bitstamp_daily.reset_index().Weighted_Price.values
price_series
```

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

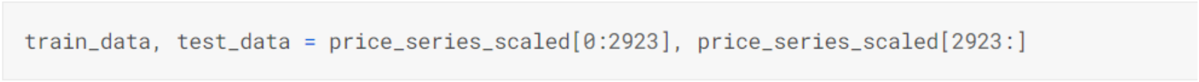


```
price_series.shape
```

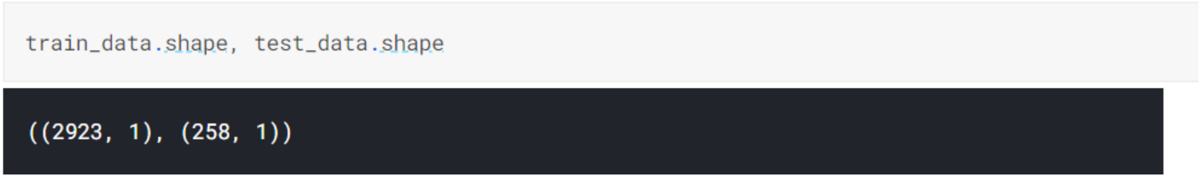
```
(3181,)
```



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

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



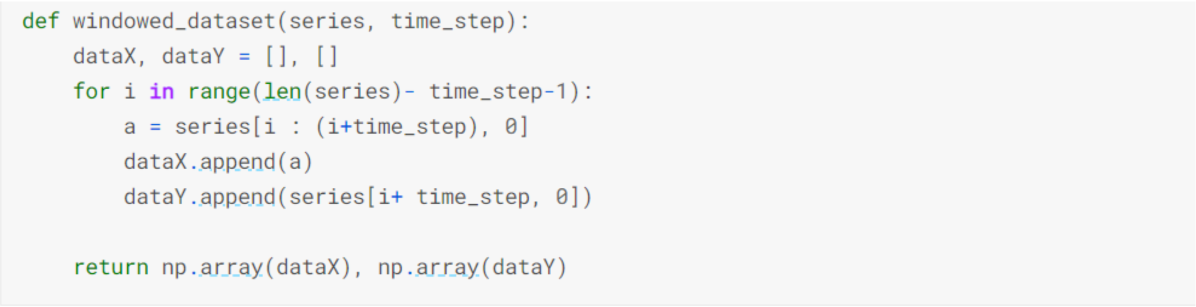
```
train_data, test_data = price_series_scaled[0:2923], price_series_scaled[2923:]
```



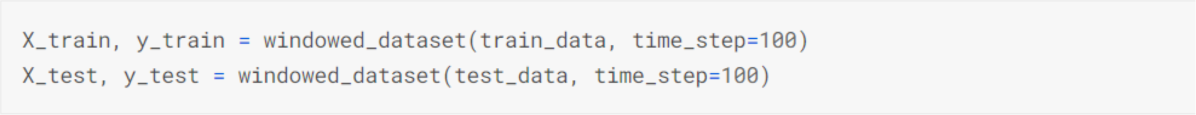
```
train_data.shape, test_data.shape
```

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

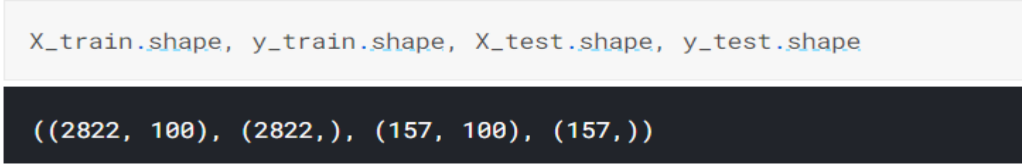
In the case of time series data, our features will be a number a values in the series, and our labels will be the next value. The number of values that we treat as our feature is referred to as the window size, which means we're using a window of data to train the model in order to predict the next value.



```
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)
```

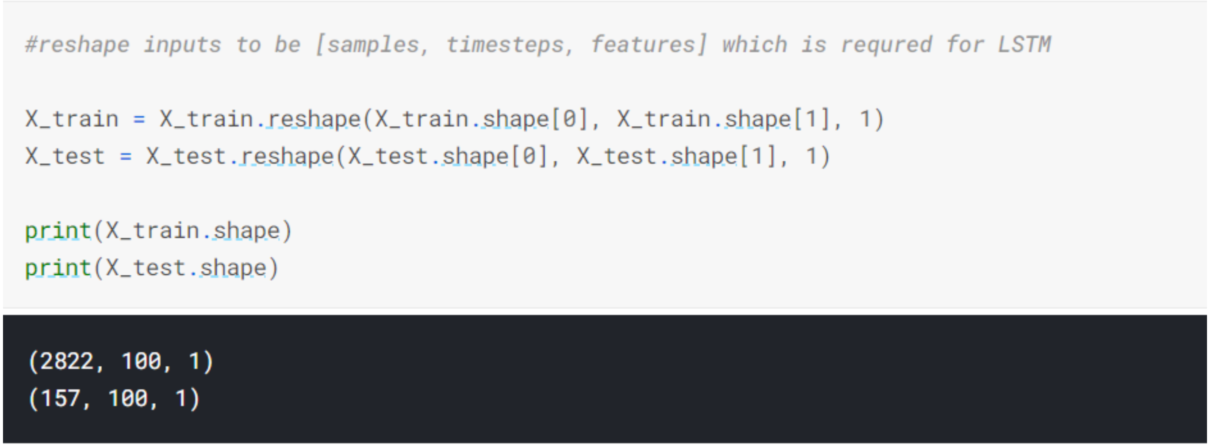


