A Project report on

BITCOIN PRICE PREDICTION USING MACHINE LEARNING ALGORITHMS

A Dissertation submitted to JNTU Hyderabad in partial fulfillment of the academic requirements for the award of the degree.

Bachelor of Technology

in

Computer Science and Engineering

Submitted by

G. Keerthi
(19H51A0572)

K. V. Sri Divya
(19H51A05D6)

L. Bhargavi
(19H51A05D8)

Under the esteemed guidance of

Mr. B. Sivaiah
(Asst Professor, Dept of CSE)

Department of Computer Science and Engineering

CMR COLLEGE OF ENGINEERING AND TECHNOLOGY
(UGC Autonomous)
*Approved by AICTE *Affiliated to JNTUH *NAAC Accredited with A$^+$ Grade

KANDLAKOYA, MEDCHAL ROAD, HYDERABAD - 501401.

2019- 2023

CMR COLLEGE OF ENGINEERING & TECHNOLOGY

KANDLAKOYA, MEDCHAL ROAD, HYDERABAD – 501401

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the Major Project report entitled **" Bitcoin Price Prediction Using Machine Learning Algorithms "** being submitted by G. Keerthi (19H51A0572), K.V.Sri Divya (19H51A05D6), L. Bhargavi (19H51A05D8) in partial fulfillment for the award of **Bachelor of Technology in Computer Science and Engineering** is a record of bonafide work carried out his/her under my guidance and supervision.

The results embodies in this project report have not been submitted to any other University or Institute for the award of any Degree.

Mr. B. Sivaiah
Associate Professor
Dept. of CSE

Dr. Siva Skandha Sanagala
Associate Professor and HOD
Dept. of CSE

Acknowledgement

With great pleasure we want to take this opportunity to express my heartfelt gratitude to all the people who helped in making this project work a grand success.

We are grateful to **Mr. B. Sivaiah,** Associate Professor, Department of Computer Science and Engineering for his valuable technical suggestions and guidance during the execution of this project work.

We would like to thank **Dr. S.Siva Skandha,** Head of the Department of Computer Science and Engineering, CMR College of Engineering and Technology, who is the major driving forces to complete my project work successfully.

We are very grateful to **Dr. Vijaya Kumar Koppula**, Dean-Academic, CMR College of Engineering and Technology, for his constant support and motivation in carrying out the project work successfully.

We are highly indebted to **Dr. V A Narayana,** Principal, CMR College of Engineering and Technology, for giving permission to carry out this project in a successful and fruitful way.

We would like to thank the Teaching & Non- teaching staff of Department of Computer Science and Engineering for their co-operation

We express our sincere thanks to **Mr. Ch. Gopal Reddy**, Secretary, CMR Group of Institutions, for his continuous care.

Finally, We extend thanks to our parents who stood behind us at different stages of this Project. We sincerely acknowledge and thank all those who gave support directly and indirectly in completion of this project work.

G. Keerthi
19H51A0572

K.V. Sri Divya
19H51A05D6

L. Bhargavi
19H51A05D8

TABLE OF CONTENTS

List of Figures

List of Tables

ABSTRACT

Cryptocurrency has been really important in reshaping the financial system due to its increasing popular appeal and worldwide acceptance. Bitcoins are put away in an advanced wallet which is essentially similar to a virtual financial balance. The Bitcoin's worth fluctuates simply like a stock though in an unexpected way. There are various calculations utilized on financial exchange information for value forecast. The goal for this project is to show how a trained machine model can predict the price of a cryptocurrency if we give the right amount of data and computational power, displaying a graph with the predicted values and to make sure with what accuracy the direction of Bitcoin price in USD can be predicted.To decrease the risks, this project has been carried out to predict the price of Bitcoin starting with Linear Regression models, and train on several important features, then we proceed with the implementation of Recurrent Neural Networks (RNN) with Long Short Term Memory (LSTM) cells. We show that the price of Bitcoin can be predicted with Machine Learning with high degree of accuracy.

.

CHAPTER 1

INTRODUCTION

CHAPTER - 1

INTRODUCTION

1.1 BITCOIN

Bitcoin is the world's most valuable cryptocurrency introduced following the release of a whitepaper published in 2008 under the pseudo name Satoshi Nakamoto. The currency is built on a decentralized, peer-to-peer network with the creation of money and transaction management carried out by the members of the network. The result of this is no central authority controls Bitcoin. All Bitcoin transactions are posted in blocks to an open ledger known as the Blockchain to be verified by miners using cryptographic proof. This verification takes place in a trustless system with no intermediary required to pass the funds from sender to receiver. Bitcoin offers a novel opportunity for prediction due its relatively young age and resulting volatility. In addition, it is unique in relation to traditional fiat currencies in terms of its open nature. In comparison, no complete data exists regarding cash transactions or money in circulation of fiat currencies. The well-known efficient market hypothesis suggests the price of assets such as currencies reflect all available information, and as a result trade at their fair value. Although there is an abundance of data available relating to Bitcoin and its network, the author argues that not all market participants will utilise all this information effectively and as a result it may not be reflected in the price. This paper aims to take advantage of this assumption through various machine learning methods. Bitcoin is traded on over 40 exchanges worldwide accepting over 30 different currencies and has a current market capitalization of 9 billion dollars 4 . Interest in Bitcoin has grown significantly with over 250,000 transactions now taking place per day. In addition to the regular use of Bitcoin by private individuals, its lack of correlation with other assets have made it an attractive hedging option to investors. Some research has found that the price volatility of Bitcoin is far greater than that of fiat currencies. This offers significant potential in comparison to mature financial markets.

1.2 MOTIVATION

Cryptocurrency is a decentralized advanced or virtual currency. Utilization of cryptography for security makes it hard to fake. The essential digital money, the Bitcoin was propelled in the year 2009 by Satoshi Nakamot. Bitcoin, Ripple, Bitcoin cash, Bit connect, Dash, Ethereum Classic, Ethereum, Iota, Litecoin, Monero, Nem, Neo, Numeraire, Stratis, Waves and so on are a portion of the well known Cryptocurrencies.

Digital forms of money began to pick up consideration in 2013 and from that point forward seen a noteworthy number of exchanges and consequently cost changes. The digital currency advertise is only like a stock showcase. It has increased open consideration thus accurate prediction of value development of cryptocurrency will help the public to contribute productively in the system and will also help several businesses.

1.3 PROBLEM DEFINITION

Bitcoin is a cryptographic money which is utilized worldwide for advanced installment or basically for speculation purposes. Bitcoin is decentralized for example it isn't possessed by anybody. Exchanges made by Bitcoins are simple as they are not attached to any nation. Speculation should be possible through different commercial centers known as "bitcoin trades". These enable individuals to sell/purchase Bitcoins utilizing various monetary forms. The biggest Bitcoin trade is Mt Gox. Bitcoins are put away in an advanced wallet which is essentially similar to a virtual financial balance. The record of the considerable number of exchanges, the timestamp information is put away in a spot called Block chain. Each record in a block chain is known as a square. Each square contains a pointer to a past square of information. The information on block chain is scrambled. During exchanges the client's name isn't uncovered, however just their wallet ID is made open.

The Bitcoin's worth fluctuates simply like a stock though in an unexpected way. There are various calculations utilized on financial exchange information for value forecast. Notwithstanding, the parameters influencing Bitcoin are extraordinary. In this manner it is important to anticipate the estimation of Bitcoin so right venture choices can be made.

The cost of Bitcoin doesn't rely upon the business occasions or mediating government not at all like securities exchange. Hence, to anticipate the worth we feel it is important to use AI innovation to foresee the cost of Bitcoin.

1.4 RESEARCH OBJECTIVE

The objective of this project is to use the LSTM version of Recurrent Neural Networks, pricing for Bitcoin. To develop a better understanding of its price influence and to compare the prediction outcomes of various machine learning model.

1.5 LIMITATION OF PROJECT

This project is more about finding the best working model for bitcoin prediction which means it won’t be real time system.

CHAPTER 2

BACKGROUND WORK

CHAPTER - 2

BACKGROUND WORK

2.1 An Optimized SVM based on PSO for Cryptocurrency Forecasting

2.1.1. Introduction

Particle Swarm Optimization (PSO) is known as a better algorithm for a static and simple optimization problem. An optimized Support Vector Machine (SVM) based on Particle Swarm Optimization (PSO) is introduced in forecasting the cryptocurrency future price. It is part of Artificial Intelligence (AI) that uses previous experience to forecast future price.

2.1.2. Merits, Demerits and Challenges Merits

- **Merits**

  Can effectively forecast the future price

- **Demerits**
  - Only consider the previous data
  - Cannot predict accurately

- **Challenges**

  To enhanced the accuracy rate of the forecasted price

2.1.3. Implementation

This project uses 5 years daily prices from 2013 through 2018 for all data models and is prepared from price of a daily trading for all six types of cryptocurrencies. This covers the open/close as well as highest/lowest price of the day. This models goes through 4 stages.

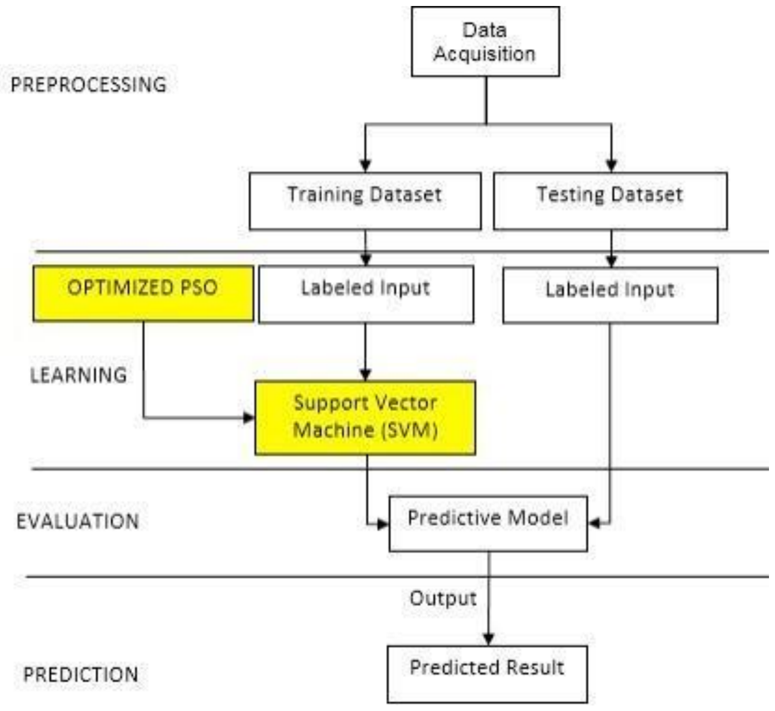


Figure 2.1.3.1: Flow chart of SVM-PSO

2.2 Time-Series Prediction of Cryptocurrency Market using Machine Learning Techniques

2.2.1. Introduction

To predict the market price and stability of Bitcoin in Crypto-market, a machine learning based time series analysis has been applied. Time-series analysis can predict the future ups and downs in the price of Bitcoin by using three algorithms ARIMA, XGBoost, FBProphet.

2.2.2. Merits, Demerits and Challenges Merits

- **Merits**

  Finds the best among 3 used algorithms.

- **Demerits**

  - Time taking Process.

- These system less inaccurate and also less reliable as the Cryptocurrency market is very complex.

- **Challenges**

  To enhance the accuracy rate of the forecasted price.

2.2.3. Implementation

This project use ARIMA, FBProphet, XG Boosting for time series analysis as a machine learning techniques. The parameters on the basis of which we have evaluated these models are Root Mean Square Error (RMSE), Mean Absolute Error (MAE) and R2 conducted experiments on these three techniques, after conducting time series analysis, ARIMA considered as the best model.

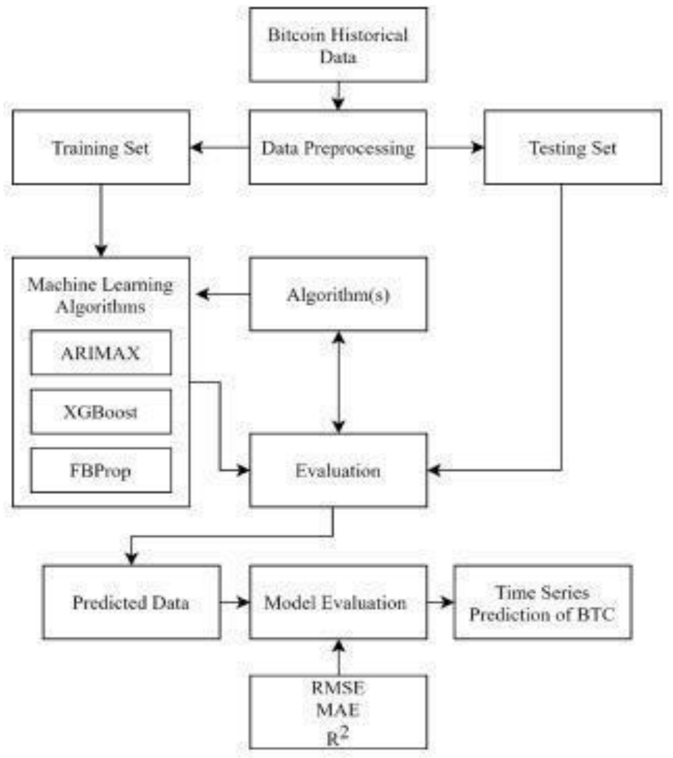


Figure 2.2.3.1: Basic methodology flowchart

2.3 An Accurate Bitcoin Price Prediction using logistic regression Machine Learning model

2.3.1. Introduction

The proposed model is more focused on leveraging the accurate forecast of bitcoin prices via the normalization of a particular dataset. It is a classification problem which is used to predict a binary outcome given a set of independent variables. It is a regression model to predict the probability that a given data entry belongs to the category numbered as “1”.

