KONKAN GYANPEETH COLLEGE OF ENGINEERING, KARJAT
Affiliated to University of Mumbai, Approved by AICTE, New Delhi.

Crypto-Currency (BITCOIN) Price Prediction

Project Members :

- Shivprakash Vishwakarma(58)
- Swaraj Kondlekar(33)
- Afzal Ali Siddique(49)

Under the Guidance of

Prof. A.S Kunte

Contents

ABSTRACT

- The decentralization of cryptocurrencies has greatly reduced the level of central control over them, impacting international relations and trade. Further, wide fluctuations in crypto Currency price indicate an urgent need for an accurate way to forecast this price.
- This project proposes a method to predict crypto Currency price by considering various factors such as market cap, volume, circulating supply, and maximum supply based on deep learning techniques such as the long short term memory (LSTM),which are effective learning models for training data, with the LSTM being better at recognizing longer-term associations.

INTRODUCTION

- Crypto Currency, is a decentralized digital or virtual Currency. Use of cryptography for security makes it difficult to counterfeit. Cryptocurrencies started to gain attention in 2013 and since then witnessed a significant number of transactions and hence price fluctuations.
- The crypto Currency market is just similar to Crypto-Currency market. It has gained public attention and so effective prediction of price movement of crypto Currency will aid public to invest profitably in the system.
- This project tries to predict the price of Cryptocurrencies like Bitcoin. Deep learning techniques were implemented and the use of Long Short Term Memory (LSTM) network proved very efficient in predicting the prices of digital currencies.

Objectives

- To implement LSTM model which will be train on available historical data.
- To forecast the price of Crypto Currency(Bitcoin).
- To integrate this model in order to develop simple system through which user can interact and perform predictions to predict the future price of bitcoin.
- To provide insights of data in graphical format such as bar charts,line chart, comparison between predicted vs actual, accuracy of prediction etc.

Purpose

- The Cryptocurrencies like Bitcoin are influenced by many uncertainties factor such as
  - Political issue
  - The economic issue at impacted to local or global levels
- For the market, we can analyze with any techniques such as technical indicator, price movements, and market technical analysis.
- To solve the problem regarding the fluctuations there’s a need automation tool for prediction to help investors decide for bitcoin or other crypto Currency market investment.
- Nowadays the automation tools are usually used in common Crypto Currency market predictions, and we can do the same works and strategy on this domain cryptocurrencies.
- The purpose of this study is to predict the price of Bitcoin and changes therein using the LSTM model and to build a system which will help common people, investors or business analyst to invest in digital currencies as bitcoin by predicting their future values and to visualize and analyse the pattern of digital currencies

Scope

- We are visualising historical bitcoin data.
- The Missing values in the dataset is handled by using Imputation Technique i.e Interpolation which imputes the missing values in time-series.
- Exploratory data analysis is done by visualising using lag plots & further time resampling i.e aggregate data into a defined time period, such as by month or by quarter.
- Model Selection and Model Building.
- Prediction using LSTMs

Implementation

Dataset Includes Historical bitcoin market data at 1-min intervals for select bitcoin exchanges where trading takes place. It consists time period of Jan 2012 to September 2020, with minute to minute updates of OHLC (Open, High, Low, Close), Volume in BTC and indicated Currency, and weighted bitcoin price.

- The **Open** and **Close** columns indicate the opening and closing price on a particular day.
- The **High** and **Low** columns provide the highest and the lowest price on a particular day, respectively.
- The **Volume** column tells us the total volume of traded on a particular day.
- The **Weighted price** is a trading benchmark used by traders that gives the weighted price a security has traded at throughout the day, based on both volume and price. It is important because it provides traders with insight into both the trend and value of a security.

Handling Missing Values in Time-series Data

- It is very common for a time-series data to have missing data. The first step is to detect the count/percentage of missing values in every column of the dataset. This will give an idea about the distribution of missing values.

```
#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
```

	Total Missing Values	Missing %
Timestamp	0	0.000000
Open	1241716	27.157616
High	1241716	27.157616
Low	1241716	27.157616
Close	1241716	27.157616
Volume_(BTC)	1241716	27.157616
Volume_(Currency)	1241716	27.157616
Weighted_Price	1241716	27.157616

Imputation refers to replacing missing data with substituted values.There are a lot of ways in which the missing values can be imputed depending upon the nature of the problem and data. Depending upon the nature of the problem, imputation techniques can be broadly they can be classified as follows:

Basic Imputation Techniques for Time-Series data

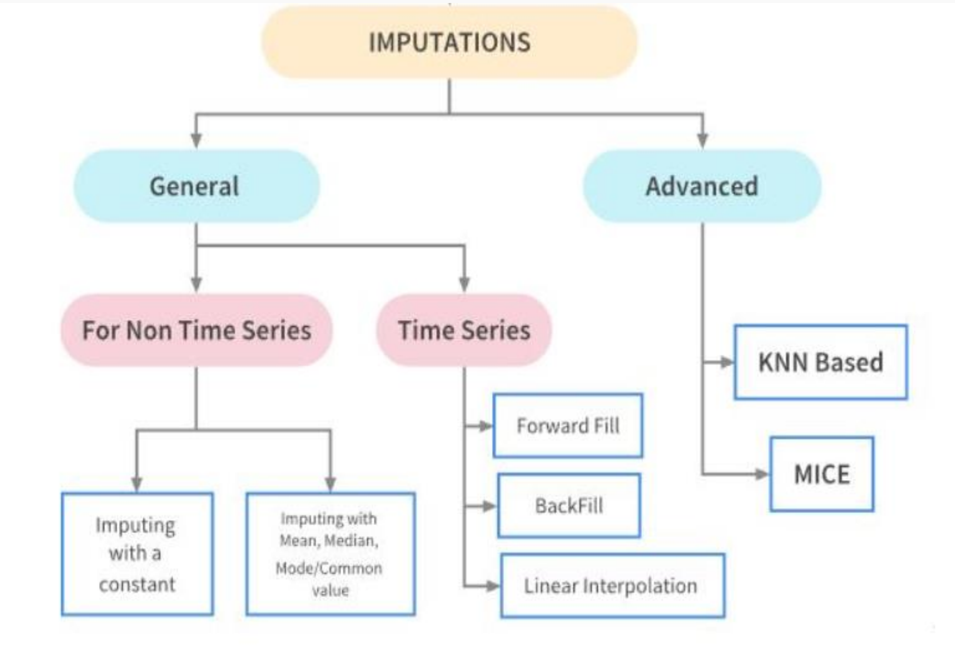
- 'ffill' or 'pad'

– Replace NaNs with last observed value

- 'bfill' or 'backfill'

– Replace NaNs with next observed value

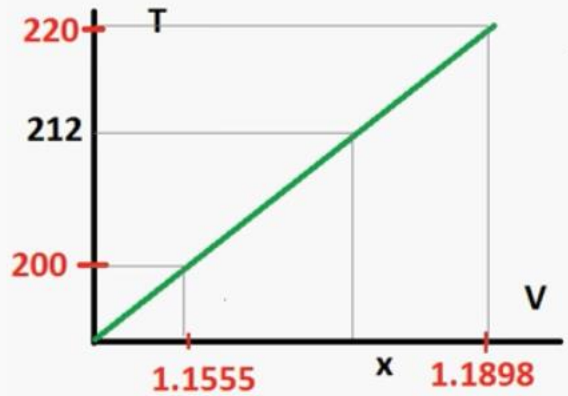
- Linear interpolation method



Imputation using Linear Interpolation method

Time series data has a lot of variations against time. Hence, imputing using backfill and forward fill isn't the best possible solution to address the missing value problem. A more apt alternative would be to use interpolation methods, where the values are filled with incrementing or decrementing values.

Linear interpolation is an imputation technique that assumes a linear relationship between data points and utilises non-missing values from adjacent data points to compute a value for a missing data point.



$$\frac{220 - 200}{212 - 200} = \frac{1.1898 - 1.1555}{x - 1.1555}$$

$$x = \frac{(1.1898 - 1.1555)(212 - 200)}{(220 - 200)} + 1.1555$$

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

    df['Volume_(BTC)'] = df['Volume_(BTC)'].interpolate()
    df['Volume_(Currency)'] = df['Volume_(Currency)'].interpolate()
    df['High'] = df['High'].interpolate()
    df['Low'] = df['Low'].interpolate()

    print(df.head())
    print(df.isnull().sum())
```

```
fill_missing(bitstamp)
```

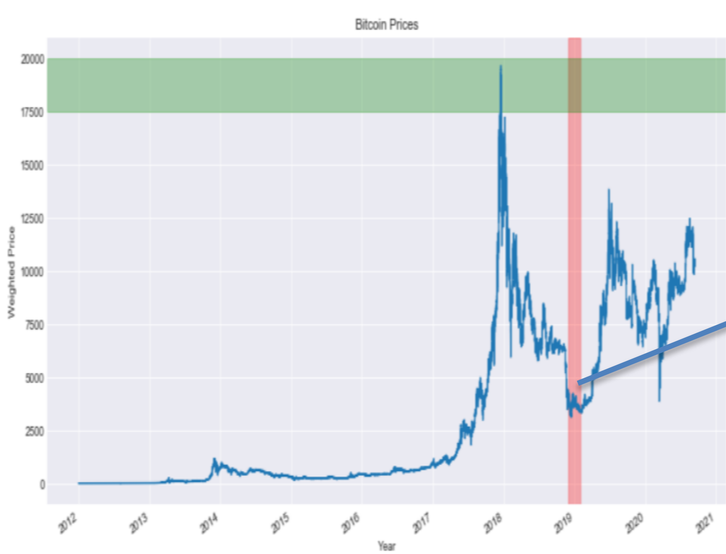
POST-IMPUTATION :