```
X_train, y_train = windowed_dataset(train_data, time_step=100)
X_test, y_test = windowed_dataset(test_data, time_step=100)
```



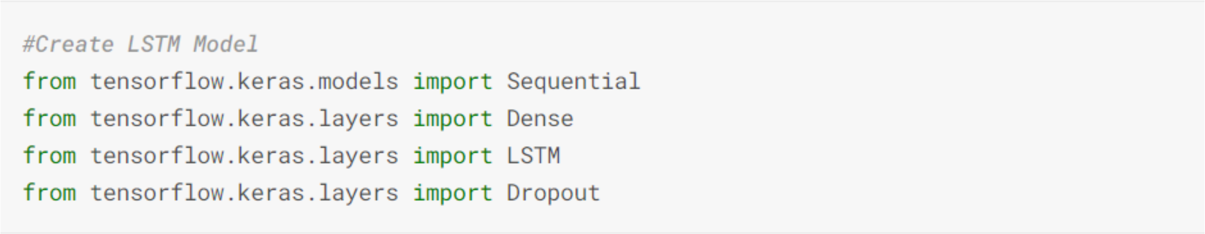
```
X_train.shape, y_train.shape, X_test.shape, y_test.shape
```

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



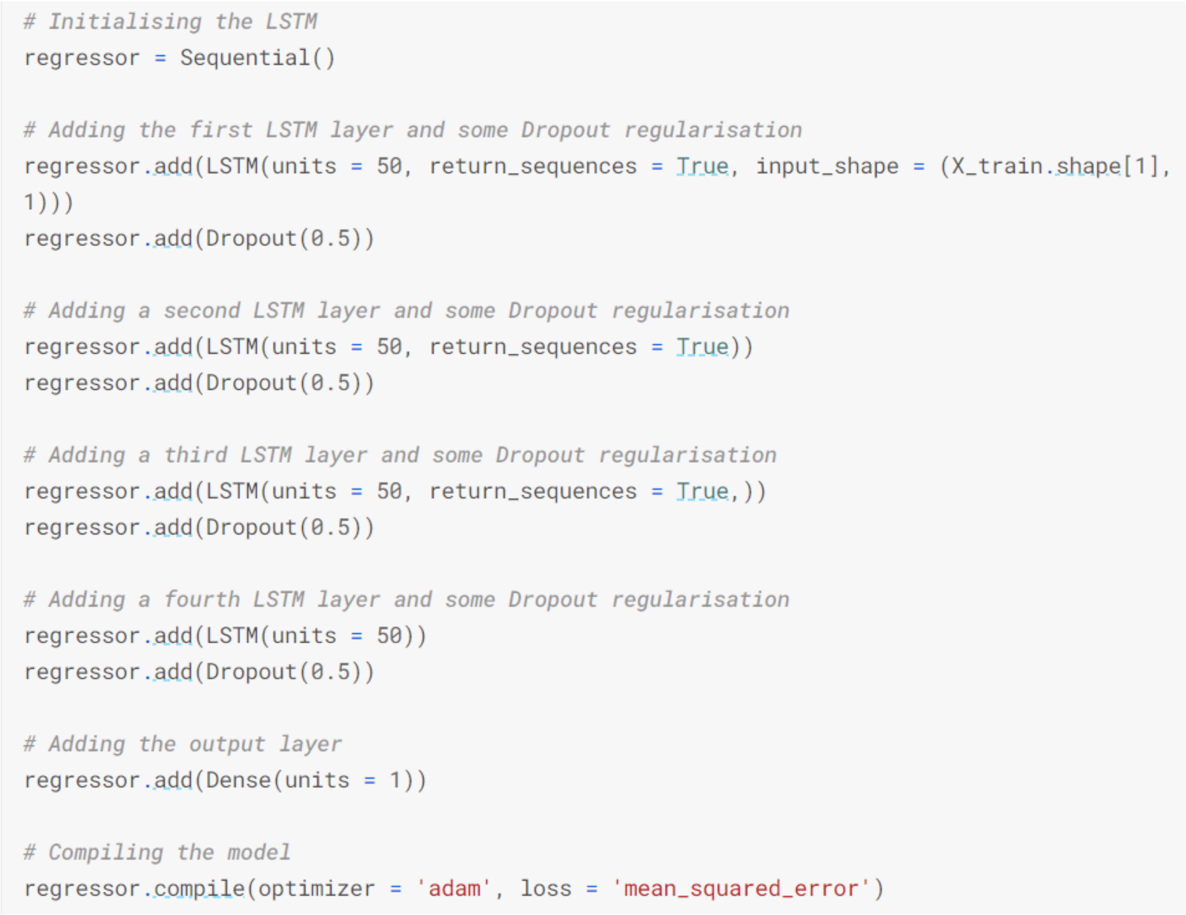
```
#reshape inputs to be [samples, timesteps, features] which is requred for LSTM

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)
```

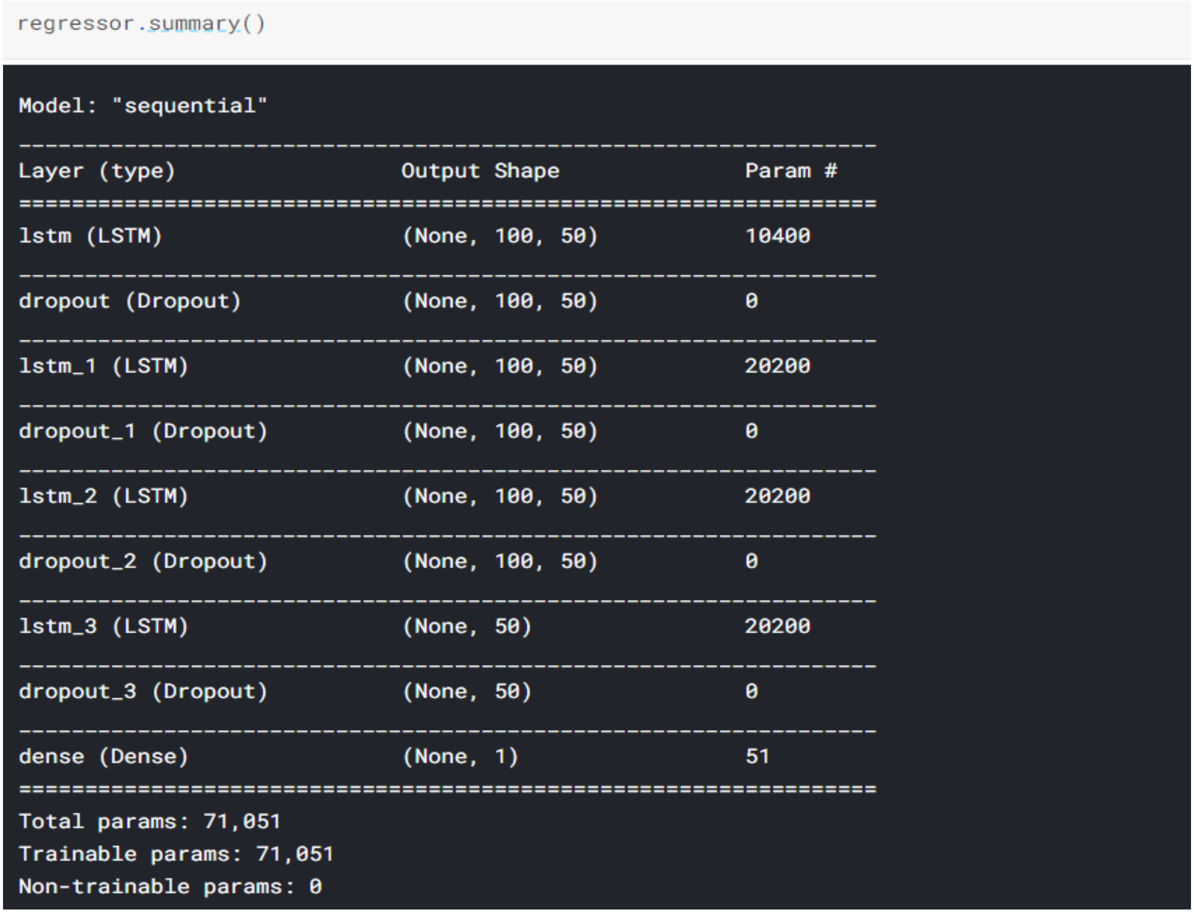


```
#Create LSTM Model
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import LSTM
from tensorflow.keras.layers import Dropout
```

After creating the LSTM model with tensorflow and initializing it, I have added four LSTM layers and adjusted the dropout to 0.5 so as to prevent overfitting and improve accuracy.

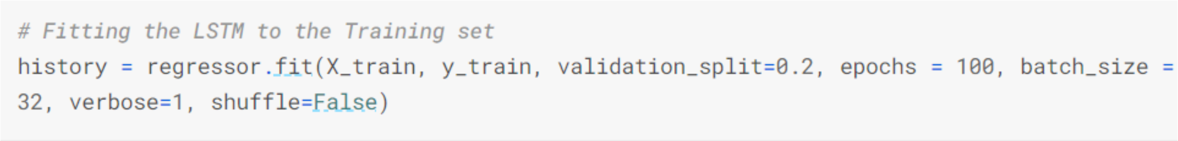


```
# Initialising the LSTM
regressor = Sequential()