2.3.2. Merits, Demerits and Challenges Merits

- **Merits**

  Don’t need to pick learning rate

- **Demerits**
  - More complex
  - More of a black box unless you learn the specifics

- **Challenges**

  Can numerically approximate gradient for you (doesn’t always work out well).

2.3.3. Implementation

The logistic regression model computes a weighted sum of the input variables similar to the linear regression, but it runs the result through a special non-linear function, the logistic function or sigmoid function to produce the output y this dataset has been trained to deploy a more accurate forecast of the bitcoin price.

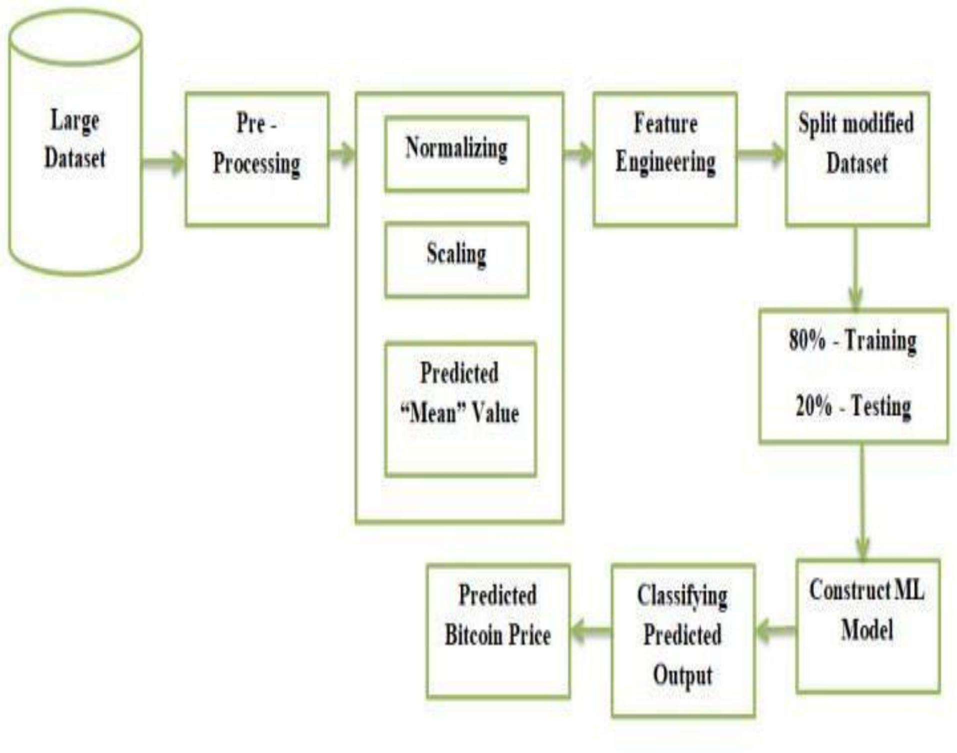


Figure 2.3.3.1: Overall proposed Framework

2.4 Disadvantages of Existing Systems

Existing System uses Linear Regression or several other algorithms which are less accurate in forecasting the accurate values of cryptocurrencies.

These systems tries to linearize the whole data which is simply not an efficient methodto forecast a complex system like cryptocurrency.

As the number of bitcoins in circulation approaches closer to the limit, bitcoins becomeincreasingly harder to mine.

One of the problem that analysts and researchers faced was to implement a systemcapable of accurately predicting the prices.

2.5 Comparison of Existing Solutions

Table 2.5.1 : Comparison of Existing solutions

	An Optimized SVM based on PSO for Cryptocurrency Forecasting	**Time-Series Prediction of Cryptocurrency Market using MachineLearning Techniques**	**An Accurate Bitcoin Price Prediction using logistic regression Machine Learning model**
Algorithms used	● SVM ● Optimized SVM-PSO	● ARIMAX ● XGBOOST ● FBProp	● Lasso ● Logistic regression
Problems	● Only consider the previous data ● Cannot predict accurately	● less inaccurate and also less reliable as the Cryptocurrency market is very complex. ● Time taking Process	● Can numerically approximate gradients for you (doesn't always work out well).
Parameters for performance evaluation	MAPE, MSE	RMSE, MAE, ^{2}	ROC-AUC curve

CHAPTER 3
PROPOSED SYSTEM

CHAPTER 3

PROPOSED SYSTEM

3.1 Objective of Proposed Model

The objective of this project is to implement multiple machine learning algorithms and evaluate their performance using evaluation metrics like mean squared error, mean absolute error and to ascertain with what accuracy can the price of Bitcoin be predicted using different machine learning algorithm and compare their accuracy.

3.2 Algorithms Used for Proposed Model

3.2.1 Linear Regression

Linear Regression is a machine learning algorithm based on supervised learning. It performs a regression task. Regression models a target prediction value based on independent variables. It is mostly used for finding out the relationship between variables and forecasting. Different regression models differ based on – the kind of relationship between dependent and independent variables, they are considering and the number of independent variables being used.

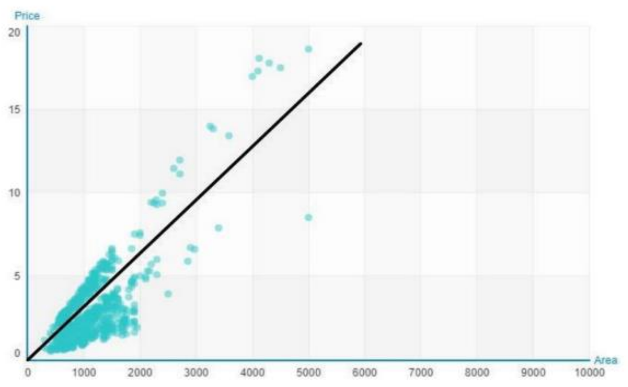


Figure 3.2.1.1: Linear regression scatter plot

Linear regression is the simplest method for prediction. It uses two things as variables which are the predictor variable and the variable which is the most crucial one first whether the predictor variable and su. These regression estimates are used to explain the relationship between one dependent variable and one or more independent variables. The equation of the regression equation with one dependent and one independent variable is defined by the formula.

b = y + x*a

where, b = estimated dependent variable score, y = constant, x = regression coefficient, and a = score on the independent variable.

3.2.2 Forest Regression

Forest regression uses the technique called as Bagging of trees. The main idea here is to decorrelate the several trees. We then reduce the Variance in the Trees by averaging them. Using this approach, a large number of decision trees are created. Random forest training algorithm applies the technique of bootstrap aggregating, or bagging, to tree learners [7].
Given a training set X = x1, ..., xn with responses Y = y1, ..., yn, bagging repeatedly (B times) selects a random sample with replacement of the training set and fits trees to these samples:

For b = 1, ..., B:

1. Sample, with replacement, n training examples from X, Y; call these Xb, Yb.
2. Train a classification or regression tree fb on Xb, Yb.

After training, predictions for unseen samples a' can be made by averaging the predictions from all the

$$\hat{f} = \frac{1}{B}\sum_{b=1}^{B} f_b(x')$$

individual regression trees on a': Additionally, an estimate of the uncertainty of the prediction can be made as the standard deviation of the predictions from all the individual regression trees on a':

$$\sigma = \sqrt{\frac{\sum_{b=1}^{B}(f_b(x') - \hat{f})^2}{B-1}}.$$

Representation of it is as follows-

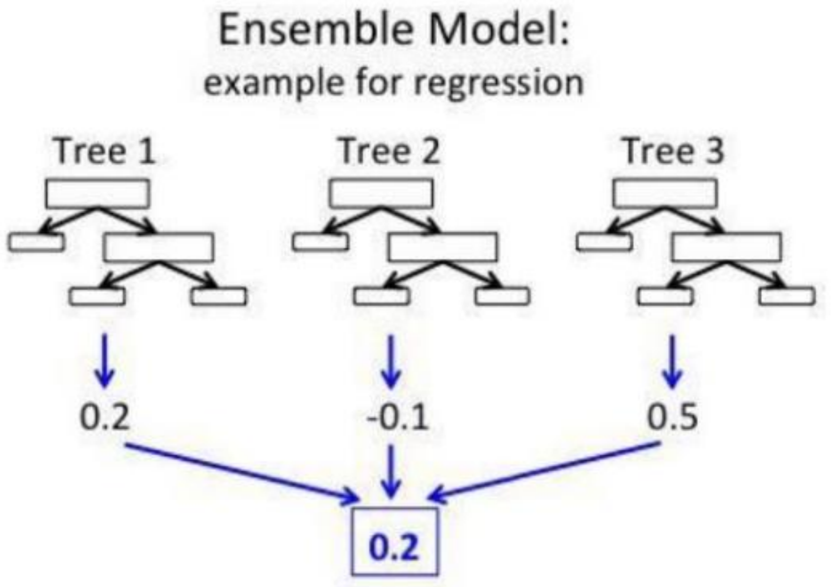


Figure 3.2.2.1: Forrest regression model

3.2.3 Neural Networks

A neural network is a series of algorithms that endeavors to recognize underlying relationships in a set of data through a process that mimics the way the human brain operates. In this sense, neural networks refer to systems of neurons, either organic or artificial in nature. Neural networks can adapt to changing input; so the network generates the best possible result without needing to redesign the output criteria. The concept of neural networks, which has its roots in artificial intelligence, is swiftly gaining popularity in the development of trading systems.

Furthermore, the results of all the above algorithms are fed as input to the neural network. We use neural network applied with boosted regression to increase the accuracy of the result. Neural network does the job pretty well by comparing all the predictions and computing them to display the most accurate result.

A neural network works similarly to the human brain's neural network. A "neuron" in a neural network is a mathematical function that collects and classifies information according to a specific architecture. The network bears a strong resemblance to statistical methods such as curve fitting and regression analysis.

A neural network contains layers of interconnected nodes. Each node is a perceptron and is similar to a multiple linear regression. The perceptron feeds the signal produced by a multiple linear regression into an activation function that may be nonlinear.

In a multi-layered perceptron (MLP), perceptrons are arranged in interconnected layers. The input layer collects input patterns. The output layer has classifications or output signals to which input patterns may map. For instance, the patterns may comprise a list of quantities for technical indicators about a security; potential outputs could be "buy," "hold" or "sell."

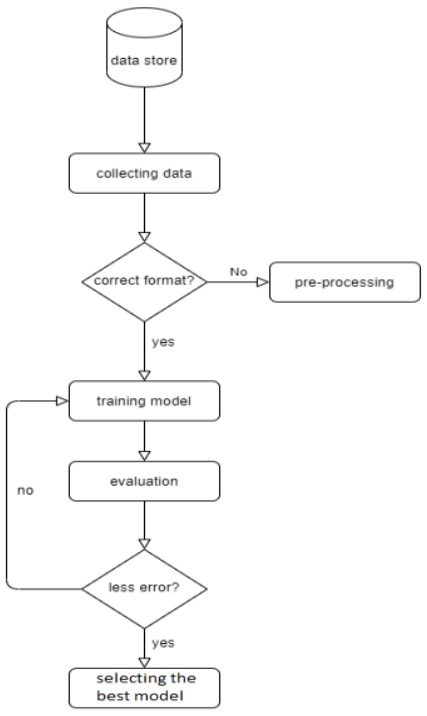


Figure 3.2.3.1: Flow chart

3.2.3.1 LSTM

The help of LSTMs preserves the error which can be reproduced over time and layers. By maintaining a more constant error, recurrent networks can continue to learn over a long period of time (over 200), opening a channel to remotely link causes and effects. S. Saravanakumar, V. Dinesh Kumar [15] LSTMs contain data outside the normal flow of the gated cell recurrent network. Information can be stored, written to or read from a cell in a computer memory similar to data. The cell decides what to store and when to read, write and erase through gates that open and close. The function below converts the series to supervised data. Two models are available in Keras. One is a sequential model that is suitable for predicting time series and the other is used with a functional API. The dense layer is used with input
shapes as the output layer. The optimizer function used is 'adam' which has a learning rate of 0.01 with the mean absolute error as the loss function. The loss method used is mean absolute error.

The long short-term memory network or LSTM addresses the common problem of disappearing gradients in the recurrent neural network. This is a type of recurrent neural network that is used in profound learning, as very large architectures can be trained. LSTM enables the network to learn more about many time steps by maintaining a moresteady error. This enables the network to learn long-term trust. LSTM cell contains forget and remember gates that allow the cell to decide which information to block or transmit based on its strength and importance [9]. As a result, weak signals that prevent the gradient from disappearing can be blocked. The performance of the RNN and LSTM network is assessed to determine the model's efficiency.

Elman's recent development of recurrent neural networks has gained popularity in network designs and increased computational power from graphical processing units. They are primarily useful with sequential data (in our case Bitcoins time series data) because each neuron or unit can access its internal memory to keep information about the previous input. Fig 2 shows simple RNN Structure [14]. One limitation of RNN is that it is influenced by the disappearing problem of the gradient. This problem is that since the layers and time steps of the network are interrelated, they are susceptible to exploding or disappearing gradients.

Recurrent Neural Networks suffer from short-term memory. If a sequence is long enough, they'll have a hard time carrying information from earlier time steps to later ones. So if you are trying to process a paragraph of text to do predictions, RNN's may leave out important information from the beginning.

During back propagation, recurrent neural networks suffer from the vanishing gradient problem. Gradients are values used to update a neural networks weights.

The vanishing gradient problem is when the gradient shrinks as it back propagates through time. If a gradient value becomes extremely small, it doesn't contribute too much learning.

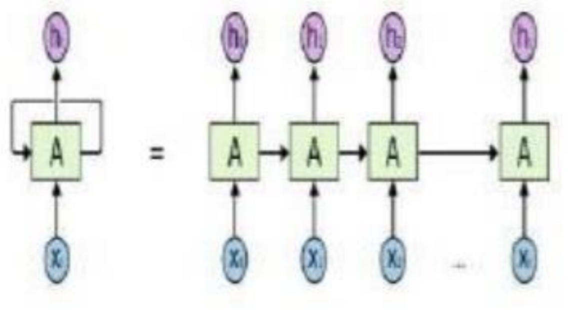


Figure. 3.2.3.1.1: RNN Structure

Vanishing gradients are a problem because they can prove to be too little to learn for the system, while inclinations can be restricted by regularization. In addition, some examinations have found that while RNN is able to deal with long-term dependencies, they often fail to learn in practice due to difficulties between gradient succession and long-term dependency.