```
            Timestamp  Open  High   Low  Close  Volume_(BTC)
0 2011-12-31 13:22:00  4.39  4.39  4.39   4.39      0.455581
1 2011-12-31 13:23:00  4.39  4.39  4.39   4.39      0.555046
2 2011-12-31 13:24:00  4.39  4.39  4.39   4.39      0.654511
3 2011-12-31 13:25:00  4.39  4.39  4.39   4.39      0.753977
4 2011-12-31 13:26:00  4.39  4.39  4.39   4.39      0.853442

   Volume_(Currency)  Weighted_Price
0           2.000000            4.39
1           2.436653            4.39
2           2.873305            4.39
3           3.309958            4.39
4           3.746611            4.39
Timestamp            0
Open                 0
High                 0
Low                  0
Close                0
Volume_(BTC)         0
Volume_(Currency)    0
Weighted_Price       0
dtype: int64
```

Exploratory Data Analysis

When working with time-series data, a lot can be revealed through visualizing it. It is possible to add markers in the plot to help emphasize the specific observations or specific events in the time series.

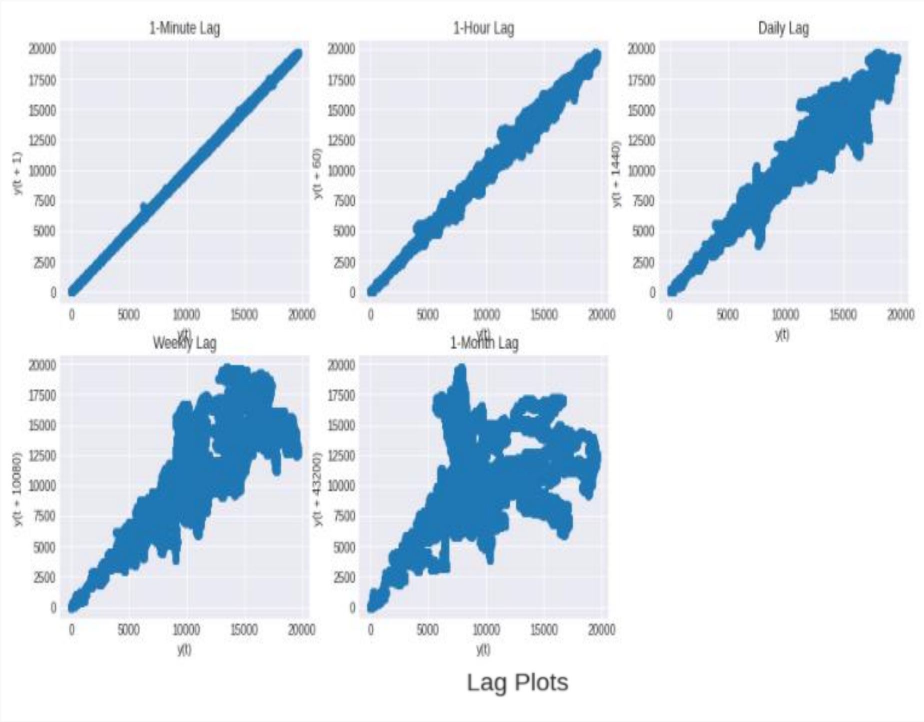
```python
ax = bitstamp['Weighted_Price'].plot(title='Bitcoin Prices', grid=True, figsize=(14,7))
ax.set_xlabel('Year')
ax.set_ylabel('Weighted Price')

ax.axvspan('2018-12-01','2019-01-31',color='red', alpha=0.3)
ax.axhspan(17500,20000, color='green',alpha=0.3)
```



- Lag plot are used to observe the autocorrelation. These are crucial when we try to correct the trend and stationarity and we have to use smoothing functions. Lag plot helps us to understand the data better.

Visualizing using Lag Plots

```
plt.figure(figsize=(15,12))
plt.suptitle('Lag Plots', fontsize=22)

plt.subplot(3,3,1)
pd.plotting.lag_plot(bitstamp['Weighted_Price'], lag=1) #minute lag
plt.title('1-Minute Lag')

plt.subplot(3,3,2)
pd.plotting.lag_plot(bitstamp['Weighted_Price'], lag=60) #hourley lag
plt.title('1-Hour Lag')

plt.subplot(3,3,3)
pd.plotting.lag_plot(bitstamp['Weighted_Price'], lag=1440) #Daily lag
plt.title('Daily Lag')

plt.subplot(3,3,4)
pd.plotting.lag_plot(bitstamp['Weighted_Price'], lag=10080) #weekly lag
plt.title('Weekly Lag')

plt.subplot(3,3,5)
pd.plotting.lag_plot(bitstamp['Weighted_Price'], lag=43200) #month lag
plt.title('1-Month Lag')

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



We can see that there is a positive correlation for minute, hour and daily lag plots. We observe absolutely no correlation for month lag plots.
It makes sense to re-sample our data atmost at the Daily level, thereby preserving the autocorrelation as well.