# 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')
```

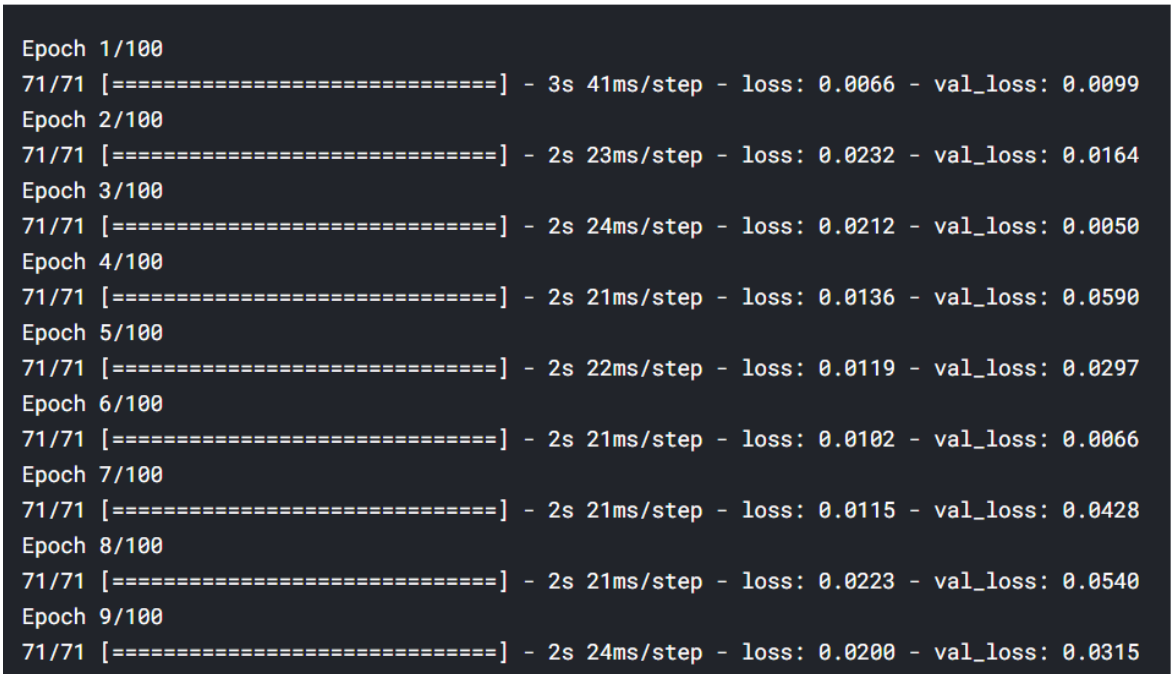


```
regressor.summary()
```

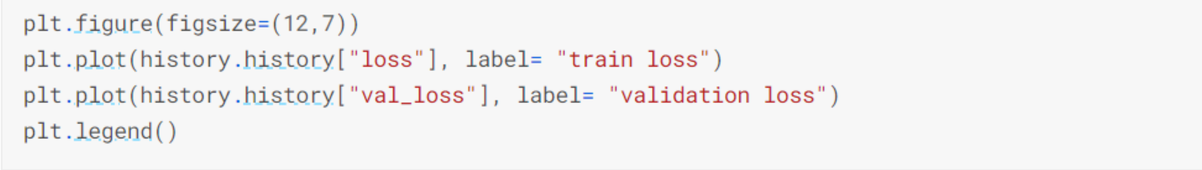
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 100, 50)           10400
_________________________________________________________________
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
```



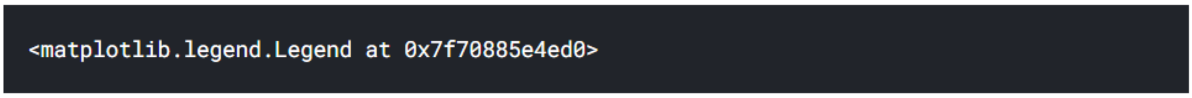
```python
# Fitting the LSTM to the Training set
history = regressor.fit(X_train, y_train, validation_split=0.2, epochs = 100, batch_size =
32, verbose=1, shuffle=False)
```



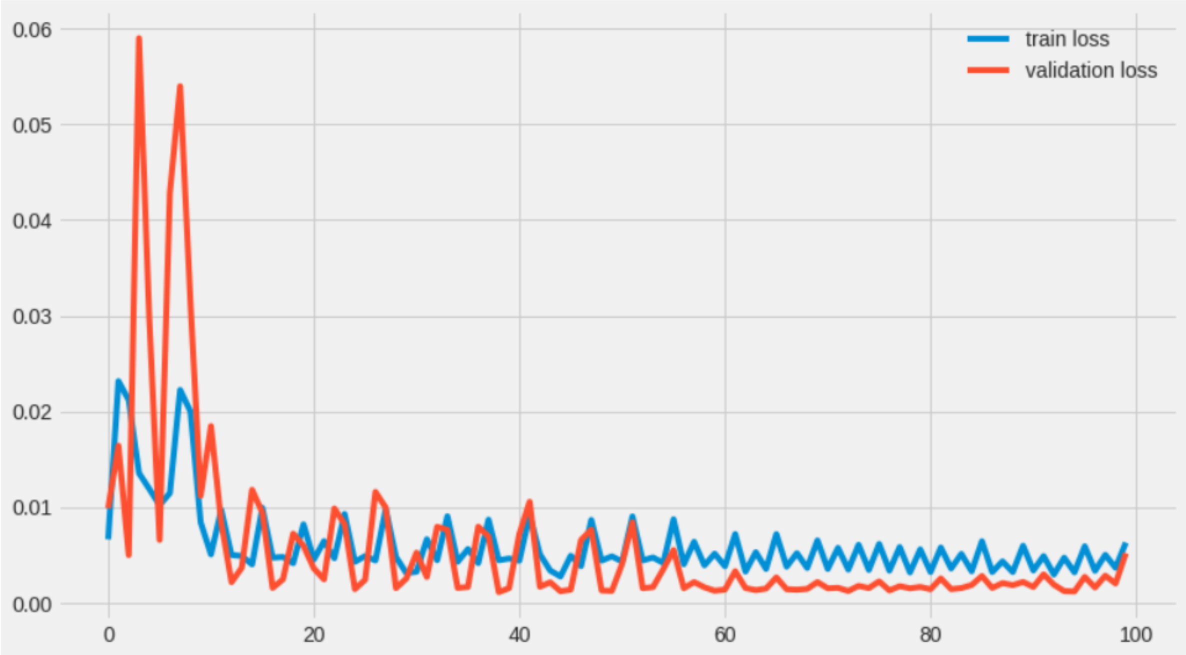
```
Epoch 1/100
71/71 [==============================] - 3s 41ms/step - loss: 0.0066 - val_loss: 0.0099
Epoch 2/100
71/71 [==============================] - 2s 23ms/step - loss: 0.0232 - val_loss: 0.0164
Epoch 3/100
71/71 [==============================] - 2s 24ms/step - loss: 0.0212 - val_loss: 0.0050
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
```

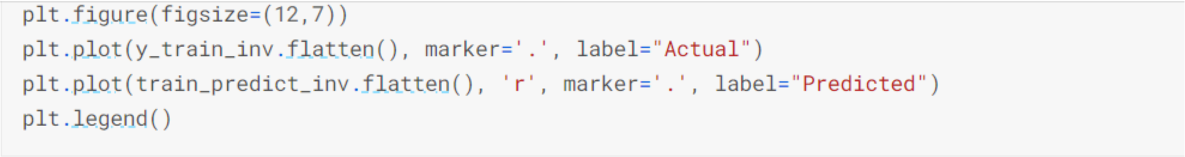


```python
plt.figure(figsize=(12,7))
plt.plot(history.history["loss"], label= "train loss")
plt.plot(history.history["val_loss"], label= "validation loss")
plt.legend()
```