LSTM avoids the vanishing gradient problem mainly by using several gate structures, namely input, output, and forget gates, together with a cell unit (the memory part of an LSTM unit), Mathematics 2019, 7, 898 9 of 20 and additive connections between cell states. For an LSTM unit at time t, its hidden state ht , i.e., the output vector of the LSTM unit, is computed as follows,

- $f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f)$
- $i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i)$
- $o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o)$
- $g_t = \tanh(W_g x_t + U_g h_{t-1} + b_g)$
- $c_t = f_t c_{t-1} + i_t g_t$
- $h_t = o_t \tanh(c_t)$

where W, U, and b are parameters to be learned and σ is a sigmoid function. ft , it , and ot , respectively, correspond to the forget, input, and output gates, and ct is the cell state vector. Thanks to the cell state, the long-term dependencies between the data points in the input sequence is maintained well and LSTMs can be applied to long sequence data. In the experiment, only LSTM-based prediction models were considered.

We have analysed a system to provide an accurate prediction of bitcoin prices. The system makes optimal use of Linear Regression, Forest regression. The efficiency of the algorithm has been further increased with use of Neural networks. The system will satisfy customers by providing accurate output and preventing the risk of investing in the wrong house. Additional features for the customer’s benefit can also be added to the system without disturbing its core functionality.

3.3 Designing

The Design of the system for Bitcoin Price Prediction using Machine Learning Algorithms For Bitcoin Price Prediction will be done based on the analysis done. As per the analysis, the finalized algorithms will be included in the Design which will be further implemented.

A data-flow diagram is a way of representing a flow of data through a process or a system. The DFD also provides information about the outputs and inputs of each entity and the process itself. A data-flow diagram has no control flow, there are no decision rules and no loops. Specific operations based on the data can be represented by a flowchart.

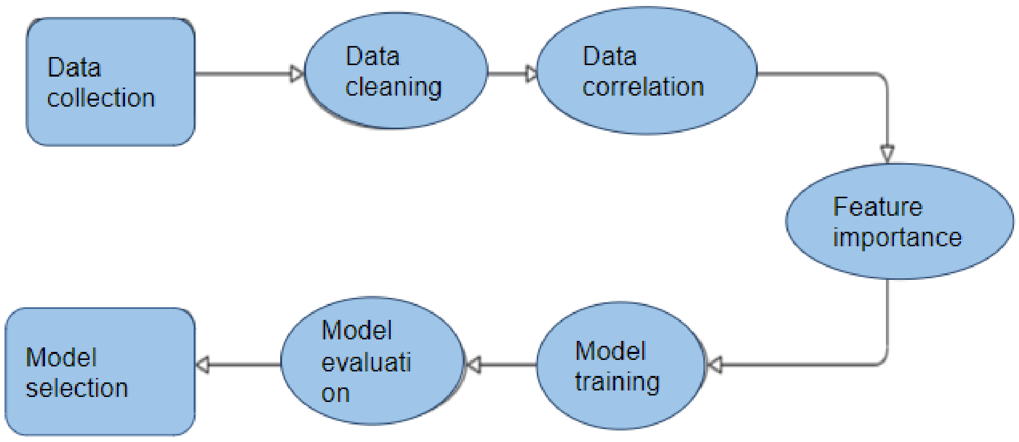


Figure 3.3.1: Data Flow Diagram

3.3.1 UML Diagram

The Unified Modeling Language (UML) is a general-purpose, developmental, modeling language in the field of software engineering that is intended to provide a standard way to visualize the design of a system.

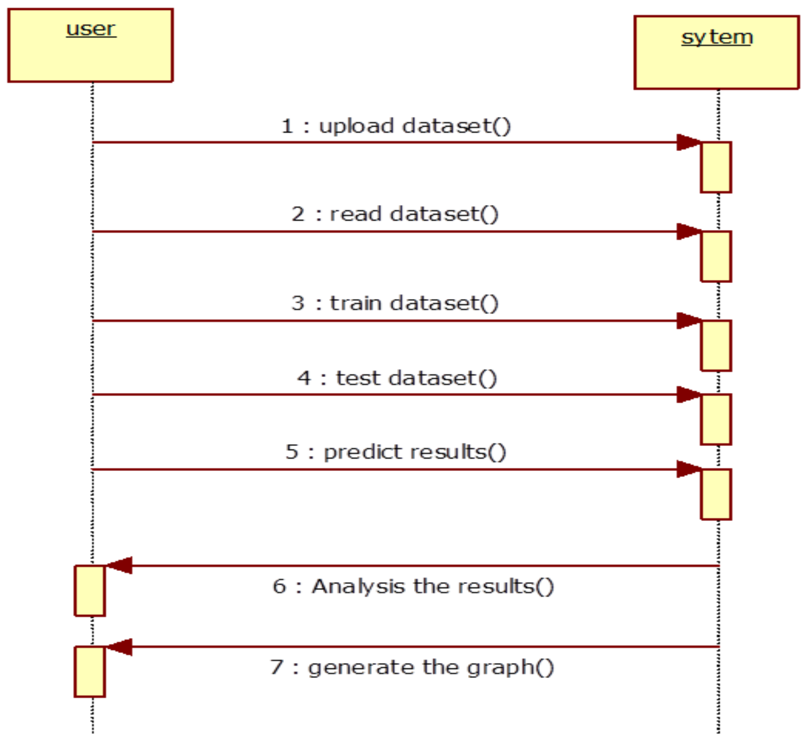


Figure 3.3.1.1: Sequence Diagram

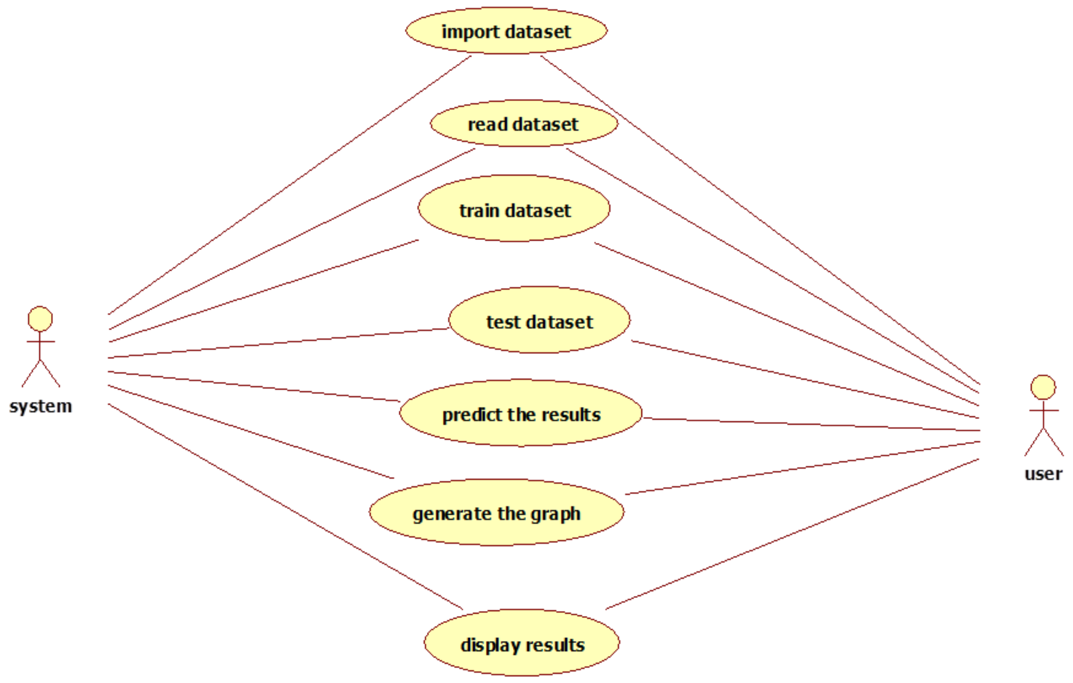


Figure 3.3.1.2: Use Case Diagram

3.3.2 Class diagram

Class diagram is a static diagram. It represents the static view of an application. Class diagram is not only used for visualizing, describing, and documenting different aspects of a system but also for constructing executable code of the software application.

Class diagram describes the attributes and operations of a class and also the constraints imposed on the system. The class diagrams are widely used in the modeling of object oriented systems because they are the only UML diagrams, which can be mapped directly with object-oriented languages.

Class diagram shows a collection of classes, interfaces, associations, collaborations, and constraints. It is also known as a structural diagram.

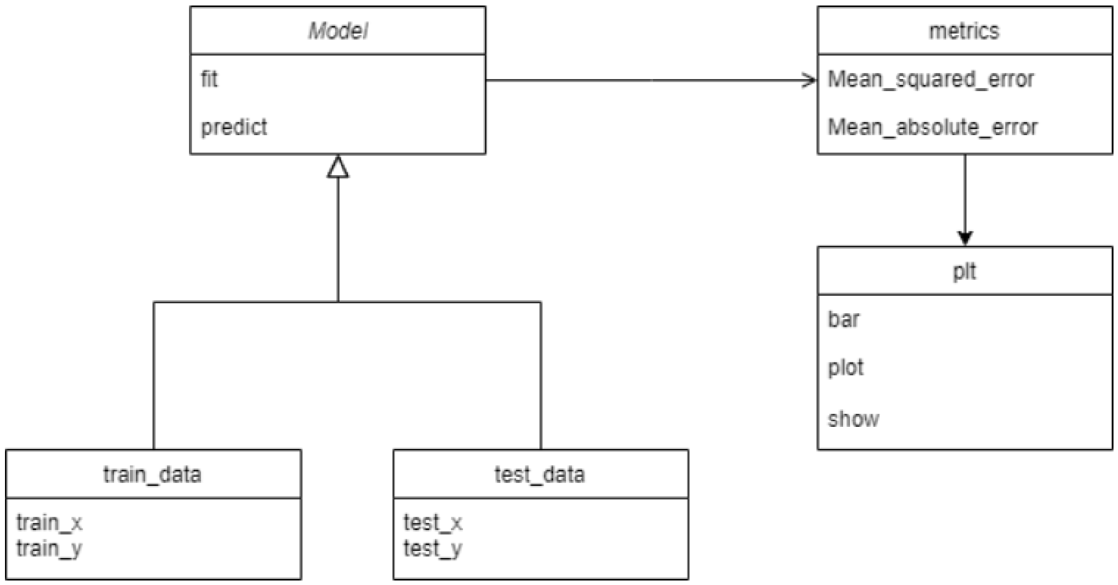


Figure 3.3.2.1: Class Diagram

Modules are files that contain Python definitions and declarations. Modules can define functions, classes, and variables. Modules can also include executable code. Grouping related code in the module can make it easier to understand and use the code. It also makes the code logical. In Python programming, treat modules as the same as code libraries.

We have designed a system to provide an accurate prediction of bitcoin prices. The system makes optimal use of Linear Regression, Forest regression. The efficiency of the algorithm has been further increased with use of Neural networks. The system will satisfy customers by providing accurate output and preventing the risk of investing in the wrong house. Additional features for the customer's benefit can also be added to the system without disturbing its core functionality.

3.4 Softwares Modules / Libraries used

3.4.1 Requirement Specifications

❖ **User Requirement**

The requirement for effective use of predicting future prices is to regularly update the dataset and retrain the model.

❖ **Software Requirement**

1. **OS : Windows XP + / Linux/ MacOS**
2. **Python 3**

Python is an interpreted, high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace. It provides constructs that enable clear programming on both small and large scales. Van Rossum led the language community until stepping down as leader in July 2018.Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object-oriented, imperative, functional and procedural. It also has a comprehensive standard library. The Python interpreter and the extensive standard library are freely available in source or binary form for all major platforms from the Python Web site, https://www.python.org/, and may be freely distributed. The same site also contains distributions of and pointers to many free third party Python modules, programs and tools, and additional documentation.

python
Programming

➢ Argument Passing

When known to the interpreter, the script name and additional arguments thereafter are turned into a list of strings and assigned to the argv variable in the sys module. You can access this list by executing import sys. The length of the list is at least one; when no script and no arguments are given, sys.argv[0] is an empty string. When the script name is given as '-' (meaning standard input), sys.argv[0] is set to '-'. When -c command is used, sys.argv[0] is set to '-c'. When -m module is used, sys.argv[0] is set to the full name of the located module. Options found after -c command or -m module are not consumed by the Python interpreter's option processing but left in sys.argv for the command or module to handle.