Time resampling

- Examining stock price data for every single day isn't of much use to financial institutions, who are more interested in spotting market trends. To make it easier, we use a process called time resampling to aggregate data into a defined time period, such as by month or by quarter. Institutions can then see an overview of stock prices and make decisions according to these trends.
- The pandas library has a .resample() function which resamples such time series data. The resample method in pandas is similar to its groupby method as it is essentially grouping according to a certain time span.

The resample() function looks like this:

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

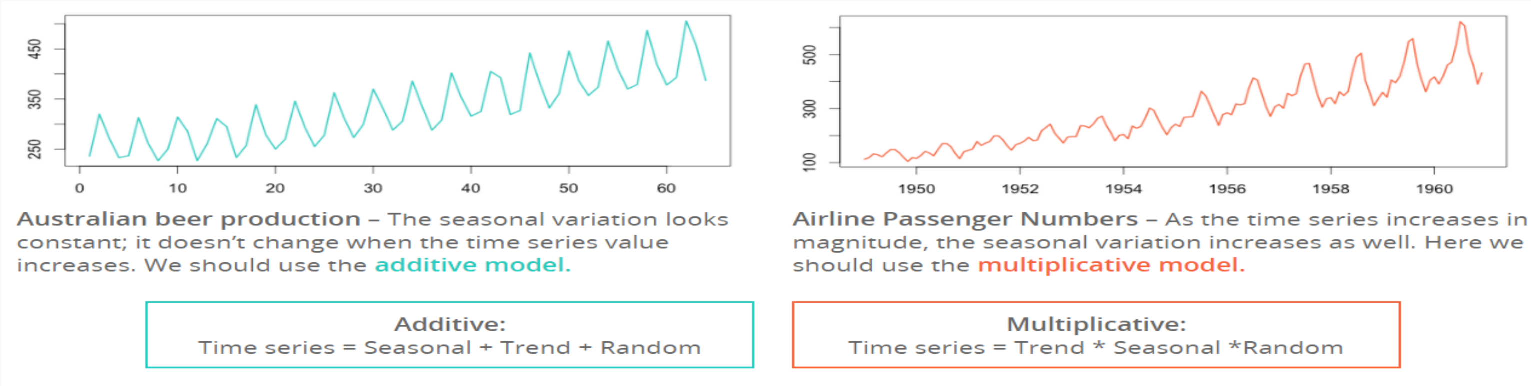
To summarize what happened above:

- data.resample() is used to resample the stock data.
- The '24H' stands for daily frequency, and denotes the offset values by which we want to resample the data.
- mean() indicates that we want the average stock price during this period.

Time Series Decomposition

- **Time series decomposition** involves thinking of a **series** as a combination of level, trend, seasonality, and noise components. **Decomposition** provides a useful abstract model for thinking about **time series** generally and for better understanding problems during **time series** analysis and forecasting.
- Decomposition is often used to remove the seasonal effect from a time series. It provides a cleaner way to understand trends. For instance, lower ice cream sales during winter don’t necessarily mean a company is performing poorly. To know whether or not this is the case, we need to remove the seasonality from the time series.

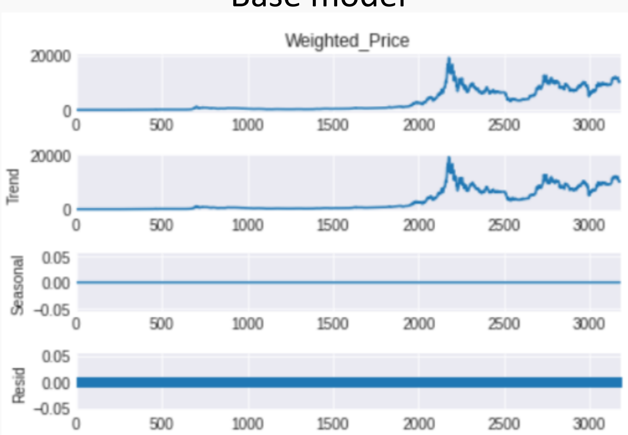
- We can decompose a time series into trend, seasonal and remainder components.