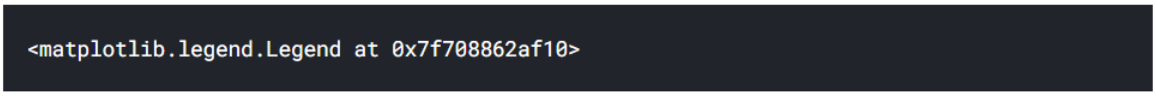


```
<matplotlib.legend.Legend at 0x7f70885e4ed0>
```

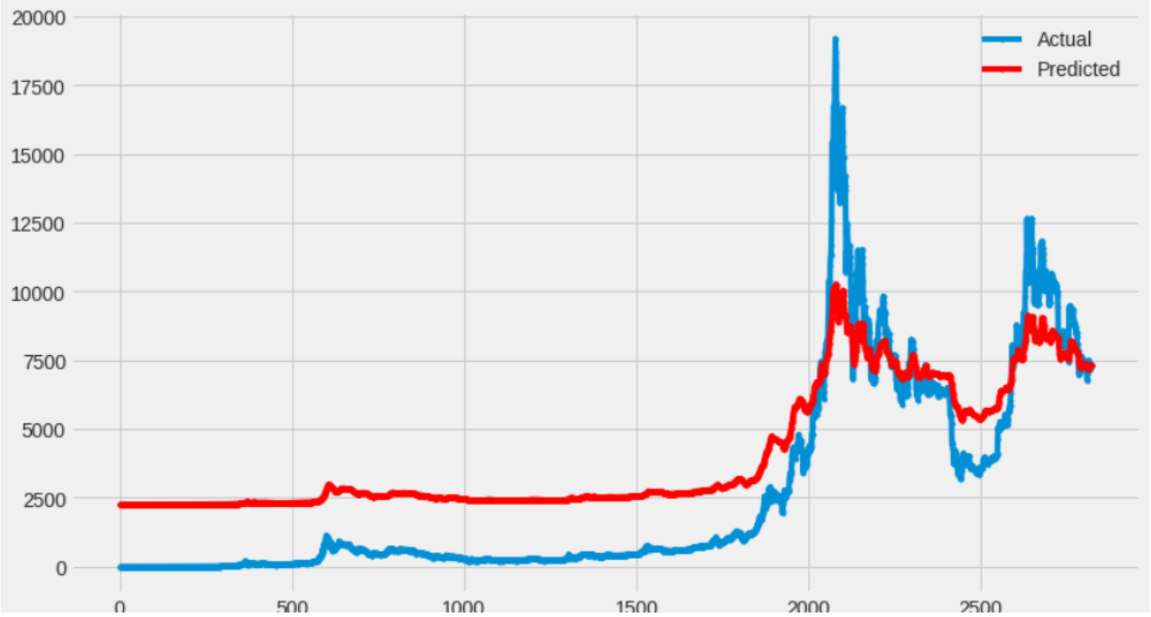




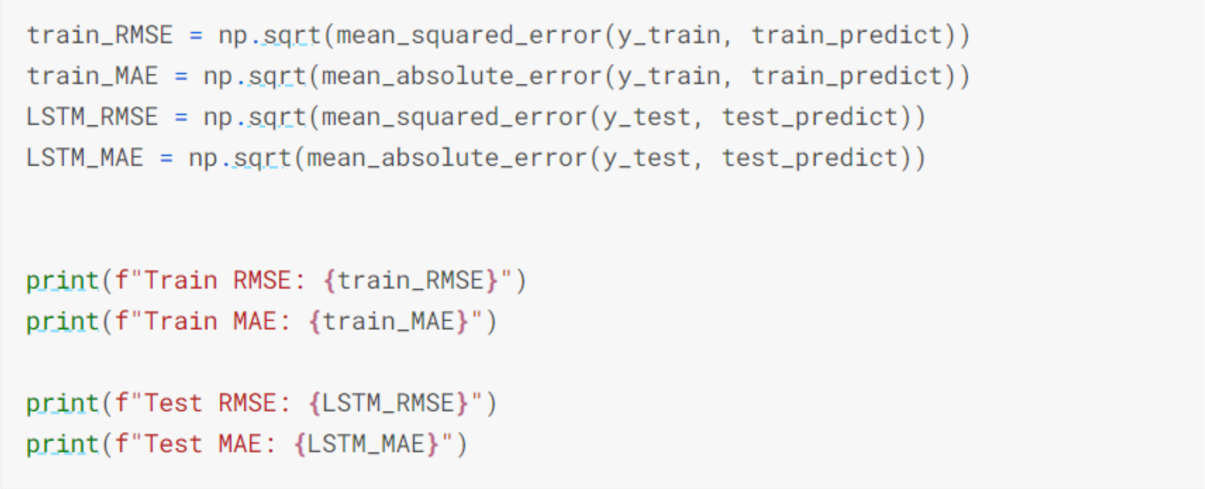
```
plt.figure(figsize=(12,7))
plt.plot(y_train_inv.flatten(), marker='.', label="Actual")
plt.plot(train_predict_inv.flatten(), 'r', marker='.', label="Predicted")
plt.legend()
```



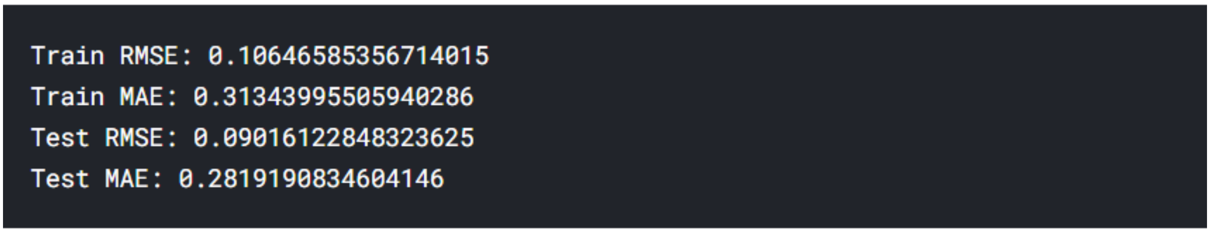
```
<matplotlib.legend.Legend at 0x7f708862af10>
```



Now we calculate the RMSE and MAE scores to see the accuracy of the model and later on conduct a comparitive analysis with other models.



```
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}")
```

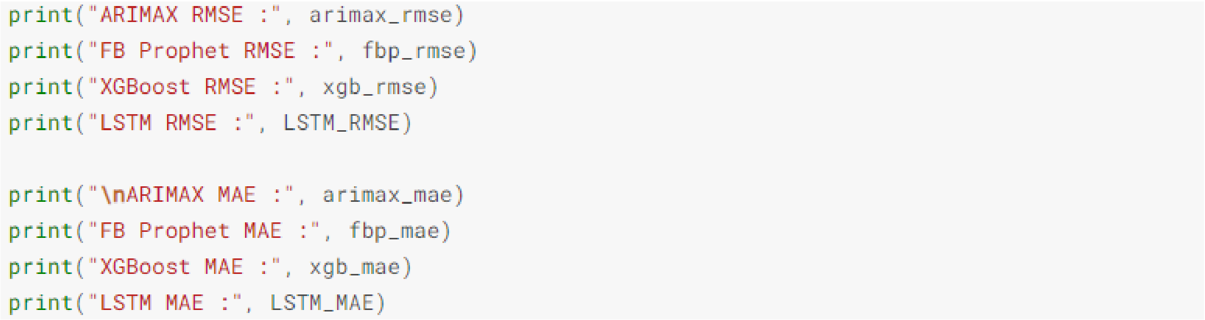


```
Train RMSE: 0.10646585356714015
Train MAE: 0.31343995505940286
Test RMSE: 0.09016122848323625
Test MAE: 0.281919083460414б
```