➢ Interactive Mode

When commands are read from a tty, the interpreter is said to be in interactive mode. In this mode it prompts for the next command with the primary prompt, usually three greater-than signs (>>>); for continuation lines it prompts with the secondary prompt, by default three dots (...). The interpreter prints a welcome message stating its version number and a copyright notice before printing the first prompt:

```
$ python3.7
Python 3.7 (default, Sep 16 2015, 09:25:04)
[GCC 4.8.2] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Continuation lines are needed when entering a multi-line construct. As an example, take a look at this if statement:

In the following examples, input and output are distinguished by the presence or absence of prompts (>>> and …): to repeat the example, you must type everything after the prompt, when the prompt appears; lines that do not begin with a prompt are output from the interpreter. Note that a secondary prompt on a line by itself in an example means you must type a blank line; this is used to end a multi-line command.

3.Scikit-learn

Scikit-learn is probably the most useful library for machine learning in Python. The sklearnlibrary contains a lot of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction. Please note that sklearn is used to build machine learning models. It should not be used for reading the data, manipulating and summarizing it. There are better libraries for that (e.g. NumPy, Pandas etc.) Scikit-learn comes loaded with a lot of features. • Supervised learning algorithms. • Cross-validation • Unsupervised learning algorithms • Various toy datasets • Feature extractionDefining scikit learn, it is a free software machine
learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, kmeans and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

Scikit-learn was initially developed by David Cournapeau as a Google summer of code project in 2007.Later Matthieu Brucher joined the project and started to use it as a part of his thesis work. In 2010 INRIA got involved and the first public release (v0.1 beta) was published in late January 2010.The project now has more than 30 active contributors and has had paid sponsorship from INRIA, Google, Tinyclues and the Python Software Foundation.

In general, a learning problem considers a set of n samples of data and then tries to predict properties of unknown data. If each sample is more than a single number and, for instance, a multi-dimensional entry (aka multivariate data), it is said to have several attributes or features.

Learning problems fall into a few categories:

• supervised learning, in which the data comes with additional attributes that we want to predict (Click here to go to the scikit-learn supervised learning page).This problem can be either:

• classification: samples belong to two or more classes and we want to learn from already labeled data how to predict the class of unlabeled data. An example of a classification problem would be handwritten digit recognition, in which the aim is to assign each input vector to one of a finite number of discrete categories. Another way to think of classification is as a discrete (as opposed to continuous) form of supervised learning where one has a limited number of categories and for each of the n samples provided, one is to try to label them with the correct category or class.

• regression: if the desired output consists of one or more continuous variables, then the task is called regression. An example of a regression problem would be the prediction of the length of a salmon as a function of its age and weight.

• unsupervised learning, in which the training data consists of a set of input vectors x without any corresponding target values. The goal in such problems may be to discover groups of similar examples within the data, where it is called clustering, or to determine the distribution of data within the input space, known as density estimation, or to project the data from a high-dimensional space down to two or three dimensions for the purpose of visualization (Click here to go to the Scikit-Learn unsupervised learning page).

Training set and testing set

Machine learning is about learning some properties of a data set and then testing those properties against another data set. A common practice in machine learning is to evaluate an algorithm by splitting a data set into two. We call one of those sets the training set, on which we learn some properties; we call the other set the testing set, on which we test the learned properties. Loading an example dataset scikit-learn comes with a few standard datasets, for instance the iris and digits datasets for classification and the boston house prices dataset for regression.

In the following, we start a Python interpreter from our shell and then load the iris and digits datasets. Our notational convention is that $ denotes the shell prompt while >>> denotes the Python interpreter prompt.

```
$ python
>>> from sklearn import datasets
>>> iris = datasets.load_iris()
>>> digits = datasets.load_digits()
```

A dataset is a dictionary-like object that holds all the data and some metadata about the data. This data is stored in the .data member, which is a n_samples, n_features array. In the case of supervised problem, one or more response variables are stored in the .target member. More details on the different datasets can be found in the dedicated section.

For instance, in the case of the digits dataset, digits.data gives access to the features that can be used to classify the digits samples: >>>

```
>>> print(digits.data)
[[ 0.   0.   5. ...   0.   0.   0.]

 [ 0.   0.   0. ...  10.   0.   0.]
 [ 0.   0.   0. ...  16.   9.   0.]
 ...

 [ 0.   0.   1. ...   6.   0.   0.]
 [ 0.   0.   2. ...  12.   0.   0.]
 [ 0.   0.  10. ...  12.   1.   0.]]
```

and digits.target gives the ground truth for the digit dataset, that is the number corresponding to each digit image that we are trying to learn: >>>

```
>>> digits.target array([0,1, 2, ..., 8, 9, 8])

>>> digits.images[0]

array([[  0.,   0.,   5.,  13.,   9.,   1.,   0.,   0.],


        [  0.,   0.,  13.,  15.,  10.,  15.,   5.,   0.],
        [  0.,   3.,  15.,   2.,   0.,  11.,   8.,   0.],
        [  0.,   4.,  12.,   0.,   0.,   8.,   8.,   0.],
        [  0.,   5.,   8.,   0.,   0.,   9.,   8.,   0.],
        [  0.,   4.,  11.,   0.,   1.,  12.,   7.,   0.],
        [  0.,   2.,  14.,   5.,  10.,  12.,   0.,   0.],
        [  0.,   0.,   6.,  13.,  10.,   0.,   0.,   0.]])
```

Learning and predicting

In the case of the digits dataset, the task is to predict, given an image, which digit it represents. We are given samples of each of the 10 possible classes (the digits zero through nine) on which we *fit* an estimator to be able to *predict* the classes to which unseen samples belong.

In scikit-learn, an estimator for classification is a Python object that implements the methods fit(X, y) and predict(T).

An example of an estimator is the class sklearn.svm.SVC, which implements support vector classification. The estimator's constructor takes as arguments the model's parameters.

For now, we will consider the estimator as a black box:

```
>>>

>>> from sklearn import svm
>>> clf = svm.SVC(gamma=0.001, C=100.)
```

The clf (for classifier) estimator instance is first fitted to the model; that is, it must learn from the model. This is done by passing our training set to the fit method. For the training set, we'll use all the images from our dataset, except for the last image, which we'll reserve for our predicting. We select the training set with the [:-1] Python syntax, which produces a new array that contains all but

the last item from digits.data:

```
>>> >>> clf.fit(digits.data[:-1],
digits.target[:-1])

SVC(C=100.0, cache_size=200,
class_weight=None, coef0=0.0,
decision_function_shape='ovr',
degree=3, gamma=0.001, kernel='rbf',
max_iter=-1, probability=False,
random_state=None, shrinking=True,

tol=0.001, verbose=False)
```

Now you can predict new values. In this case, you'll predict using the last image from digits.data. By predicting, you'll determine the image from the training set that best matches the last image.

```
>>>

>>>
clf.predict(digi
ts.data[-1:])

 array([8])
```

4. Pandas

Pandas is a Python package providing fast, flexible, and expressive data structures designed to make working with “relational” or “labeled” data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, real-world data analysis in Python. pandas is well suited for many different kinds of data:

- Tabular data with heterogeneously-typed columns, as in an SQL table or Excelspreadsheet
- Ordered and unordered (not necessarily fixed-frequency) time series data.
- Arbitrary matrix data (homogeneously typed or heterogeneous) with row andcolumn labels
- Any other form of observational / statistical data sets. The data need not be labeledat all to be placed into a pandas data structure

Pandas is fast. Many of the low-level algorithmic bits have been extensively tweaked in Cython code. However, as with anything else generalization usually sacrifices performance. So if you focus on one feature for your application you may be able to create a faster specialized tool. pandas is a dependency of statsmodels, making it an important part of the statistical computing ecosystem in Python. pandas has been used extensively inproduction in financial applications.

The two primary data structures of pandas, Series (1-dimensional) and DataFrame (2dimensional), handle the vast majority of typical use cases in finance, statistics, social science, and many areas of engineering. For R users, DataFrame provides everything that R’s data.frame provides and much more. pandas is built on top of NumPy and is intended to integrate well within a scientific computing environment with many other 3rd party libraries.

Here are just a few of the things that pandas does well:

• Easy handling of missing data (represented as NaN) in floating point as well as non-floating point data

• Size mutability: columns can be inserted and deleted from DataFrame and higher dimensional objects

• Automatic and explicit data alignment: objects can be explicitly aligned to a set of labels, or the user can simply ignore the labels and let Series, DataFrame, etc. automatically align the data for you in computations

• Powerful, flexible group by functionality to perform split-apply-combine operations on data sets, for both aggregating and transforming data

• Intelligent label-based slicing, fancy indexing, and subsetting of large data sets

• Intuitive merging and joining data sets

• Flexible reshaping and pivoting of data sets

• Hierarchical labeling of axes (possible to have multiple labels per tick)

• Robust IO tools for loading data from flat files (CSV and delimited), Excel files, databases, and saving / loading data from the ultrafast HDF5 format

• Time series-specific functionality: date range generation and frequency conversion, moving window statistics, moving window linear regressions, date shifting and lagging, etc.

Many of these principles are here to address the shortcomings frequently experienced using other languages / scientific research environments. For data scientists, working with data is typically divided into multiple stages: munging and cleaning data, analyzing / modeling it, then organizing the results of the analysis into a form suitable for plotting or tabular display. pandas is the ideal tool for all of these tasks.

4. Tensorflow

TensorFlow is a free and open-source software library for dataflow and differentiable programming across a range of tasks. It is a symbolic math library, and is also used for machine learning applications such as neural networks. It is used for both research and production at Google. It is a standard expectation in the industry to have experience in TensorFlow to work in machine learning. TensorFlow was developed by the Google Brain team for internal Google use. It was released under the Apache 2.0 open-source license on November 9, 2015.

Starting in 2011, Google Brain built DistBelief as a proprietary machine learning system based on deep learning neural networks. Its use grew rapidly across diverse Alphabet companies in both research and commercial applications. Google assigned multiple computer scientists, including Jeff Dean, to simplify and refactor the codebase of DistBelief into a faster, more robust application-grade library, which became TensorFlow. In 2009, the team, led by Geoffrey Hinton, had implemented generalized backpropagation and other improvements which allowed generation of neural networks with substantially higher accuracy, for instance a 25% reduction in errors in speech recognition.

TensorFlow is Google Brain's second-generation system. Version 1.0.0 was released on February 11, 2017. While the reference implementation runs on single devices, TensorFlow can run on multiple CPUs and GPUs (with optional CUDA and SYCL extensions for general-purpose computing on graphics processing units). TensorFlow is available on 64-bit Linux, macOS, Windows, and mobile computing platforms including Android and iOS.

Its flexible architecture allows for the easy deployment of computation across a variety of platforms (CPUs, GPUs, TPUs), and from desktops to clusters of servers to mobile and edge devices.

TensorFlow computations are expressed as stateful dataflow graphs. The name TensorFlow derives from the operations that such neural networks perform on multidimensional data arrays, which are referred to as tensors. During the Google I/O Conference in June 2016, Jeff Dean stated that 1,500 repositories on GitHub mentioned TensorFlow, of which only 5 were from Google.

Eager Execution

TensorFlow's eager execution is an imperative programming environment that evaluates operations immediately, without building graphs: operations return concrete values instead of constructing a computational graph to run later. This makes it easy to get started with TensorFlow and debug models, and it reduces boilerplate as well. To follow along with this guide, run the code samples below in an interactive python interpreter.

Eager execution is a flexible machine learning platform for research and experimentation, providing:

- *An intuitive interface*—Structure your code naturally and use Python data structures. Quickly iterate on small models and small data.
- *Easier debugging*—Call ops directly to inspect running models and test changes. Use standard Python debugging tools for immediate error reporting.
- *Natural control flow*—Use Python control flow instead of graph control flow, simplifying the specification of dynamic models.

Eager execution supports most TensorFlow operations and GPU acceleration.

Tensor Values

The central unit of data in TensorFlow is the tensor. A tensor consists of a set of primitive values shaped into an array of any number of dimensions. A tensor's rank is its number of dimensions, while its shape is a tuple of integers specifying the array's length along each dimension. Here are some examples of tensor values:

3. # a rank 0 tensor; a scalar with shape [],

[1., 2., 3.] # a rank 1 tensor; a vector with shape [3]

[[1., 2., 3.], [4., 5., 6.]] # a rank 2 tensor; a matrix with shape [2, 3]

[[[1., 2., 3.]], [[7., 8., 9.]]] # a rank 3 tensor with shape [2, 1, 3]

TensorFlow uses numpy arrays to represent tensor values.

TensorFlow Core Walkthrough

You might think of TensorFlow Core programs as consisting of two discrete sections:

1. Building the computational graph (a tf.Graph).
2. Running the computational graph (using a tf.Session).

Graph

A computational graph is a series of TensorFlow operations arranged into a graph. The graph is composed of two types of objects.

• tf.Operation (or "ops"): The nodes of the graph. Operations describe calculations that consume and produce tensors.

• tf.Tensor: The edges in the graph. These represent the values that will flow through the graph. Most TensorFlow functions return tf.Tensors.

Let's build a simple computational graph. The most basic operation is a constant. The Python function that builds the operation takes a tensor value as input. The resulting operation takes no inputs. When run, it outputs the value that was passed to the constructor. We can create two floating point constants a and b as follows:

```
a = tf.constant(3.0, dtype=tf.float32)

b = tf.constant(4.0) # also tf.float32 implicitly

total = a + b

print(a)

print(b)

print(total)
```

The print statements produce:

```
Tensor("Const:0", shape=(), dtype=float32)

Tensor("Const_1:0", shape=(), dtype=float32)

Tensor("add:0", shape=(), dtype=float32)
```

Notice that printing the tensors does not output the values 3.0, 4.0, and 7.0 as you might expect. The above statements only build the computation graph. These tf.Tensor objects just represent the results of the operations that will be run.

Each operation in a graph is given a unique name. This name is independent of the names the objects are assigned to in Python. Tensors are named after the operation that produces them followed by an output index, as in "add:0" above.

TensorBoard

TensorFlow provides a utility called TensorBoard. One of TensorBoard's many capabilities is visualizing a computation graph. You can easily do this with a few simple commands.

First you save the computation graph to a TensorBoard summary file as follows:

```
Writer          =          tf.summary.FileWriter('.')
writer.add_graph(tf.get_default_graph())
writer.flush()
```

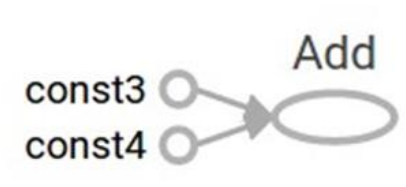
This will produce an event file in the current directory with a name in the following format:

```
events.out.tfevents.{timestamp}.{hostname}
```

Now, in a new terminal, launch TensorBoard with the following shell command:

```
tensorboard –logdir
```

Then open TensorBoard's graphs page in your browser, and you should see a graph similar to the following:



For more about TensorBoard's graph visualization tools see TensorBoard: Graph Visualization.

Session

To evaluate tensors, instantiate a tf.Session object, informally known as a session. A session encapsulates the state of the TensorFlow runtime, and runs TensorFlow operations. If a tf.Graph is like a .py file, a tf.Session is like the python executable. The following code creates a tf.Session object and then invokes its run method to evaluate the total tensor we created above:

```
sess = tf.Session()
print(sess.run(total))
```

When you request the output of a node with Session.run TensorFlow backtracks through the graph and runs all the nodes that provide input to the requested output node. So this prints the expected value of 7.0:

```
7.0
```

You can pass multiple tensors to tf.Session.run. The run method transparently handles any combination of tuples or dictionaries, as in the following example:

```
print(sess.run({'ab':(a, b), 'total':total}))
```

which returns the results in a structure of the same layout:

```
{'total': 7.0, 'ab': (3.0, 4.0)}
```

During a call to tf.Session.run any tf.Tensor only has a single value. For example, the following code calls tf.random_uniform to produce a tf.Tensor that generates a random 3-element vector (with values in [0,1)):

```
vec = tf.random_uniform(shape=(3,))

out1 = vec + 1

out2 = vec + 2

print(sess.run(vec))

print(sess.run(vec))

print(sess.run((out1, out2)))
```

The result shows a different random value on each call to run, but a consistent value during a single run (out1 and out2 receive the same random input):

```
[ 0.52917576  0.64076328  0.68353939]
[ 0.66192627  0.89126778  0.06254101]
(
  array([ 1.88408756,  1.87149239,  1.84057522], dtype=float32),   array([
2.88408756,  2.87149239,  2.84057522], dtype=float32)
)
```

Some TensorFlow functions return tf.Operations instead of tf.Tensors. The result of calling run on an Operation is None. You run an operation to cause a side-effect, not to retrieve a value. Examples of this include the initialization, and training ops demonstrated later.

Feeding

As it stands, this graph is not especially interesting because it always produces a constant result. A graph can be parameterized to accept external inputs, known as placeholders. A placeholder is a promise to provide a value later, like a function argument.