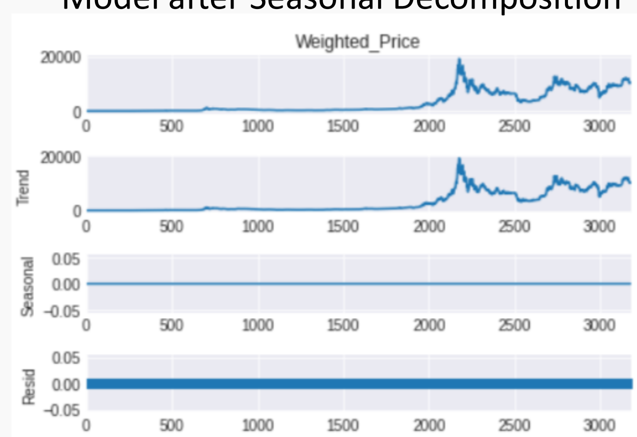
Performing Time Decomposition

```
plt.figure(figsize=(15,12))
series = bitstamp_daily.Weighted_Price
result = seasonal_decompose(series, model='additive',period=1)
result.plot()
```

Base model



Model after Seasonal Decomposition



Post time series decomposition we don't observe any seasonality. Also, there is no constant mean, variance and covariance, hence the series is **Non Stationary.**

Stationarity is an important concept in time series analysis as it means that the statistical properties of a a time series do not change over time.

We will perform statistical tests like KPSS and ADF to confirm our understanding.

KPSS Test

- The KPSS test, short for, Kwiatkowski-Phillips-Schmidt-Shin (KPSS), is a type of Unit root test that tests for the stationarity of a given series around a deterministic trend.
- Here, the null hypothesis is that the series is **stationary**.
- That is, if p-value is < signif level (say 0.05), then the series is non-stationary and vice versa
- ‘**ct**’ : The data is stationary around a trend.

```
stats, p, lags, critical_values = kpss(series, 'ct')
```

```
print(f'Test Statistics : {stats}')
print(f'p-value : {p}')
print(f'Critical Values : {critical_values}')

if p < 0.05:
    print('Series is not Stationary')
else:
    print('Series is Stationary')
```

```
p-value is smaller than the indicated p-value

Test Statistics : 0.9719743430417129
p-value : 0.01
Critical Values : {'10%': 0.119, '5%': 0.146, '2.5%': 0.176, '1%': 0.216}
Series is not Stationary
```

ADF Test

- The null hypothesis of the test is the presence of unit root, that is, the series is non-stationary.

```
def adf_test(timeseries):
    print ('Results of Dickey-Fuller Test:')
    dftest = adfuller(timeseries, autolag='AIC')
    dfoutput = pd.Series(dftest[0:4], index=['Test Statistic','p-value','#Lags Used','Number of
Observations Used'])
    for key,value in dftest[4].items():
       dfoutput['Critical Value (%s)'%key] = value

    print (dfoutput)

    if p > 0.05:
        print('Series is not Stationary')
    else:
        print('Series is Stationary')

adf_test(series)
```

```
Results of Dickey-Fuller Test:
Test Statistic                   -1.257922
p-value                           0.648197
#Lags Used                       29.000000
Number of Observations Used    3151.000000
Critical Value (1%)              -3.432427
Critical Value (5%)              -2.862458
Critical Value (10%)             -2.567259
dtype: float64
Series is Stationary
```

KPSS says series is not stationary and ADF says series is stationary. It means series is **difference stationary**, we will use **differencing** to make series stationary.

Model Building

- As for Model Selection We studied some research papers based on time-series data prediction and we encountered some models namely ARIMA,XGBOOST,LSTM etc.
- On the basis of research papers as well as comparing the models accuracy we made a decision of using LSTM to build our model due to high performance and accuracy than other models while dealing with time-series data

Before Model Building one important step is **CROSS-VALIDATION**.

- To measure the performance of our forecasting model, We typically want to split the time series into a training period and a validation period. This is called fixed partitioning

- If the time series has some seasonality, you generally want to ensure that each period contains a whole number of seasons. For example, one year, or two years, or three years, if the time series has a yearly seasonality. You generally don't want one year and a half, or else some months will be represented more than others.

we will opt for a ***hold-out based validation***.
Hold-out is used very frequently with time-series data. In this case, we will select all the data for 2020 as a hold-out and train our model on all the data from 2012 to 2019.

```python
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)
```

Long Short Term Memory (LSTM) **Networks**

- **Long Short Term Memory networks** – usually just called "LSTMs" – are a special kind of RNN, capable of learning long-term dependencies.
- LSTMs are explicitly designed to avoid the long-term dependency problem. Remembering information for long periods of time is practically their default behavior, not something they struggle to learn like RNNs!
- All recurrent neural networks have the form of a chain of repeating modules of neural network. In standard RNNs, this repeating module will have a very simple structure, such as a single tanh layer.
- Also, they don't suffer from problems like **vanishing/exploding gradient descent**.

Feature Scaling or Standardization: It is a step of Data Pre Processing which is applied to independent variables or **features** of data. It basically helps to normalise the data within a particular range. Sometimes, it also helps in speeding up the calculations in an algorithm.