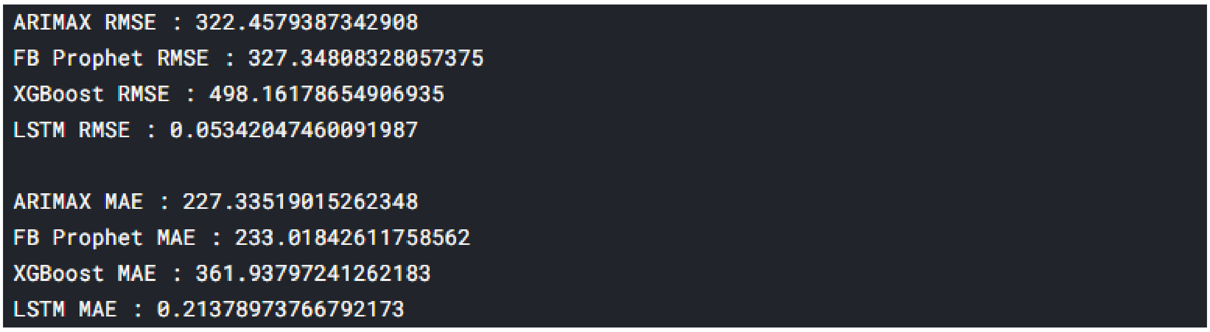
VII. COMPARATIVE STUDY / RESULTS AND DISCUSSION

In this section the results of the four tested methodologies are compared. I have executed 4 different algorithms – ARIMA model, Facebook Prophet, XGBoost model, and LSTM model – on the Bitcoin Price Prediction Dataset. From the literature survey, it was found out that LSTM worked best with time series datasets. So, I applied that model on our Bitcoin dataset as well and found the similar results as mentioned in the research papers. I have compared our models on the basis of error score. I 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.

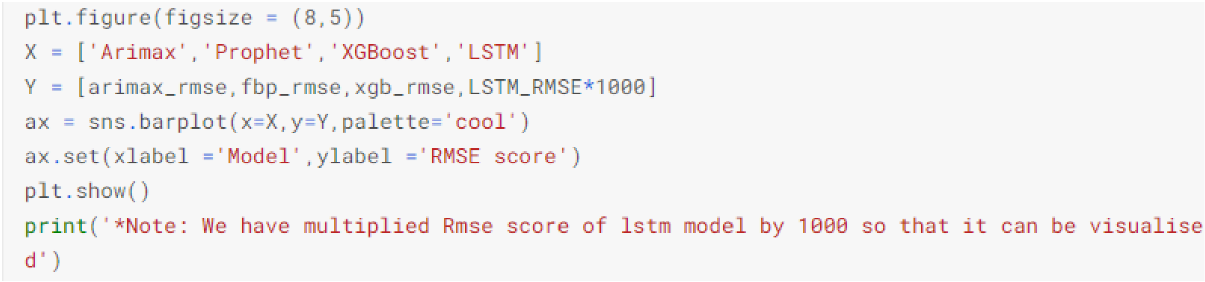


```
print("ARIMAX RMSE :", arimax_rmse)
print("FB Prophet RMSE :", fbp_rmse)
print("XGBoost RMSE :", xgb_rmse)
print("LSTM RMSE :", LSTM_RMSE)

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

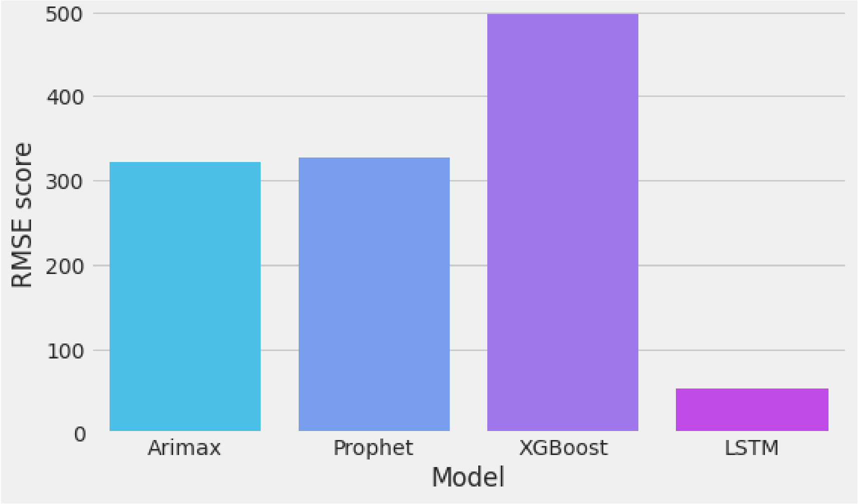


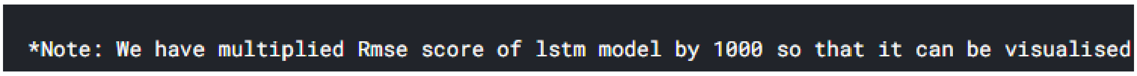
```
ARIMAX RMSE : 322.4579387342908
FB Prophet RMSE : 327.34808328057375
XGBoost RMSE : 498.16178654906935
LSTM RMSE : 0.05342047460091987

ARIMAX MAE : 227.33519015262348
FB Prophet MAE : 233.01842611758562
XGBoost MAE : 361.93797241262183
LSTM MAE : 0.21378973766792173
```

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.

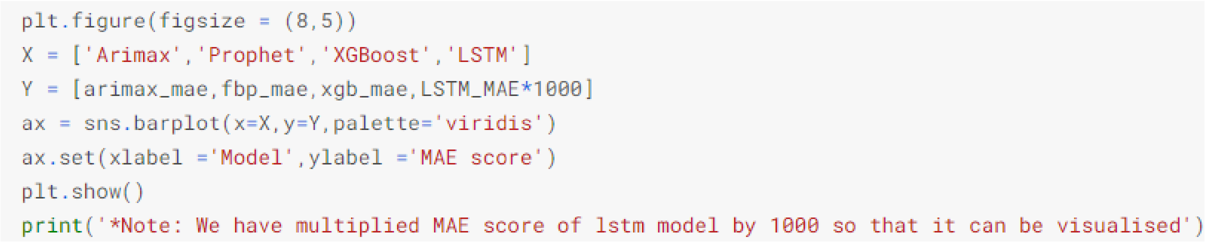