```
x=tf.placeholder(t
f.float32)
y=tf.placeholder(t
f.float32)
 z = x + y
```

The preceding three lines are a bit like a function in which we define two input parameters (x and y) and then an operation on them. We can evaluate this graph with multiple inputs by using the feed_dict argument of the tf.Session.run method to feed concrete values to the placeholders:

```
print(sess.run(z,feed_dict={x: 3, y: 4.5}))
print(sess.run(z,feed_dict={x: [1, 3], y: [2,
4]}))
```

This results in the following output:

```
7.5
[ 3.  7.]
```

Also note that the feed_dict argument can be used to overwrite any tensor in the graph. The only difference between placeholders and other tf.Tensors is that placeholders throw an error if no value is fed to them.

Data Sets

Placeholders work for simple experiments, but tf.data are the preferred method of streaming data into a model. To get a runnable tf.Tensor from a Dataset you must first convert it to a tf.data.Iterator, and then call the Iterator's tf.data.Iterator.get_next method. The simplest way to create an Iterator is with the tf.data.Dataset.make_one_shot_iterator method. For example, in the following code the next_item tensor will return a row from the my_data array on each run call:

```
my_data = [
        [0, 1,],
        [2, 3,],
        [4, 5,],
        [6, 7,],
]
slices   =   tf.data.Dataset.from_tensor_slices(my_data)
next_item = slices.make_one_shot_iterator().get_next()
```

Reaching the end of the data stream causes Dataset to throw an tf.errors.OutOfRangeError. For example, the following code reads the next_item until there is no more data to read:

```
while True:

try:

print(sess.run(next_item))

except tf.errors.OutOfRangeError:

break
```

If the Dataset depends on stateful operations you may need to initialize the iterator before using it, as shown below:

```
r = tf.random_normal([10,3])

dataset=tf.data.Dataset.from_ten

sor_slices(r)

iterator=dataset.make_initializab

le_iterator()       next_row       =

iterator.get_next()

 ess.run(iterator.initializer) while True:

 try:

    print(sess.run(next_

row))
```

```
 except
tf.errors.OutOfRange
Error:
break
```

Layers

A trainable model must modify the values in the graph to get new outputs with the same input. tf.layers are the preferred way to add trainable parameters to a graph.

Layers package together both the variables and the operations that act on them. For example a densely-connected layer performs a weighted sum across all inputs for each output and applies an optional activation function. The connection weights and biases are managed by the layer object.

Creating Layers

The following code creates a tf.layers.Dense layer that takes a batch of input vectors, and produces a single output value for each. To apply a layer to an input, call the layer a

```
 x = tf.placeholder(tf.float32, shape=[None, 3])

linear_model = tf.layers.Dense(units=1)

y = linear_model(x)
```

layer inspects its input to determine sizes for its internal variables. So here w must set the shape of the x placeholder so that the layer can build a weight matrix of the correct size. Now that we have defined the calculation of the output, y, there is one more detail we need to take care of before we run the calculation.

5. Numpy

NumPy gives us the best of both worlds: element-by-element operations are the “default mode” when an ndarray is involved, but the element-by-element operation 8 is speedily executed by pre-compiled C code. NumPy fully supports an objectoriented approach, starting, once again, with ndarray. For example, ndarray is a class, possessing numerous methods and attributes. Many of its methods are mirrored by functions in the outer-most NumPy namespace, allowing the programmer to code in whichever paradigm they prefer.This flexibility has allowed the NumPy array dialect and NumPy ndarray class to becomethe de-facto language of multidimensional data interchange used in Python.

6. Matplotlib

Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt,or GTK+. There is also a procedural "pylab" interface based on a statemachine (like OpenGL), designed to closely resemble that of MATLAB, though its use isdiscouraged. SciPy makes use of Matplotlib. Matplotlib was originally written by John D. Hunter, since then it has an active development community, and is distributed under a BSD-style license. Michael Droettboom was nominated as matplotlib's lead developer shortly before John Hunter's death in August 2012, and further joined by Thomas Caswell.

7. Seaborn

Seaborn is a library for making statistical graphics in Python. It builds on top of matplotlib and integrates closely with pandas data structures. Seaborn helps you explore and understand your data. Its plotting functions operate on dataframes and arrays containing whole datasets and internally perform the necessary semantic mapping and statistical aggregation to produce informative plots. Its dataset- oriented, declarative API lets you focus on what the different elements of your plots mean, rather than on the details of how to draw them.

Seaborn is the only library we need to import for this simple example. By convention, itis imported with the shorthand sns. Behind the scenes, seaborn uses matplotlib to draw its plots. For interactive work, it’s recommended to use a Jupyter/Python interface in matplotlib mode, or else you’ll have to call matplotlib.pyplot.show() when you want to see the plot.

- **Hardware Requirement**

i.	Processor	: I3 or above
ii.	RAM	: 4GB or more
iii.	Storage	: 120GB or more

3.5 Stepwise Implementation and Code

As the Design of the System was finalized, the Implementation of the system is needed to be done. The implementation of the system will be done using Python Programming and other necessary Machine Learning Modules. The System for BITCOIN PRICE PREDICTION USING MACHINE LEARNING ALGORITHMS will be implemented on the finalized algorithms.

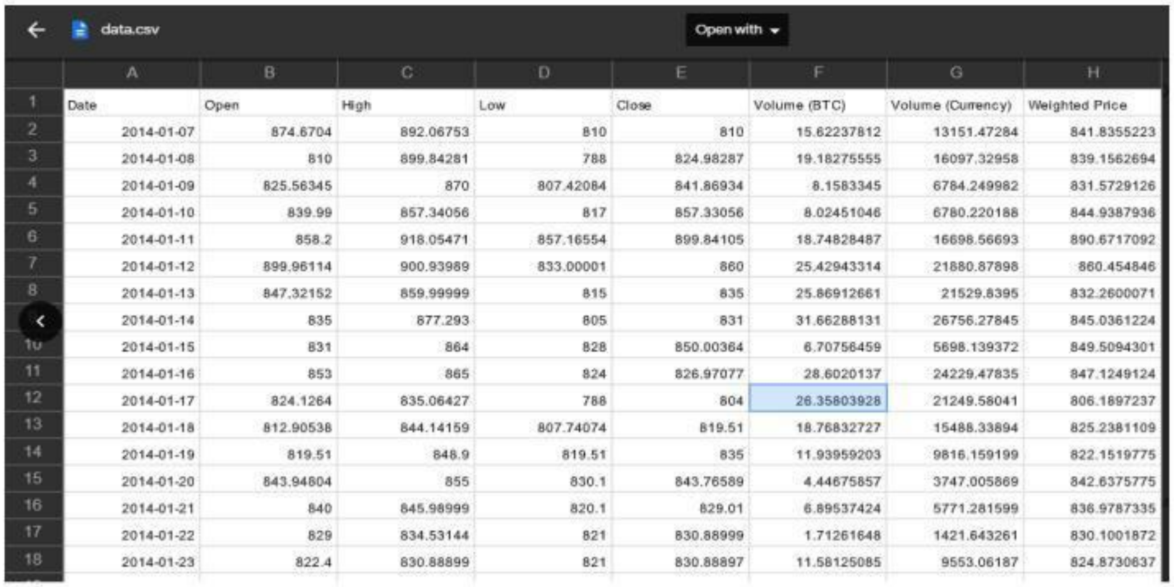
3.5.1 Explanation of Key Functions

- **Load_dataset:**
  The Function is responsible for loading the dataset from the file available.
- **Train_test_split:**
  The Function is responsible for splitting the dataset in two parts out of which one will be used for Training and another will be used for testing.
- **Train:**
  The Function is responsible for taking the training dataset and train the Machine Learning Model.
- **Predict:**
  The Function is responsible for predicting the results using the Trained Machine Learning Model.

3.5.2 Data Collection

We collected data from bitcoin.com and downloaded all of the data they had available, ending up with 37 different Excel files. Tough luck. There was a way to feed each file separately into the neural network, but it was way easier to manually combine the files

into one big file that has 37 columns instead of just one column per file. So I did this and got a huge file that was 37 columns by around 2667 rows (each row is a day, each column is a feature of Bitcoin for that
day). We also had to do some data preprocessing to make sure my data was fed into the neural network in the best way.



	A	B	C	D	E	F	G	H
1	Date	Open	High	Low	Close	Volume (BTC)	Volume (Currency)	Weighted Price
2	2014-01-07	874.6704	892.06753	810	810	15.62237812	13151.47284	841.8355223
3	2014-01-08	810	899.84281	788	824.98287	19.18275555	16097.32958	839.1562694
4	2014-01-09	825.56345	870	807.42084	841.86934	8.1583345	6784.249982	831.5729126
5	2014-01-10	839.99	857.34056	817	857.33056	8.02451046	6780.220188	844.9387936
6	2014-01-11	858.2	918.05471	857.16554	899.84105	18.74828487	16698.56693	890.6717092
7	2014-01-12	899.96114	900.93989	833.00001	860	25.42943314	21880.87898	860.454846
8	2014-01-13	847.32152	859.99999	815	835	25.86912661	21529.8395	832.2600071
	2014-01-14	835	877.293	805	831	31.66288131	26756.27845	845.0361224
10	2014-01-15	831	864	828	850.00364	6.70756459	5698.139372	849.5094301
11	2014-01-16	853	865	824	826.97077	28.6020137	24229.47835	847.1249124
12	2014-01-17	824.1264	835.06427	788	804	26.35803928	21249.58041	806.1897237
13	2014-01-18	812.90538	844.14159	807.74074	819.51	18.76832727	15488.33894	825.2381109
14	2014-01-19	819.51	848.9	819.51	835	11.93959203	9816.159199	822.1519775
15	2014-01-20	843.94804	855	830.1	843.76589	4.44675857	3747.005869	842.6375775
16	2014-01-21	840	845.98999	820.1	829.01	6.89537424	5771.281599	836.9787335
17	2014-01-22	829	834.53144	821	830.88999	1.71261648	1421.643261	830.1001872
18	2014-01-23	822.4	830.88899	821	830.88897	11.58125085	9553.06187	824.8730637

Figure 3.5.2.1 : Data Collection

3.5.3 Data Preprocessing

Okay buckle up because data preprocessing has some pretty technical steps to it. The first thing I did was to apply a sliding window transformation to the data? What's that to? Basically, I slid an imaginary window over the big Excel file to make it into arrays of 50 **days by 37 features. So imagine** changing a 2D rectangle into a 3D rectangular prism. It's kind of like that. The next thing I did was some normalization on the data. Since the range of values for each feature varied so much, it was in my best interest to normalize the numbers for each feature so that each separate data point would contribute about the same to the overall training of the neural network. Sounds like a lot of work! Hold your horses — I still had to split the data into training, validation, and testing sets. This step is pretty easy though; I basically took the most recent 10 percent of the data as a test set and took the other 90 percent as training data (5 percent of that 90 percent was split off into a validation set). With the data preprocessing done I could finally get started making a cool neural network!

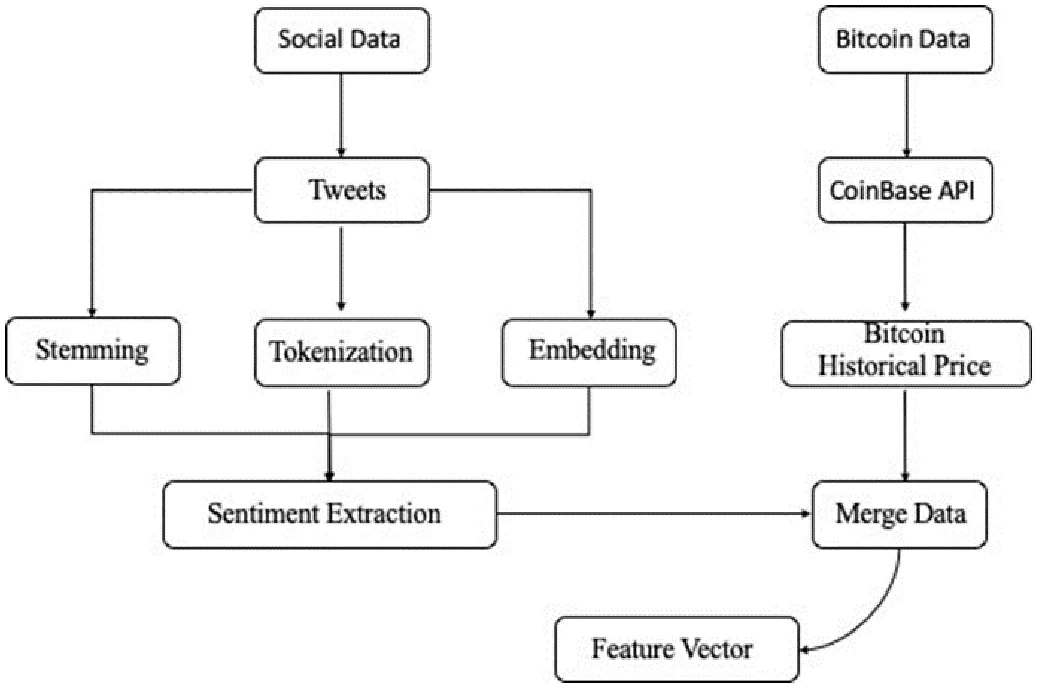


Figure 3.5.3.1: Data Preprocessing

As shown in Figure 3.5.2.1, the dataset is by minute, and contains around 3,409,920 points. Since we predicted the price by hours, we have had 1,409,920/60 which is 56,832 datapoints. The dataset is further split into training, validating and testing sets. As shown in Figure 3.5.2.1, training data takes up to 80% of the entire dataset, and validating and testing 10% respectively. As the time series data, samples are not randomized. We used the first 24 hours' Bitcoin price as input to predict the next hours' Bitcoin price. Several other pre-processing methods are implemented to improve data processing and model convergency efficiency. Minibatch is used to split large data into small batches, which improves memory efficiency. Minimum-Maximum normalization and window-based normalization is used to set the whole training dataset to (−1, 1) scale. Window normalization is based on the reference of stock market. The normalization methods will take each sized window and normalize each one to reflect percentage changes from the start hour of the window .

3.5.4 Data Preparation

Data Preparation In our study, the raw dataset S consisted of 2590 days Bitcoin data (from 29 November 2011 to 31 December 2018). Each data point, p, is an 18dimensional real-valued vector where each dimension stores a daily value of one of the Bitcoin blockchain features discussed in Section 2. To predict Bitcoin prices, sequences of previous data were exploited. When the size of each sequence was m, a total of 2590 − m + 1 sequence data were used to train and test various deep learning-based prediction models, that is, from S[1 : m] to S[2590 − m + 1 : 2590], . The sequence size was experimentally determined as explained in Section 4, and the first 80% of the sequence data were used as the training data and the rest as the test data. Given a sequence S[i : i + m − 1], for a regression problem, we predicted the Bitcoin price of the (i + m)-th day, and for a classification problem, we predicted whether the price would go up or down in the (i + m)-th day with respect to the (i + m − 1)-th day. In other words, by analyzing previous Bitcoin prices including today's price, we predict tomorrow's price for the regression case and predict if tomorrow's price will go up or down with respect to today's price for the classification case, as it is what most investors or traders would be interested in. Figure 4. Data preprocessing. A total of 2590 − m + 1 sequence data are generated from the whole dataset of 2590 days if m consecutive days are analyzed to predict the next Bitcoin price. Each sequence X was normalized in the following way. For all i and j, Y[i][j] = X[i][j] X[0][j] − 1 Then, the normalized sequences Ys were used instead of the original sequences Xs to build and test the prediction models. This normalization method is more effective in capturing local trends than the usual minmax normalization method. Assuming that maxj and minj are, respectively, the maximum Mathematics 2019, 7, 898 7 of 20 and the minimum value of the j-th dimension of the raw data points, the "minmax normalization" method translates each data point as follows, S 0 [i][j] = S[i][j]−minj maxj−minj That is, every value is normalized into a value between 0 and 1. As the highest price is 7054.33 times higher than the lowest price in our dataset, if the usual minmax normalization method is used, the Bitcoin prices hardly change in most sequence data, and thus the prediction accuracy is degraded. Experimental results also confirmed that the first value-based normalization was more effective than the minmax normalization.

3.5.5 Implementation

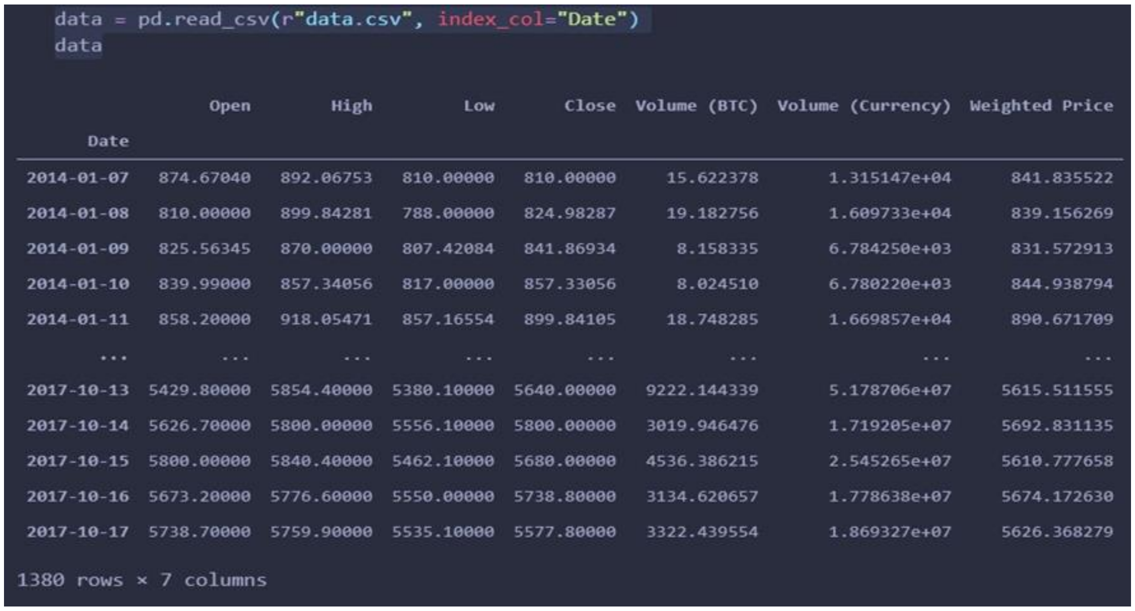
Import necessary libraries