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

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

```python
price_series_scaled, price_series_scaled.shape
```

```
(array([[6.08556702e-06],
        [2.11105388e-05],
        [3.36730766e-05],
        ...,
        [5.40552894e-01],
        [5.41733970e-01],
        [5.38430384e-01]]),
 (3181, 1))
```

We Split the dataset into train_data and test_data:

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

Then we are splitting our dataset into a window of particular Time-step:

```
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,))
```

Now Reshaping our inputs as LSTM takes input with timesteps and features and here our Feature value is 1 i.e our values lies between 0 & 1

```
#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)
```

```
print(y_train.shape)
print(y_test.shape)
```

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

```python
#Create Stacked 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
```

```python
# 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(0.2))

# Adding a second LSTM layer and some Dropout regularisation
regressor.add(LSTM(units = 50, return_sequences = True))
regressor.add(Dropout(0.2))

# Adding a third LSTM layer and some Dropout regularisation
regressor.add(LSTM(units = 50, return_sequences = True))
regressor.add(Dropout(0.2))

# Adding a fourth LSTM layer and some Dropout regularisation
regressor.add(LSTM(units = 50))
regressor.add(Dropout(0.2))

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

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

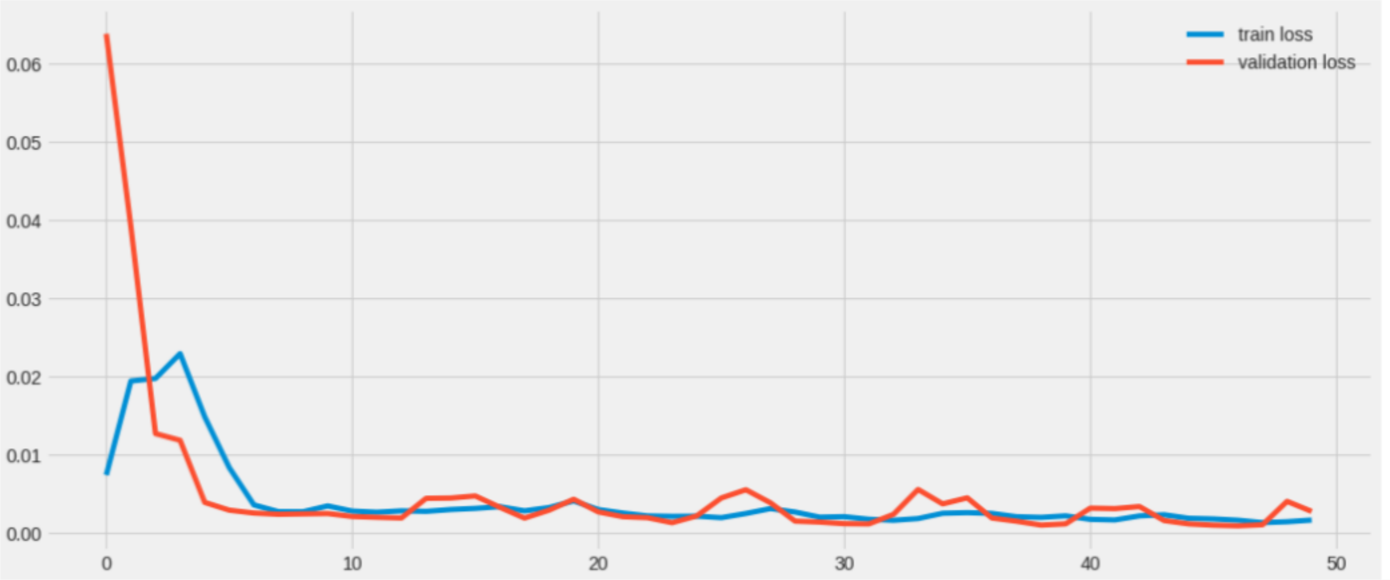
```python
# Fitting the RNN to the Training set
history = regressor.fit(X_train, y_train, validation_split=0.1, epochs = 50, batch_size = 32, v
erbose=1, shuffle=False)
```

```
Epoch 1/50
80/80 [==============================] - 17s 212ms/step - loss: 0.0074 - val_loss: 0.0638
Epoch 2/50
80/80 [==============================] - 15s 192ms/step - loss: 0.0194 - val_loss: 0.0391
Epoch 3/50
80/80 [==============================] - 16s 194ms/step - loss: 0.0197 - val_loss: 0.0127
Epoch 4/50
80/80 [==============================] - 16s 202ms/step - loss: 0.0229 - val_loss: 0.0118
Epoch 5/50
80/80 [==============================] - 16s 199ms/step - loss: 0.0148 - val_loss: 0.0039
Epoch 6/50
80/80 [==============================] - 17s 209ms/step - loss: 0.0083 - val_loss: 0.0029
Epoch 7/50
80/80 [==============================] - 16s 197ms/step - loss: 0.0035 - val_loss: 0.0025
Epoch 8/50
80/80 [==============================] - 16s 197ms/step - loss: 0.0027 - val_loss: 0.0023
Epoch 9/50
80/80 [==============================] - 15s 193ms/step - loss: 0.0027 - val_loss: 0.0024
Epoch 10/50
80/80 [==============================] - 16s 196ms/step - loss: 0.0034 - val_loss: 0.0024
Epoch 11/50
80/80 [==============================] - 16s 198ms/step - loss: 0.0028 - val_loss: 0.0021
```

Through Training and Validation loss we can see that the model is not overfitting i.e its pretty good for Prediction.