```
plt.figure(figsize = (8,5))
X = ['Arimax','Prophet','XGBoost','LSTM']
Y = [arimax_rmse,fbp_rmse,xgb_rmse,LSTM_RMSE*1000]
ax = sns.barplot(x=X,y=Y,palette='cool')
ax.set(xlabel ='Model',ylabel ='RMSE score')
plt.show()
print('*Note: We have multiplied Rmse score of lstm model by 1000 so that it can be visualise
d')
```

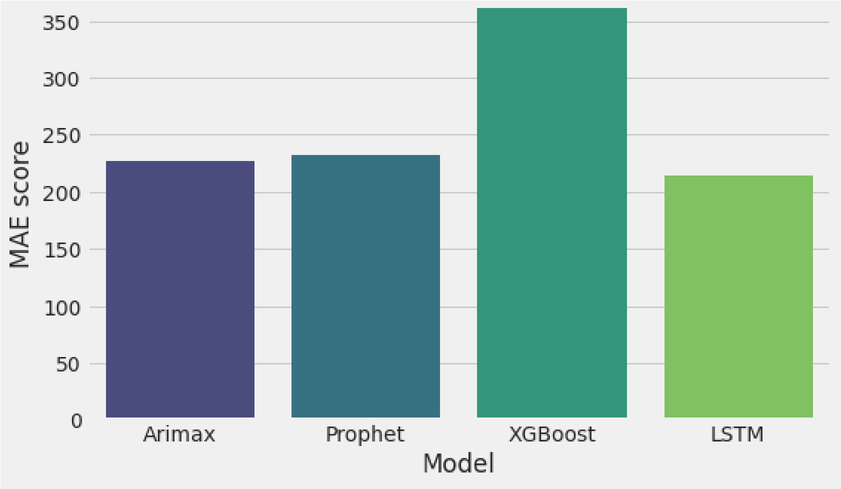


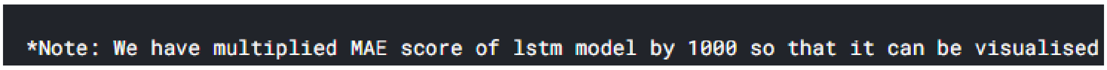


```
*Note: We have multiplied Rmse score of lstm model by 1000 so that it can be visualised
```