```
# handling the data
import numpy as np
import systemcheck
import pandas as pd
from math import sqrt
from numpy import concatenate
from datetime import datetime

# transformation of the data
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import LabelEncoder
# model evaluation
from sklearn.metrics import mean_squared_error

# for data visualization
import seaborn as sns
import plotly.offline as py
import plotly.graph_objs as go
from matplotlib import pyplot as plt
py.init_notebook_mode(connected=True)
%matplotlib inline

# for creating and training RNN(LSTM)
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
 from tensorflow.keras.layers import LSTM
```

# Loading dataset

data = pd.read_csv(r"data.csv", index_col="Date")

data



```
data = pd.read_csv(r"data.csv", index_col="Date")
data

                  Open        High         Low       Close  Volume (BTC)  Volume (Currency)  Weighted Price
      Date
2014-01-07   874.67040   892.06753   810.00000   810.00000     15.622378       1.315147e+04      841.835522
2014-01-08   810.00000   899.84281   788.00000   824.98287     19.182756       1.609733e+04      839.156269
2014-01-09   825.56345   870.00000   807.42084   841.86934      8.158335       6.784250e+03      831.572913
2014-01-10   839.99000   857.34056   817.00000   857.33056      8.024510       6.780220e+03      844.938794
2014-01-11   858.20000   918.05471   857.16554   899.84105     18.748285       1.669857e+04      890.671709
       ...         ...         ...         ...         ...           ...                ...             ...
2017-10-13  5429.80000  5854.40000  5380.10000  5640.00000   9222.144339       5.178706e+07     5615.511555
2017-10-14  5626.70000  5800.00000  5556.10000  5800.00000   3019.946476       1.719205e+07     5692.831135
2017-10-15  5800.00000  5840.40000  5462.10000  5680.00000   4536.386215       2.545265e+07     5610.777658
2017-10-16  5673.20000  5776.60000  5550.00000  5738.80000   3134.620657       1.778638e+07     5674.172630
2017-10-17  5738.70000  5759.90000  5535.10000  5577.80000   3322.439554       1.869327e+07     5626.368279

1380 rows × 7 columns
```

Figure 3.5.5.1: Importing Datasets

#data analysis

data.info()

btc_WP = go.Scatter(x=data.index, y=data['Weighted Price'], name= 'Price')

py.iplot([btc_WP])

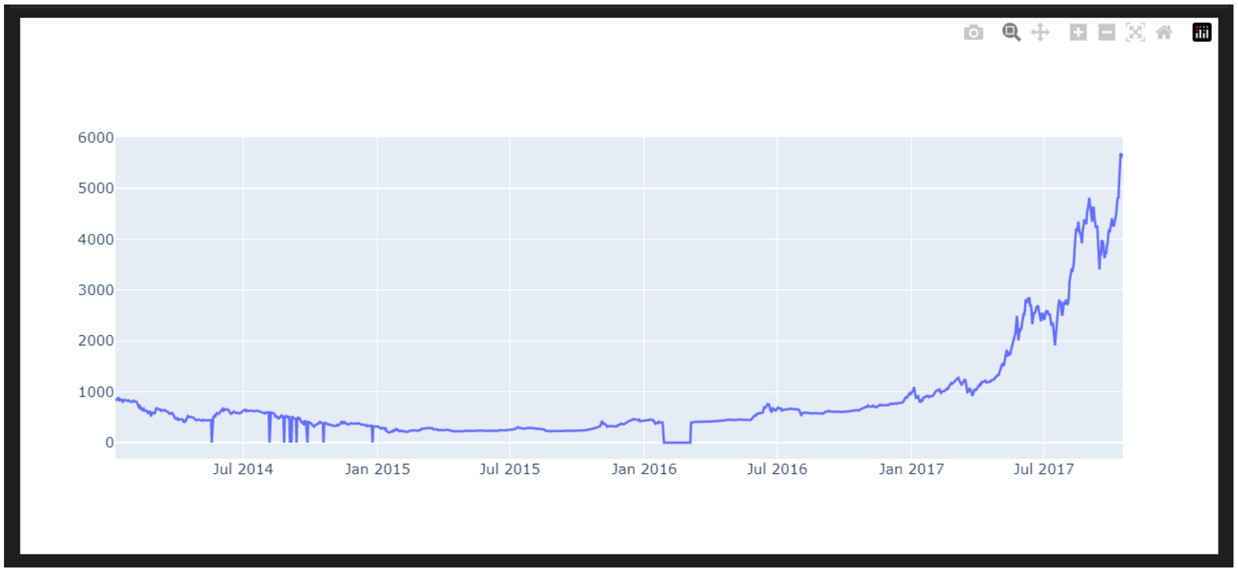


Figure 3.5.5.2: Data Analysis

```
#processing the values
data['Weighted Price'].replace(0, np.nan, inplace=True)
data['Weighted Price'].fillna(method='ffill', inplace=True)
btc_WP = go.Scatter(x=data.index, y=data['Weighted Price'], name= 'Price')
py.iplot([btc_WP])

#Using Weighted Price as a feature to train the LSTM model
#Use MinMaxScaler to normalize Weighted Price to range from 0 to 1
from sklearn.preprocessing import MinMaxScaler
values = data['Weighted Price'].values.reshape(-1,1)
values = values.astype('float32')
scaler = MinMaxScaler(feature_range=(0, 1))
scaled = scaler.fit_transform(values)

#Split the data 70% for training and 30% for testing
train_size = int(len(scaled) * 0.7)
test_size = len(scaled) - train_size
train, test = scaled[0:train_size,:], scaled[train_size:len(scaled),:]
print("train shape: ",train.shape,"\n","test shape: ",test.shape)

#Create function for creating dataset with look back
def create_dataset(dataset, look_back=1):
    dataX, dataY = [], []
    for i in range(len(dataset) - look_back):
        a = dataset[i:(i + look_back), 0]
        dataX.append(a)
        dataY.append(dataset[i + look_back, 0])
    print(len(dataY))
    return np.array(dataX), np.array(dataY)
```

#Generate dataset for trainX, trainY, testX, testY

```
look_back = 5
trainX, trainY = create_dataset(train, look_back)
testX, testY = create_dataset(test, look_back)
trainX[:5]
trainY[:5]
```

#Reshape X for model training

```
trainX = np.reshape(trainX, (trainX.shape[0], 1, trainX.shape[1]))
testX = np.reshape(testX, (testX.shape[0], 1, testX.shape[1]))
print("train shape: ",trainX.shape,"\n","test shape: ",testX.shape)
```

#Creating and training the LSTM model

```
model = Sequential()
model.add(LSTM(100, input_shape=(trainX.shape[1], trainX.shape[2])))
model.add(Dense(1))
model.compile(loss='mae', optimizer='adam')
history = model.fit(trainX, trainY, epochs=100, batch_size=100, validation_data=(testX, testY), verbose=1, shuffle=False)
```

#Plot line graph to show amount loss according the the epoch

```
plt.plot(history.history['loss'], label='train')
plt.plot(history.history['val_loss'], label='test')
plt.legend()
plt.show()
```

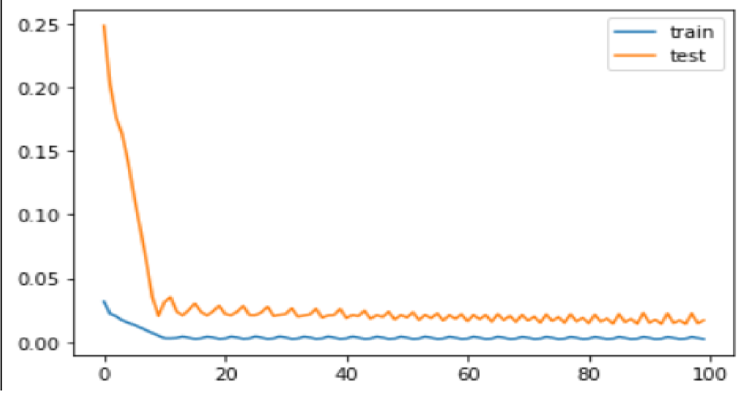


Figure 3.5.5.3: Line Graph

#Make prediction using textX and plotting line graph against testY