```
plt.figure(figsize=(16,7))
plt.plot(history.history["loss"], label= "train loss")
plt.plot(history.history["val_loss"], label= "validation loss")
plt.legend()
```

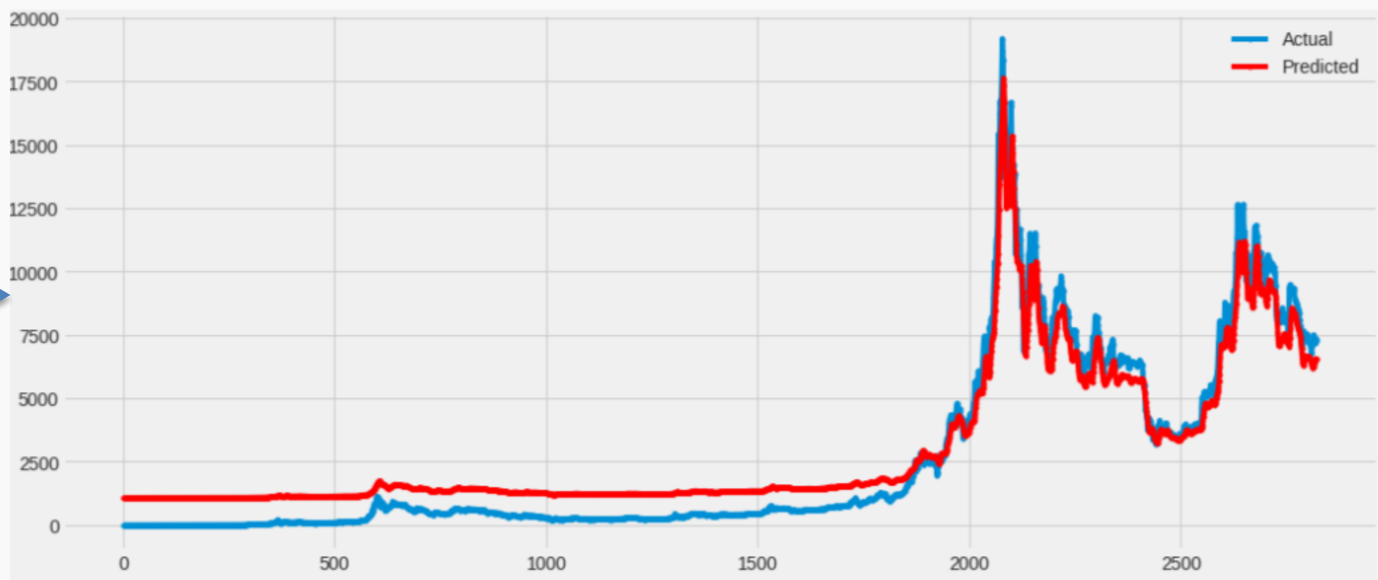
```
<matplotlib.legend.Legend at 0x7f776be63f50>
```



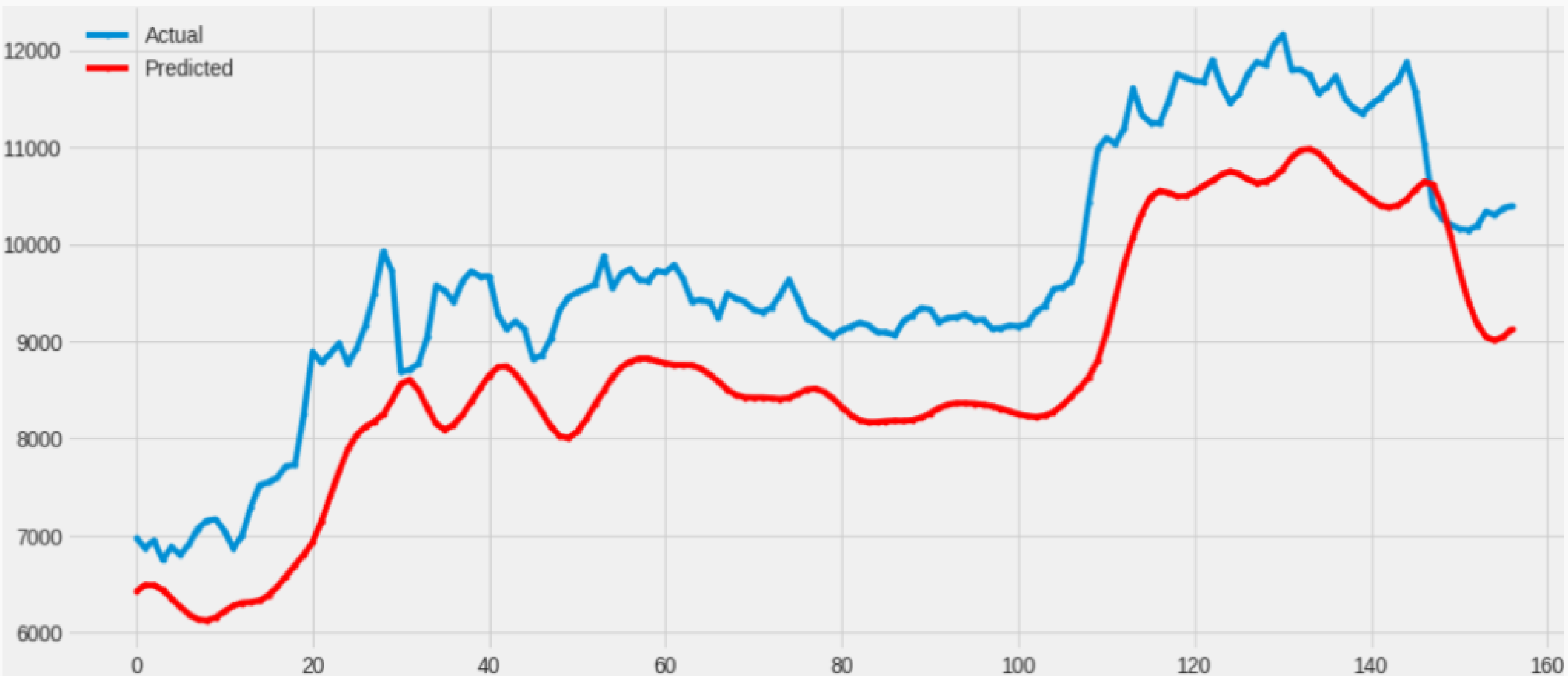
```
#Lets do the prediction and performance checking