```
plt.figure(figsize = (8,5))
X = ['Arimax','Prophet','XGBoost','LSTM']
Y = [arimax_mae,fbp_mae,xgb_mae,LSTM_MAE*1000]
ax = sns.barplot(x=X,y=Y,palette='viridis')
ax.set(xlabel ='Model',ylabel ='MAE score')
plt.show()
print('*Note: We have multiplied MAE score of lstm model by 1000 so that it can be visualised')
```



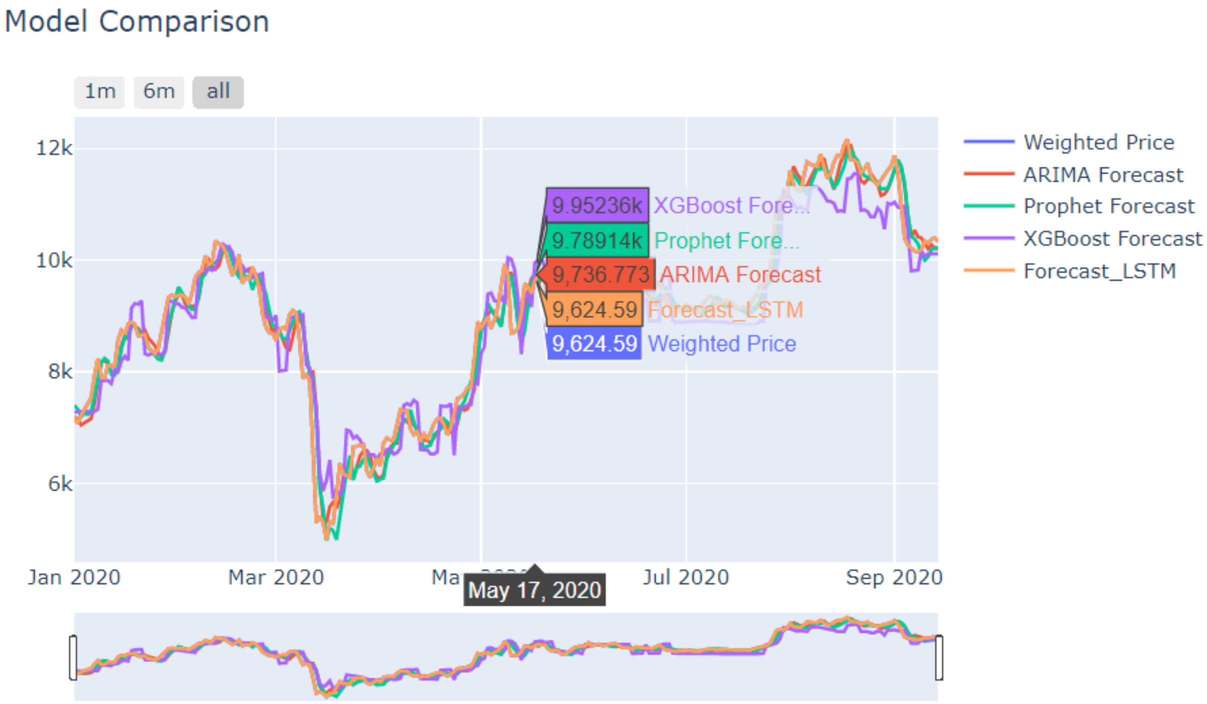


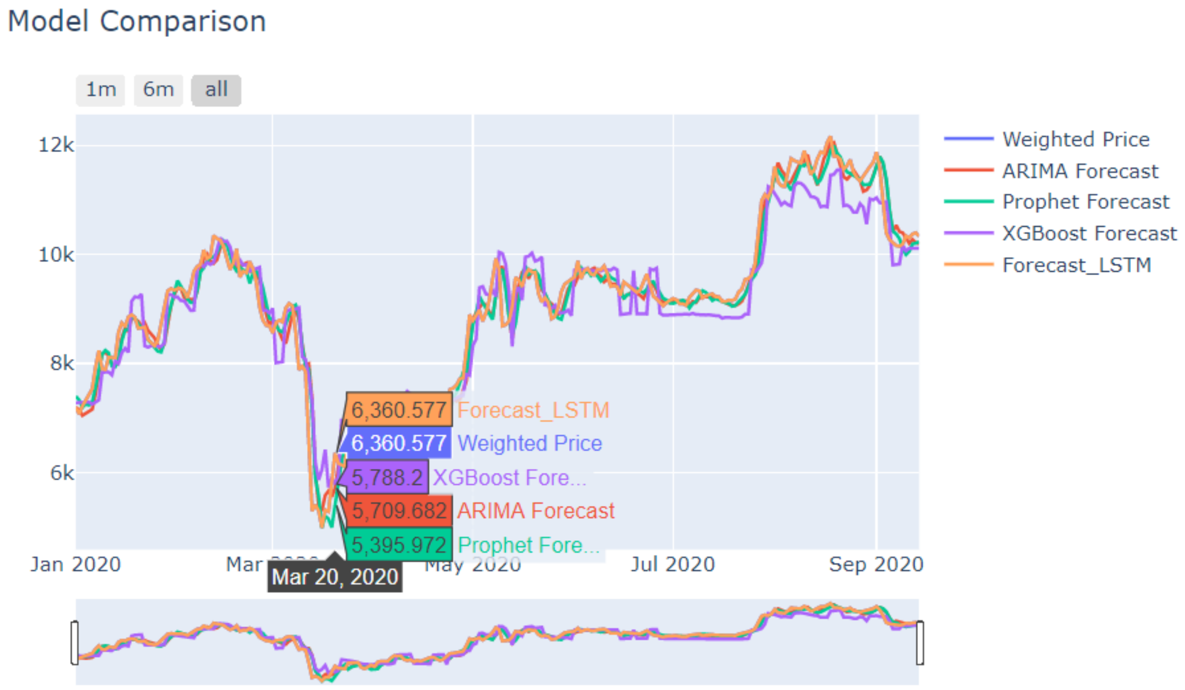
```
*Note: We have multiplied MAE score of lstm model by 1000 so that it can be visualised
```

XGBoost proved to be the worst model for the time series analysis. As an ensemble classifier, I 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, I 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.

I 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
1m
6m
all
12k
10k
8k
6k
Jan 2020
Mar 2020
May 2020
Jul 2020
Sep 2020
Weighted Price
ARIMA Forecast
Prophet Forecast
XGBoost Forecast
Forecast_LSTM
6,360.577 Forecast_LSTM
6,360.577 Weighted Price
5,788.2 XGBoost Fore...
5,709.682 ARIMA Forecast
5,395.972 Prophet Fore...
Mar 20, 2020

VIII. 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.

IX. REFERENCES

1. Siddhi Velankar, Sakshi Valecha, Shreya Maji, *Bitcoin Price Prediction using Machine Learning,* 20th International Conference on Advanced Communication Technology (ICACT), February 2018
2. Sean McNally, Jason Roche, Simon Caton, *Predicting the Price of Bitcoin Using Machine Learning,* 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP), March 2018
3. Thearasak Phaladisailoed, Thanisa Numnonda, *Machine Learning Models Comparison for Bitcoin Price Prediction,* 10th International Conference on Information Technology and Electrical Engineering (ICITEE), July 2018
4. Prachi Vivek Rane, Sudhir N. Dh age, *Systematic Erudition of Bitcoin Price Prediction using Machine Learning Techniques,* 5th International Conference on Advanced Computing & Communication Systems (ICACCS), March 2019
5. Karunya Rathan, Somarouthu Venkat Sai and Tubati Sai Manikanta, *Crypto-Currency price prediction using Decision Tree and Regression techniques,* 3rd International Conference on Trends in Electronics and Informatics (ICOEI), April 2019
6. Inbing Yao, Ke Xu, Qi Li, *Exploring the Influence of News Articles on Bitcoin Price with Machine Learning,* IEEE Symposium on Computers and Communications (ISCC), July 2019
7. Apoorva Aggarwal, Isha Gupta, Novesh Garg, Anurag Goel, *Comparison of Forcasting Ability betIen Backpropagation Network and ARIMA in the Prediction of Bitcoin Price,* International Symposium on Intelligent Signal Processing and Communication Systems (ISPACS), August 2019
8. I Made Wirawan, Triyanna Widiyaningtyas, Muchammad Maulana Hasan, *Short Term Prediction on Bitcoin Price Using ARIMA Method,* International Seminar on Application for Technology of Information and Communication (iSemantic), September 2019
9. Bohdan M. Pavlyshenko, *Bitcoin Price Predictive Modeling Using Expert Correction,* XIth International Scientific and Practical Conference on Electronics and Information Technologies (ELIT), September 2019
10. Leonardo Felizardo, Roberth Oliveira, Emilio Del-Moral-Hernandez, Fabio Cozman, *Comparative study of Bitcoin price prediction using WaveNets, Recurrent Neural Networks and other Machine Learning Methods,* 6th International Conference on Behavioral, Economic and Socio-Cultural Computing (BESC), October 2019

11. Ferdiansyah Ferdiansyah, Siti Hajar Othman, Raja Zahilah Raja Md Radzi, Deris Stiawan, Yoppy Sazaki, Usman Ependi, *A LSTM-Method for Bitcoin Price Prediction: A Case Study Yahoo Finance Stock Market*, International Conference on Electrical Engineering and Computer Science (ICECOS), October 2019
12. Aditi Mittal, Vipasha Dhiman, Ashi Singh, Chandra Prakash, *Short-Term Bitcoin Price Fluctuation Prediction Using Social Media and Ib Search Data*, TIlfth International Conference on Contemporary Computing (IC3), August 2019
13. Zhaoying Qiao, Tianrui Chai, Jialu Gu, Xinyi Zhou, Shuning Dai, Xuanming Zhang, *Singular Spectrum Analysis based Long Short-Term Memory for Predicting Bitcoin Price*, IEEE 7th International Conference on Computer Science and Network Technology (ICCSNT), October 2019
14. Muhammad Rizwan, Sanam Narejo, Moazzam Javed, *Bitcoin Price Prediction using Deep Learning Algorithm*, 13th International Conference on Mathematics, Actuarial Science, Computer Science and Statistics (MACS), December 2019
15. Chung-Chieh Chen, Jung-Hsin Chang, Fang-Cih Lin, Jui-Cheng Hung, Cheng-Shian Lin, Yi-Hsein Wang, *Comparison of Forcasting Ability betIen Backpropagation Network and ARIMA in the Prediction of Bitcoin Price*, International Symposium on Intelligent Signal Processing and Communication Systems (ISPACS), December 2019
16. Shubhankar Mohapatra, Nauman Ahmed, Paulo Alencar, *KryptoOracle: A Real-Time Cryptocurrency Price Prediction Platform Using Twitter Sentiments*, IEEE International Conference on Big Data (Big Data), December 2019
17. Yan Li, Ii Dai, *Bitcoin price forecasting method based on CNN-LSTM hybrid neural network model*, The 3rd Asian Conference on Artificial Intelligence Technology (ACAIT), Jamuary 2020
18. H Kavitha, Uttam Kumar Sinha, Surbhi S Jain, *Performance Evaluation of Machine Learning Algorithms for Bitcoin Price Prediction*, Fourth International Conference on Inventive Systems and Control (ICISC), January 2020
19. Rahmat Albariqi, Edi Winarko, *Prediction of Bitcoin Price Change using Neural Networks*, International Conference on Smart Technology and Applications (ICoSTA), February 2020
20. Patel Jay, Vasu Kalariya, Pushpendra Parmar, Sudeep Tanwar, Neeraj Kumar, Mamoun Alazab, *Stochastic Neural Networks for Cryptocurrency Price Prediction*, IEEE Access (Volume: 8), April 2020

21. M. Sivaram, E. Laxmi Lydia, Irina V. Pustokhina, Denis Alexandrovich Pustokhin, Mohamed Elhoseny, Gyanendra Prasad Joshi, K. Shankar, *An Optimal Least Square Support Vector Machine Based Earnings Prediction of Blockchain Financial Products*, IEEE Access (Volume: 8), June 2020
22. Dane Vanderbilt, Kun Xie, Inying Sun, *An Applied Study of RNN Models for Predicting Cryptocurrency Prices*, Issues in Information Systems, Volume 21, Issue 2, pp. 135-143, June 2020
23. Giulia Serafini, Ping Yi, Qingquan Zhang, Marco Brambilla, Jiayue Wang, YiIi Hu, Beibei Li, *Sentiment-Driven Price Prediction of the Bitcoin based on Statistical and Deep Learning Approaches*, International Joint Conference on Neural Networks (IJCNN), July 2020
24. Ahmed F. Ibrahim, Liam Corrigan, Rasha Kashef, *Predicting the Demand in Bitcoin Using Data Charts: A Convolutional Neural Networks Prediction Model*, IEEE Canadian Conference on Electrical and Computer Engineering (CCECE), August 2020
25. MyungJae Shin, David Mohaisen, Joongheon Kim, *Bitcoin Price Forecasting via Ensemble-based LSTM Deep Learning Networks*, International Conference on Information Networking (ICOIN), January 2021

Appendix

Link to project’s complete code (My Kaggle notebook): Click Here