```
y_p1 = model.predict(testX)
plt.plot(y_p1, label='predict')
plt.plot(testY, label='true')
plt.legend()
plt.show()
```

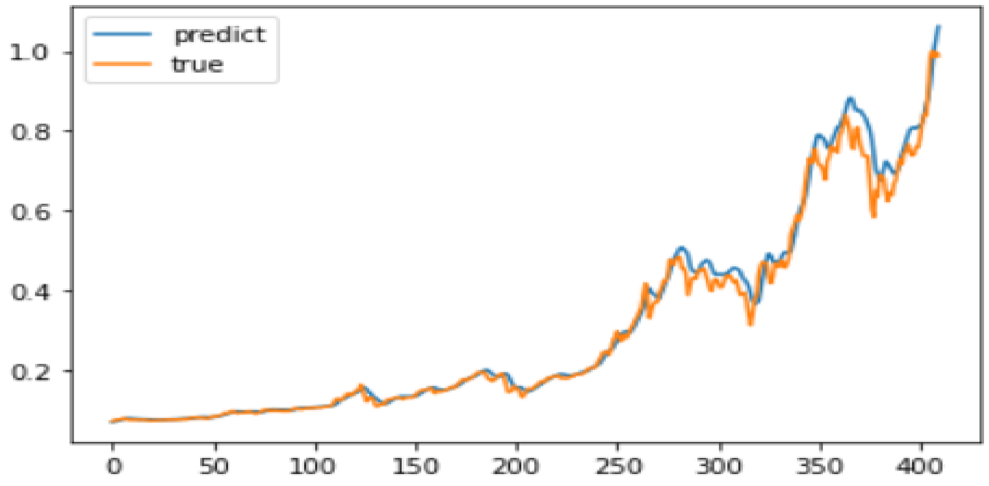


Figure 3.5.5.4: Line graph against testY

#Scaler Inverse Y back to normal value

```
y_p_inverse = scaler.inverse_transform(y_p1.reshape(-1, 1))
testY_inverse = scaler.inverse_transform(testY.reshape(-1, 1))
```

#evaluation

```
rmse = sqrt(mean_squared_error(testY_inverse, y_p_inverse))

 print('Test RMSE: %.3f' % rmse)
```

#Plot line graph with Y as USD

```
plt.plot(y_p_inverse, label='predict')
plt.plot(testY_inverse, label='actual', alpha=0.5)
plt.legend()
plt.show()
```

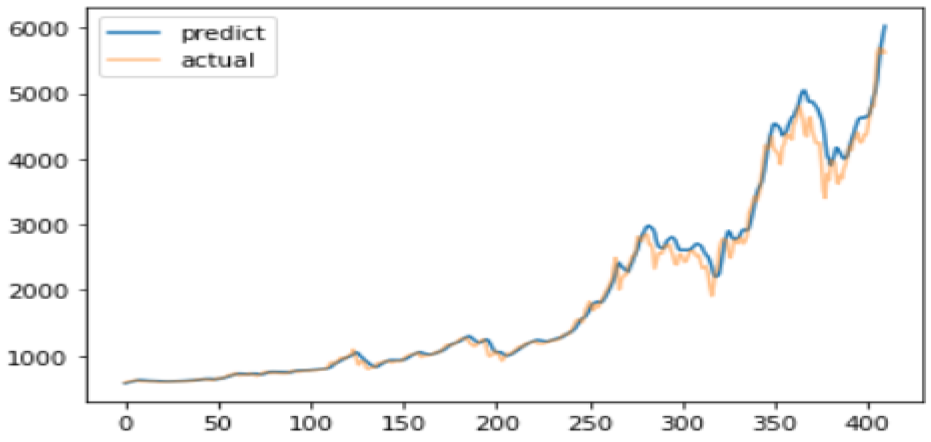


Figure 3.5.5.5: Line Graph with Y as USD

#Convert X to dates

```
predictDates = data.tail(len(testX)).index
```

#Reshape testY and y_pred for plotly

```
testY_reshape = testY_inverse.reshape(len(testY_inverse))
yhat_reshape = y_p_inverse.reshape(len(y_p_inverse))
```

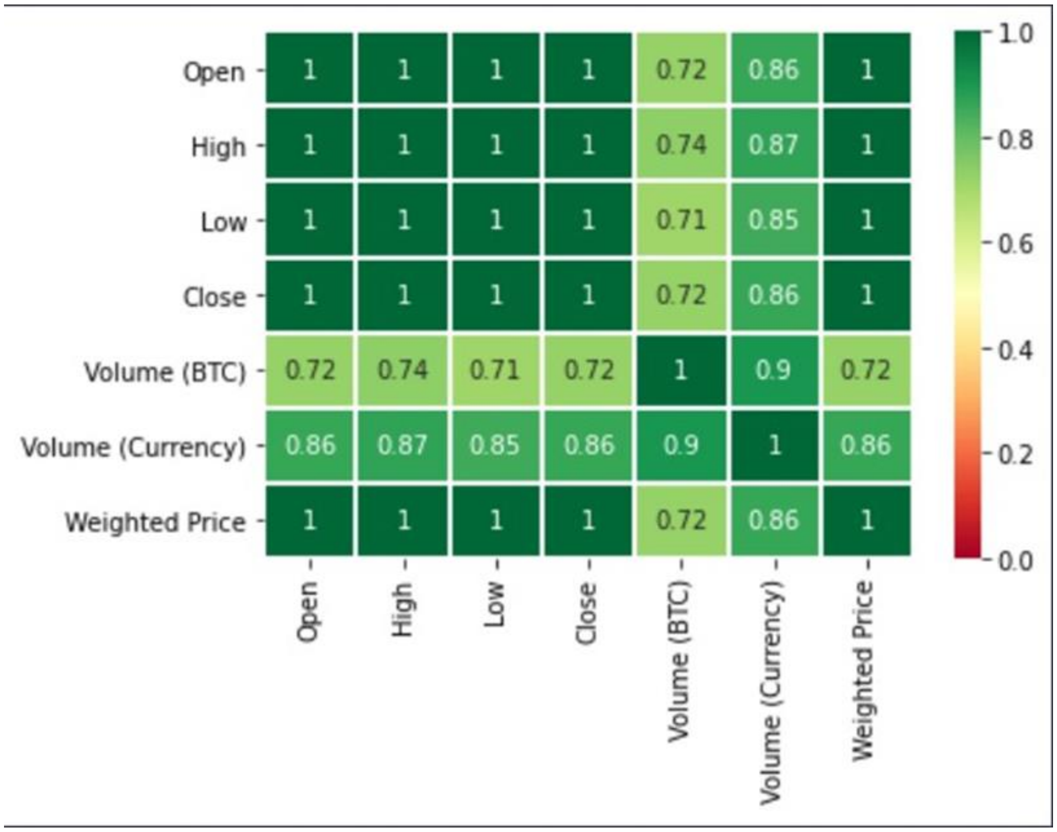
#Plot predicted and actual line graph with X=dates, Y=USD

```
actual_chart = go.Scatter(x=predictDates, y=testY_reshape, name= 'Actual Price')
predict_chart = go.Scatter(x=predictDates, y=yhat_reshape, name= 'Predict Price')
py.iplot([predict_chart, actual_chart])
```

Second-Model: Using additional features for model training

#Find corrleration in features to Weighted Price

```
sns.heatmap(data.corr(), annot=True, cmap='RdYlGn', linewidths=0.1, vmin=0)
```



	Open	High	Low	Close	Volume (BTC)	Volume (Currency)	Weighted Price
Open	1	1	1	1	0.72	0.86	1
High	1	1	1	1	0.74	0.87	1
Low	1	1	1	1	0.71	0.85	1
Close	1	1	1	1	0.72	0.86	1
Volume (BTC)	0.72	0.74	0.71	0.72	1	0.9	0.72
Volume (Currency)	0.86	0.87	0.85	0.86	0.9	1	0.86
Weighted Price	1	1	1	1	0.72	0.86	1

Figure 3.5.5.6: Corrleation Matrix

```
#Function to convert series to supervised learning
def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
  n_vars = 1 if type(data) is list else data.shape[1]
  df = pd.DataFrame(data)
  cols, names = list(), list()
  # input sequence (t-n, ... t-1)
  for i in range(n_in, 0, -1):
    cols.append(df.shift(i))
    names += [('var%d(t-%d)' % (j+1, i)) for j in range(n_vars)]
  # forecast sequence (t, t+1, ... t+n)
  for i in range(0, n_out):
    cols.append(df.shift(-i))
    if i == 0:
            names += [('var%d(t)' % (j+1)) for j in range(n_vars)]
        else:
            names += [('var%d(t+%d)' % (j+1, i)) for j in range(n_vars)]
    # put it all together
    agg = pd.concat(cols, axis=1)
    agg.columns = names
    # drop rows with NaN values
    if dropnan:
        agg.dropna(inplace=True)
     return agg

#Get all data values
values = data[['Weighted Price'] + ['Volume (BTC)'] + ['Volume (Currency)']].values
values = values.astype('float32')
 values[:5]

#Normalize features to range from 0 to 1
 scaler = MinMaxScaler(feature_range=(0, 1))
 scaled = scaler.fit_transform(values)

#Frame as supervised learning
 reframed = series_to_supervised(scaled, 1, 1)
 reframed.head()

#Drop unncessary columns
 reframed.drop(reframed.columns[[4,5]], axis=1, inplace=True)
 print(reframed.head())

#Split data to 70% training, 30% testing
 values = reframed.values
 n_train_hours = int(len(values) * 0.7)
 train = values[:n_train_hours, :]
```

```
test = values[n_train_hours:, :]
```

# split into input and outputs

```
train_X, train_y = train[:, :-1], train[:, -1]

test_X, test_y = test[:, :-1], test[:, -1]
```

# reshape input to be 3D [samples, timesteps, features]

```
train_X = train_X.reshape((train_X.shape[0], 1, train_X.shape[1]))

test_X = test_X.reshape((test_X.shape[0], 1, test_X.shape[1]))

print(train_X.shape, train_y.shape, test_X.shape, test_y.shape)
```

#**Training the LSTM model-2**

```
multi_model = Sequential()

multi_model.add(LSTM(100, input_shape=(train_X.shape[1], train_X.shape[2])))

multi_model.add(Dense(1))

multi_model.compile(loss='mae', optimizer='adam')

multi_history = multi_model.fit(train_X, train_y, epochs=100, batch_size=100,
validation_data=(test_X, test_y), verbose=1, shuffle=False)
```

#Plot line graph to show amount loss according the the epoch

```
plt.plot(multi_history.history['loss'], label='multi_train')

plt.plot(multi_history.history['val_loss'], label='multi_test')

plt.legend()

plt.show()
```

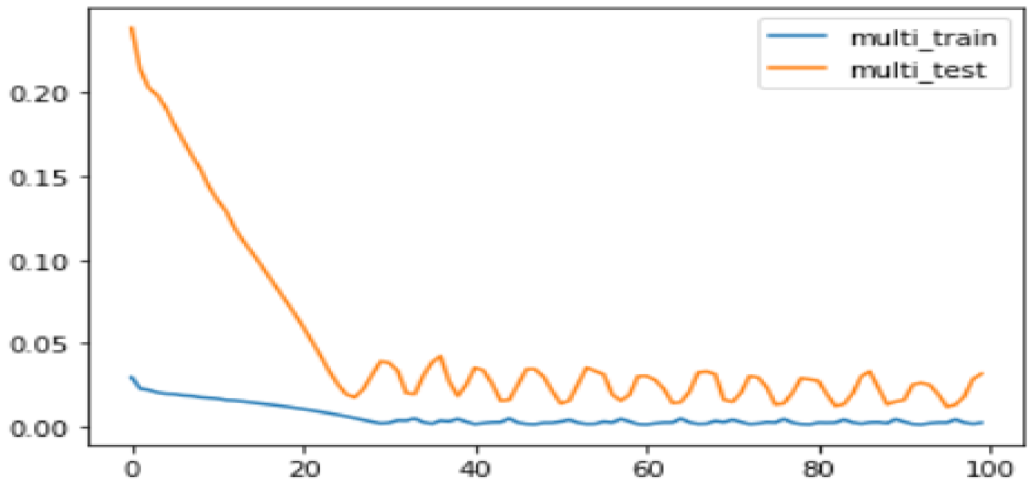


Figure 3.5.5.7: Line Graph to show amount loss

#Make prediction using textX and plotting line graph against testY

```
yhat = multi_model.predict(test_X)

plt.plot(yhat, label='predict')

plt.plot(test_y, label='true')

plt.legend()

plt.show()
```

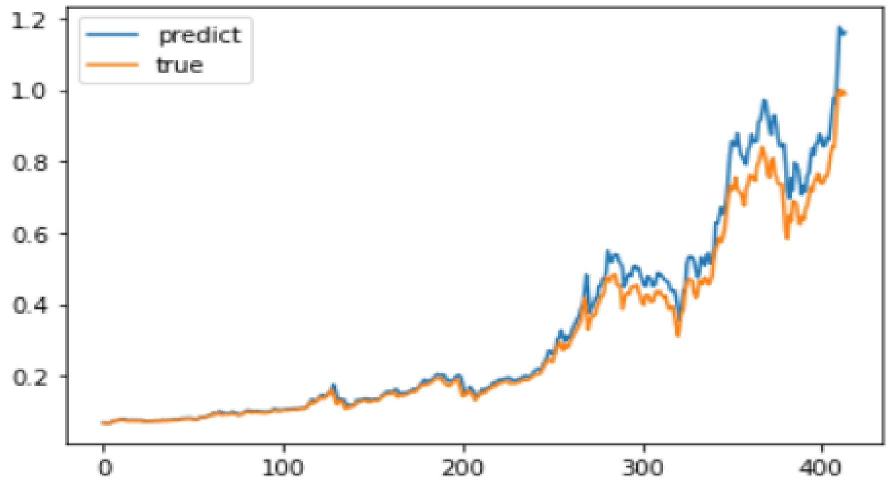


Figure 3.5.5.8: Line Graph against testY

#Scaler Inverse Y back to normal value