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)
```

```
plt.figure(figsize=(16,7))
plt.plot(y_train_inv.flatten(), marker='.', label="Actual")
plt.plot(train_predict_inv.flatten(), 'r', marker='.', label="Predicted")
plt.legend()
```



```
plt.figure(figsize=(16,7))
plt.plot(y_test_inv.flatten(), marker='.', label="Actual")
plt.plot(test_predict_inv.flatten(), 'r', marker='.', label="Predicted")
plt.legend()
```



```python
from sklearn.metrics import mean_absolute_error, mean_squared_error

train_RMSE = np.sqrt(mean_squared_error(y_train, train_predict))
test_RMSE = np.sqrt(mean_squared_error(y_test, test_predict))
train_MAE = np.sqrt(mean_absolute_error(y_train, train_predict))
test_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: {test_RMSE}")
print(f"Test MAE: {test_MAE}")
```

```
Train RMSE: 0.047928890214183744
Train MAE: 0.20979423661732005
Test RMSE: 0.0544482676350978
Test MAE: 0.22614883920646772
```

Conclusion

- Our Proposed model has been succeeded to provide the result prediction bitcoin from Historical bitcoin dataset. Our model with time series techniques can build produce the results with split the data to train and test that we mention above.
- Afterward, as we mentioned before in the article, the Crypto Currency market is influenced by many uncertainty factors. The Cryptocurrencies are influenced by many uncertainties factor such as political issue, the economic issue at impacted to local or global levels. So prediction price bitcoin using LSTM isn’t good enough to make the decision to invest in bitcoin, it is another side for taking the decisions.

Future Scope

- we will be making a website, which will be using API's of with tech stack of frameworks as,

  Frontend:

  Reactjs, context API/redux, html5, css3,bootstrap5, materialcss.

  Backend:

  django/flask

  Database:

  Mongodb

  Tools:

  GIT, visualstudio code, etc

- Our website will have functionalities like crypto Currency price prediction for specific period of time & visualizing/analysing pattern of bitcoin data using different plotting techniques

References

[1] https://www.researchgate.net/publication/340567768_Bitcoin_Price_Prediction_using_ARIMA_Model

[2] https://www.researchgate.net/publication/307598782_Resampling_Strategies_for_Imbalanced_Time_Series

[3] https://en.wikipedia.org/wiki/Long_short-term_memory

[4]https://www.researchgate.net/publication/271978892_Comparison_of_Linear_Interpolation_Method_and_Mean_Method_to_Replace_the_Missing_Values_in_Environmental_Data_Set

[5]https://www.researchgate.net/publication/313867740_A_review_of_missing_values_handling_methods_on_time-series_data

[6]https://www.researchgate.net/publication/336061476_A_Comparative_Study_of_Bitcoin_Price_Prediction_Using_Deep_Learning

[7] http://colah.github.io/posts/2015-08-Understanding-LSTMs/