```
import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
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_2021-03-31.csv")
bitstamp.head()
# Converting the Timestamp column from string to datetime
bitstamp['Timestamp'] = [datetime.fromtimestamp(x) for x in bitstamp['Timestamp']]
split_date = datetime(2020,9,14)
bitstamp = bitstamp.loc[bitstamp['Timestamp'] <= split_date]
bitstamp.head()
print('Dataset Shape: ',  bitstamp.shape)
bitstamp.set_index("Timestamp").Weighted_Price.plot(figsize=(10,7), title="Bitcoin Weighted Price", color='green')
#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)
missing_table
#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 bfi
ll/pad
a['interp'] = a['Weighted_Price'].interpolate()         # Imputation using int
erpolation

a.head()
# 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())
fill_missing(bitstamp)
#created a copy
bitstamp_non_indexed = bitstamp.copy()
bitstamp = bitstamp.set_index('Timestamp')
bitstamp.head()
#Resampling data
hourly_data = bitstamp.resample('1H').mean()
hourly_data = hourly_data.reset_index()
hourly_data.head()
#daily resampling
bitstamp_daily = bitstamp.resample("24H").mean()
bitstamp_daily.head()
from plotly.offline import init_notebook_mode, iplot
init_notebook_mode(connected=True)
import plotly.graph_objs as go
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")
#BTC Volume vs USD visualization
trace = go.Scattergl(
    y = bitstamp_daily['Volume_(BTC)'].astype(float),
    x = bitstamp_daily['Weighted_Price'].astype(float),
    mode = 'markers',
    marker = dict(
        line = dict(width = 1),
        color='#00FF00'
    )
)
layout = go.Layout(
    title='BTC Volume v/s USD',
    xaxis=dict(
        title='Weighted Price',
        titlefont=dict(
            family='Times New Roman, monospace',
            size=18
        )
    ),
    yaxis=dict(
        title='Volume BTC',
        titlefont=dict(
            family='Times New Roman, monospace',
            size=18
    )))
data = [trace]
fig = go.Figure(data=data, layout=layout)
iplot(fig, filename='compare_webgl')
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
fill_missing(bitstamp_daily)
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()
print("Dicky-
Fuller stationarity test: p=%f" % sm.tsa.adfuller(bitstamp_daily["Weighted_Pri
ce"])[1])
df = bitstamp_daily.set_index("Timestamp")
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.head()
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()
df_train = df[df.Timestamp < "2020"]
df_valid = df[df.Timestamp >= "2020"]

print('train shape :', df_train.shape)
print('validation shape :', df_valid.shape)
!pip install pmdarima
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_fe
atures], trace=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
df_valid[["Weighted_Price", "Forecast_ARIMAX"]].plot(figsize=(14, 7))
from sklearn.metrics import mean_squared_error, mean_absolute_error

print("RMSE of Auto ARIMAX:", np.sqrt(mean_squared_error(df_valid.Weighted_Pri
ce, df_valid.Forecast_ARIMAX)))
print("\nMAE of Auto ARIMAX:", mean_absolute_error(df_valid.Weighted_Price, df
_valid.Forecast_ARIMAX))
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']].renam
e({'Timestamp':'ds','Weighted_Price':'y'}, axis=1)
daily_data_fb.head()
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]
print("train data shape :", train_fb.shape)
print("test data shape :", test_fb.shape)
model_fbp = Prophet()
for feature in exogenous_features:
    model_fbp.add_regressor(feature)

model_fbp.fit(df_train[["Timestamp", "Weighted_Price"] + exogenous_features].r
ename(columns={"Timestamp": "ds", "Weighted_Price": "y"}))

forecast = model_fbp.predict(df_valid[["Timestamp", "Weighted_Price"] + exogen
ous_features].rename(columns={"Timestamp": "ds"}))
forecast.head()
df_valid["Forecast_Prophet"] = forecast.yhat.values
# Plot Our Predictions
fig1 = model_fbp.plot(forecast)
model_fbp.plot_components(forecast)
df_valid[["Weighted_Price", "Forecast_Prophet"]].plot(figsize=(14, 7))
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['F
orecast_Prophet']))

print(f"Prophet's MAE : {test_mae}")
print(f"Prophet's RMSE : {test_rmse}")
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()
# Hyper Parameter Optimization
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=1
0)],
 "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,
                )

model.fit(X_train, y_train)
print(f"Model Best Parameters : {model.best_params_}")
model.best_estimator_
model.score(X_test,y_test)
df_train['Predicted_Weighted_Price'] = model.predict(X_train)

df_train[['Weighted_Price','Predicted_Weighted_Price']].plot(figsize=(15, 5))
plt.show()
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))
overall_data[['Weighted_Price','Forecast_XGBoost']].plot(figsize=(15, 5))
train_mae = mean_absolute_error(df_train['Weighted_Price'], df_train['Predicte
d_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}")
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['F
orecast_XGBoost']))

print(f"test MAE : {test_mae}")
print(f"test RMSE : {test_rmse}")
price_series = bitstamp_daily.reset_index().Weighted_Price.values
price_series
price_series.shape
# Feature Scaling
from sklearn.preprocessing import MinMaxScaler

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

```
price_series_scaled, price_series_scaled.shape
train_data, test_data = price_series_scaled[0:2923], price_series_scaled[2923:
]
train_data.shape, test_data.shape
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
#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)
#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()
# 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)
plt.figure(figsize=(12,7))
plt.plot(history.history["loss"], label= "train loss")
plt.plot(history.history["val_loss"], label= "validation loss")
plt.legend()
#prediction
train_predict = regressor.predict(X_train)
test_predict = regressor.predict(X_test)
#transformation to original form
y_train_inv = scaler.inverse_transform(y_train.reshape(-1, 1))
y_test_inv = scaler.inverse_transform(y_test.reshape(-1, 1))
train_predict_inv = scaler.inverse_transform(train_predict)
test_predict_inv = scaler.inverse_transform(test_predict)
#Prediction on Training data
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()
#Prediction on Test data
plt.figure(figsize=(12,7))
plt.plot(y_test_inv.flatten(), marker='.', label="Actual")
plt.plot(test_predict_inv.flatten(), 'r', marker='.', label="Predicted")
plt.legend()
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}")
arimax_rmse = np.sqrt(mean_squared_error(df_valid['Weighted_Price'], df_valid[
'Forecast_ARIMAX']))
fbp_rmse = np.sqrt(mean_squared_error(df_valid['Weighted_Price'], df_valid['Fo
recast_Prophet']))
```

```
xgb_rmse = np.sqrt(mean_squared_error(df_valid['Weighted_Price'], df_valid['Forecast_XGBoost']))

arimax_mae = mean_absolute_error(df_valid['Weighted_Price'], df_valid['Forecast_ARIMAX'])
fbp_mae = mean_absolute_error(df_valid['Weighted_Price'], df_valid['Forecast_Prophet'])
xgb_mae = mean_absolute_error(df_valid['Weighted_Price'], df_valid['Forecast_XGBoost'])
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)
plt.figure(figsize = (8,5))
X = ['Arimax','Prophet','XGBoost','LSTM']
Y = [arimax_rmse,fbp_rmse,xgb_rmse,LSTM_RMSE*1000]
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 1000 so that it can be visualised')
plt.figure(figsize = (8,5))
X = ['Arimax','Prophet','XGBoost','LSTM']
Y = [arimax_mae,fbp_mae,xgb_mae,LSTM_MAE*1000]
ax = sns.barplot(x=X,y=Y,palette='viridis')
ax.set(xlabel ='Model',ylabel ='MAE score')
plt.show()
print('*Note: We have multiplied MAE score of lstm model by 1000 so that it can be visualised')
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")
```