```
test_X = test_X.reshape((test_X.shape[0], test_X.shape[2]))
```

# invert scaling for forecast

```
inv_yhat = concatenate((yhat, test_X[:, 1:]), axis=1)

inv_yhat = scaler.inverse_transform(inv_yhat)

inv_yhat = inv_yhat[:,0]
```

# invert scaling for actual

```
test_y = test_y.reshape((len(test_y), 1))

inv_y = concatenate((test_y, test_X[:, 1:]), axis=1)

inv_y = scaler.inverse_transform(inv_y)

inv_y = inv_y[:,0]
```

#RMSE

```
rmse = sqrt(mean_squared_error(inv_y, inv_yhat))

print('Test RMSE: %.3f' % rmse)
```

Final result

Plot line graph with actual price, predicted price with feature Weighted Price, predicted price with features Volume and Weighted Price

```
actual_chart = go.Scatter(x=predictDates, y=inv_y, name= 'Actual Price')

multi_predict_chart = go.Scatter(x=predictDates, y=inv_yhat, name= 'Multi Predict Price')

predict_chart = go.Scatter(x=predictDates, y=yhat_reshape, name= 'Predict Price')

py.iplot([predict_chart, multi_predict_chart, actual_chart])
```

CHAPTER 4

RESULTS AND DISCUSSION

CHAPTER 4
RESULTS AND DISCUSSION

4.1.1 Performance metrics

In previous methods performance metrics like accuracy, precision, recall, auc-roc, etc. are used to evaluate the performance of models. Classification metrics are used for Classification models and Regression metrics are used for Regression models.

In our project we are using Regression models. They are Linear regression, Recurrent Neural Network (RNN), and Long Short Term Memory (LSTM) Machine Learning algorithms. Evaluating their performance using evaluation metrics like , Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE).

- **Mean Absolute Error (MAE)**

$$\mathrm{MAE} = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|$$

where $\mathbf{y_i}$ is the actual expected output and $\mathbf{\hat{y}_i}$ is the model's prediction.

It is the simplest evaluation metric for a regression scenario and is not much popular compared to the following metrics.

- **Mean Squared Error (MSE)**

$$\mathrm{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

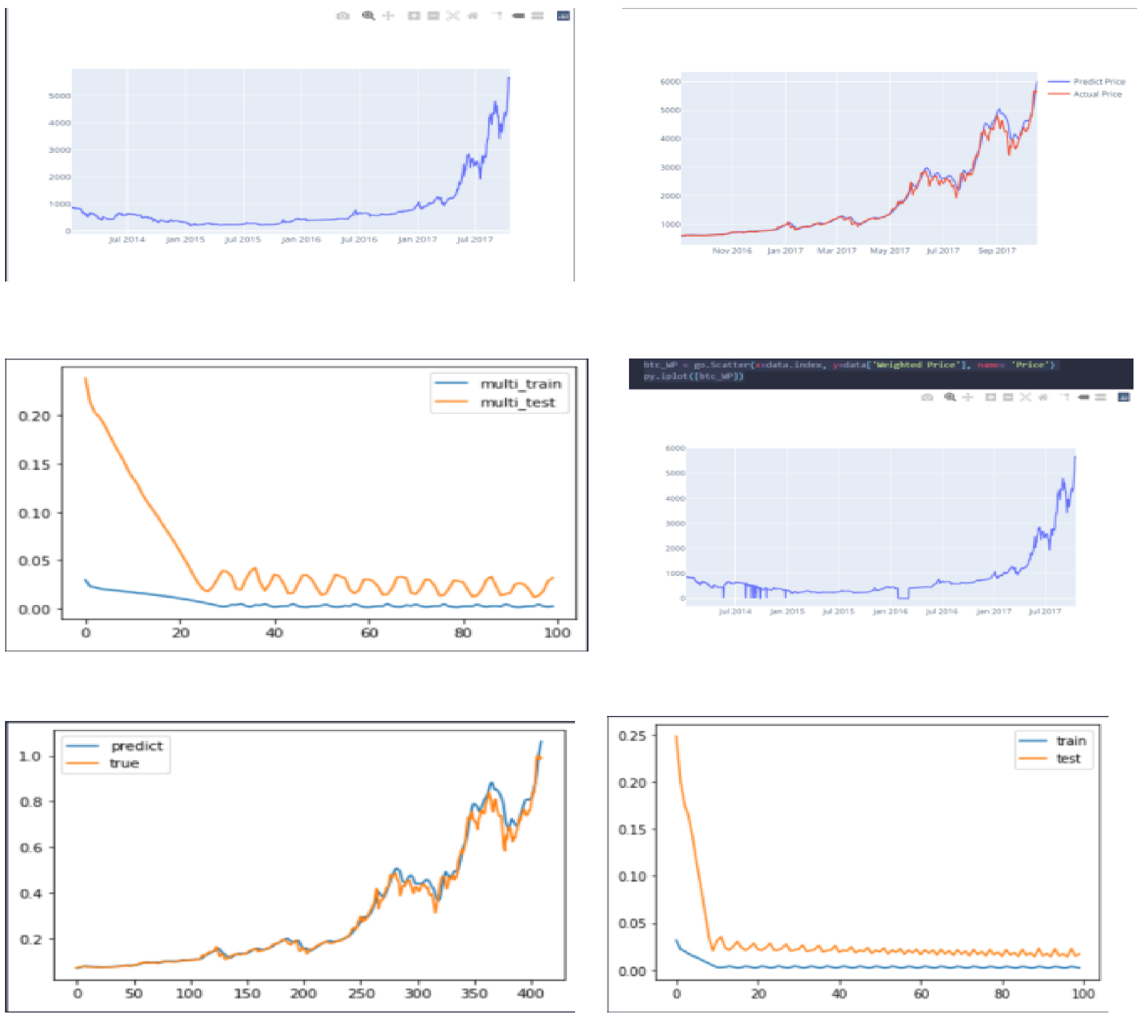
Here, the error term is squared and thus **more sensitive to outliers** as compared to Mean Absolute Error (MAE).

- **Root Mean Squared Error (RMSE)**

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y}_j)^2}$$

Since MSE includes squared error terms, we take the square root of the MSE, which gives rise to Root Mean Squared Error (RMSE).

EXPLORATORY DATA



```
btc_WP = go.Scatter(x=data.index, y=data['Weighted Price'], name= 'Price')
py.iplot([btc_WP])
```

Figure 4.1.1.1: Output Screen Line Graph Analysis

```
actual_chart = go.Scatter(x=predictDates, y=testY_reshape, name= 'Actual Price')
predict_chart = go.Scatter(x=predictDates, y=yhat_reshape, name= 'Predict Price')
py.iplot([predict_chart, actual_chart])
```

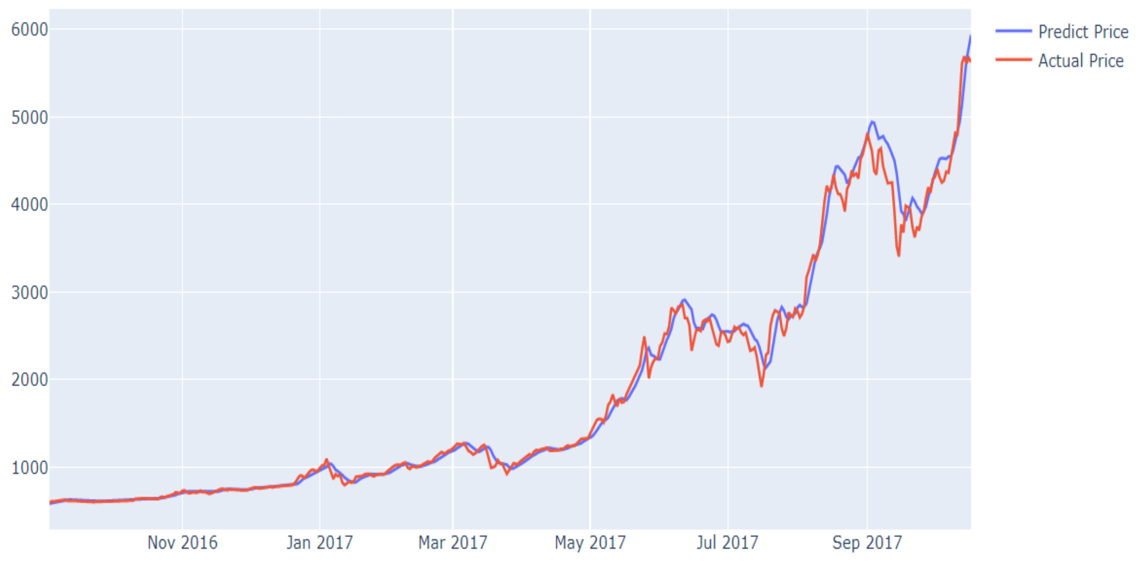


Figure 4.1.1.2: Output Screen Multiple Predict Vs True

```
actual_chart = go.Scatter(x=predictDates, y=inv_y, name= 'Actual Price')
multi_predict_chart = go.Scatter(x=predictDates, y=inv_yhat, name= 'Multi Predict Price')
predict_chart = go.Scatter(x=predictDates, y=yhat_reshape, name= 'Predict Price')
py.iplot([predict_chart, multi_predict_chart, actual_chart])
```

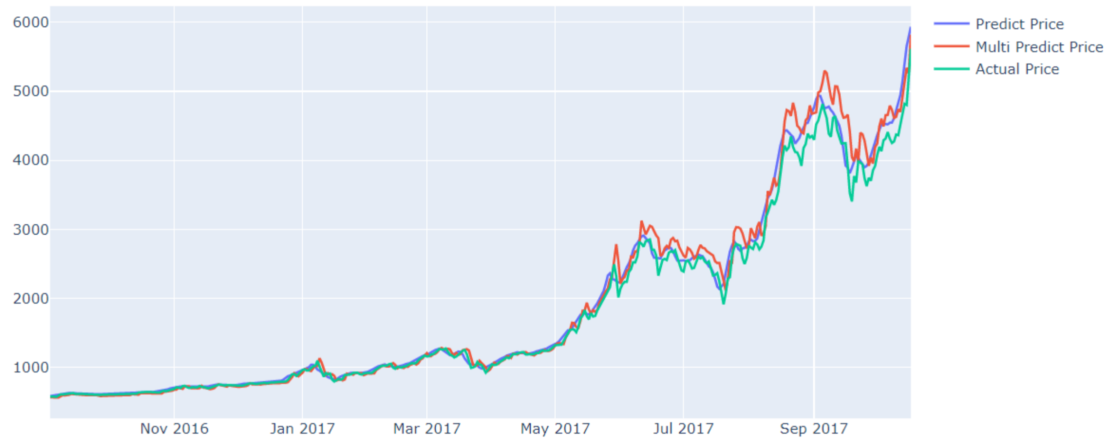


Figure 4.1.1.3: Output Screen Final Result

CHAPTER 5
CONCLUSION

CHAPTER 5
CONCLUSION AND FUTURE ENHANCEMENTS

Conclusion

A system that aims to provide an accurate prediction of bitcoin prices has been developed. The system makes optimal use of Linear Regression, Forest regression. The efficiency of the algorithm has been further increased with use of Neural networks. The system will satisfy customers by providing accurate output and preventing the risk of investing in the wrong house. Additional features for the customer’s benefit can also be added to the system without disturbing its core functionality.

Future Enhancements

The accuracy of the system can be improved. Several more crypto currencies can be included in the system if the size and computational power increases of the system. Furthermore, we can integrate different UI/UX methodology for better visualization of the results in a more interacting way using Augmented Reality . Also, a learning system can be created which will gather users’ feedback and history so that the system can display the most suitable results to the user according to his preferences.

CHAPTER 6
REFERENCES

CHAPTER 6
REFERENCES

[1]. Project Based Learning: Predicting Bitcoin Prices using Deep Learning‖ S. Yogeshwaran ; PiyushMaheshwari; ManinderJeet Kaur ; Amity University Dubai Dubai, UAE; IEEE 2019

[2]. Predicting the Price of Bitcoin Using MachineLearning‖ Sean McNally ; Jason Roche ; Simon Caton; Ireland, Dublin , IEEE 2018

[3]. Bitcoin Volatility Forecasting with a Glimpse into Buy and Sell Orders‖ Tian Guo ; Albert Bifet ; Nino AntulovFantulin ; IEEE 2018

[4]. F. Andrade de Oliveira, L. Enrique ZÃ¡rate and M. de Azevedo Reis; C. NeriNobre, The use of artificial neural networks in the analysis and prediction of stock prices,‖ in IEEE International Conference on Systems, Man, and Cybernetics, 2011, pp. 2151-2155.

[5]. Bitcoin Cost Prediction using Deep Neural Networ Technique‖ Kalpanasonika ,Sayasri S , Vinothini , SugaPriya , IEEE 2018

[6]. An improved K-Means clustering algorithm‖ ;Juntao Wang, Xiaolong Su, IEEE 2017

[7]. Application of Random Forest Algorithm on Feature Subset Selection and Classification and Regression‖ ;Jitendra Kumar Jaiswal, Rita Samikannu IEEE 2017

[8]. Improved Random Forest for Classification‖ ; Angshuman Paul, Dipti Prasad Mukherjee, Senior Member, IEEE, Prasun Das, Abhinandan Gangopadhyay, Appa Rao Chintha and SaurabhKundu IEEE2018

Github Link

https://github.com/KeerthiGuntaka/-BITCOIN-PRICE-PREDICTION-USING-MACHINE-LEARNING-ALGORITHMS